Universidad Zaragoza

Trabajo Fin de Grado

Detección de Ataques de Integridad en Redes Inteligentes mediante el uso de Técnicas de Aprendizaje Automático

Detection of Integrity Attacks to Smartgrids using Machine Learning Techniques

Autor

Raúl Javierre Cabrero

Directores

Simona Bernardi
José Javier Merseguer Hernáiz

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2021

Agradecimientos

Me gustaría empezar dando las gracias tanto a Simona como a José por haber depositado su confianza en mí y por haberme apoyado desde que empecé a trabajar en el proyecto. Sois unos grandísimos docentes e investigadores, pero también unas grandísimas personas. Espero haber estado a la altura y haberos sido de ayuda.

También quiero dar las gracias a todos los compañeros y compañeras de la carrera con los que he compartido miedos, alegrías, tristezas y (sobre todo) muchos buenos momentos durante estos cuatro años. Echo mucho de menos pasar días enteros con vosotros por la EINA (y eso que nos quejábamos mucho por tener que quedarnos todo el día ahí). No quiero mencionar a nadie, porque sois muchos y seguro que me olvido de alguien... Pero ya sabéis quiénes sois.

Gracias a mis amigos de Huesca por conseguir hacerme desconectar cada vez que lo necesitaba. Habéis sido mi brújula durante toda la carrera, pero especialmente este último año, que he podido hacer más planes con vosotros. Por cierto, me alegro mucho de que los domingos de excursión ya se hayan convertido en una tradición.

Gracias a Alba por ser un pilar fundamental en mi vida... Ya van más de 6 años a tu lado y cada día consigues que te quiera un poquito más. Dentro de nada serás una médico excelente, que no te quepa la menor duda.

Gracias a mis padres por arroparme tantas y tantas veces durante toda la vida. Perdón por mis enfados, por mis días grises... De vosotros aprendí que sin trabajo y sin esfuerzo no se consigue nada.

Gracias, tato, por ser mi referente desde el día que nací. Ahora los dos somos ingenieros informáticos, y parece que también voy a terminar contando en las aulas lo que sé de este mundillo (tú sabes un poquito más que yo ahora mismo, pero no te confíes). Gracias por ser mi escudo, mi confidente, un hombro en el que llorar... Gracias por cuidarme tanto en todo momento. Si tuviera que volver a elegir un hermano te elegiría 1 y 1000 veces a ti.

Resumen

Las redes inteligentes (*smart grids*) no están exentas de ser atacadas; de hecho, son un gran atractivo para los ciberdelincuentes ya que pueden obtener beneficios económicos importantes falsificando sus consumos energéticos.

En el presente trabajo se desarrollan y evalúan detectores de ataques de integridad sobre estas redes. Además, se llevan a cabo ataques de integridad sobre conjuntos de datos reales para poner a prueba los detectores implementados.

Tanto los conjuntos de datos, como los escenarios de ataque, como los detectores están integrados en un *framework*. El propósito de este *framework* es facilitar la inclusión de nuevos conjuntos de datos, escenarios de ataque y detectores para así proporcionar a futuras programadoras y programadores una herramienta sencilla con la que puedan experimentar e investigar en este área. Este *framework* está desarrollado en el lenguaje de programación *Python* y se encuentra dividido en módulos, probados con más de 1000 *tests*. Cada uno de éstos módulos representa una actividad a realizar: preprocesar los datos, implementar detectores, lanzar los experimentos y analizar los resultados.

Los detectores se basan en distintas técnicas: modelos autorregresivos, *clústering*, árboles de decisión, comparación de distribuciones de frecuencia relativa y aprendizaje profundo. Los escenarios de ataque también son variopintos; los hay basados en valores medios del pasado, otros que se basan en permutar valores dependiendo del precio del kWh en ese instante, otros que generan consumos sintéticos solucionando problemas de programación lineal y otros que multiplican todos los registros por un valor entre 0 y 1.

La ejecución de los experimentos es computacionalmente muy costosa. Se requiere aproximadamente un mes de cómputo en un *clúster de supercomputación* para finalizar la experimentación utilizando todos los conjuntos de datos, escenarios de ataque y detectores considerados en este trabajo si se utilizan 1000 contadores energéticos por cada conjunto de datos.

Se ha desarrollado una interfaz web interactiva que facilita la visualización de los resultados de los experimentos filtrando por conjunto de datos y/o escenario de ataque. Esto permite al o a la analista evaluar el rendimiento de cada detector de manera global o específica a cada escenario de ataque.

Se han alcanzado todos los objetivos que se plantearon en la propuesta de este Trabajo Fin de Grado. Además, sin estar finalizado en su totalidad, un trabajo [1] relacionado con este proyecto de investigación ha sido aceptado por el comité del European Dependable Computing Conference (EDCC) para ser presentado en septiembre de este mismo año.

Índice

1. Introducción **1**
1.1. Metodología 2
1.2. Estructura del documento 3

2. Bases de la experimentación **4**
2.1. Conjuntos de datos utilizados 4
2.2. Escenarios de consumo 5
2.3. Detectores 7

3. *Framework* desarrollado **14**
3.1. Módulo de preprocesado 15
3.2. Módulo de detectores 15
3.3. Módulo de lanzamiento de los experimentos 17
3.4. Módulo de análisis de los experimentos 19
3.5. Módulo de infraestructura de *testing* 23

4. Experimentación **24**
4.1. Conclusiones de la experimentación 25

5. Conclusiones **27**
5.1. Lecciones aprendidas 27
5.2. Trabajo futuro 27
5.3. Recomendaciones 28

A. Estado del arte y herramientas **34**
A.1. Las redes eléctricas inteligentes 34
A.2. Herramientas de minería de datos 35

B. Gestión del proyecto **37**
B.1. Gestión de esfuerzos 37
B.2. Gestión de riesgos 42

C. Conjuntos de datos utilizados **44**
C.1. ISSDA CER (Electricidad) 44
C.2. ISSDA CER (Gas) 46

D. *Framework* desarrollado **48**
D.1. Módulo de detectores 48
D.2. Módulo de lanzamiento de los experimentos 51
D.3. Módulo de análisis de los experimentos 52
D.4. Módulo de infraestructura de *testing* 55

E. Resultados **58**

1. Introducción

Las redes inteligentes son sistemas ciberfísicos, que suponen una evolución tecnológica importante frente a las redes energéticas tradicionales. Aun estando bien implantadas a día de hoy, se prevé que su uso vaya a aumentar en los próximos años. Las redes inteligentes, a pesar de ser más resilientes y seguras que la red convencional, están sometidas a múltiples ciberataques. El objetivo del presente trabajo es investigar acerca de aquellos ciberataques encaminados a la manipulación del consumo. Su importancia viene motivada por las pérdidas económicas que suponen estos ciberataques, que en el caso de la sustracción de electricidad alcanza los 96 mil millones de dólares anuales a nivel mundial [2].

Los objetivos concretos del proyecto son:

- Estudiar, en el ámbito de las redes inteligentes, los diferentes tipos de ciberataques, de detectores de los mismos y de conjuntos de datos apropiados que existen en la literatura.
- Implementar los detectores y los escenarios de ataque seleccionados y experimentar con los mismos.
- Desarrollar un entorno de experimentación (*framework*) para facilitar el análisis comparativo de la calidad de los detectores y de la eficacia de los ataques.
- Evaluar, con la ayuda del *framework*, la calidad de los detectores implementados, bajo distintos escenarios de ataque, y la eficacia de los ciberataques, en términos de beneficio económico.
- Proponer nuevos detectores de mejor calidad.
- Desarrollar una batería de pruebas para verificar el correcto funcionamiento del *framework*.

Para alcanzar estos objetivos se ha experimentado con datos reales, y en la experimentación ha sido necesario ponerse tanto en la piel del atacante como en la del ingeniero/ingeniera de seguridad. Por lo que este trabajo ha estado guiado por preguntas como las siguientes: ¿cómo puedo pagar menos de lo que consumo sin que lo detecten? ¿Cómo puedo detectar escenarios de consumo anómalos?

El proyecto desarrolla un *framework* en Python [3] que permite abordar los objetivos y comprende:

- Un *módulo de preprocesado de datos*, que a partir de información de consumo energético genera datos de entrenamiento y de validación.
- Un *módulo de detectores*, cuyo propósito es descubrir anomalías en los datos de consumo preprocesados.

- Un *módulo de experimentación*, que utiliza los detectores, además de los datos de entrenamiento y validación, para generar resultados.
- Un *módulo de análisis*, que convierte los resultados de la experimentación en información útil para que el ingeniero o ingeniera pueda tomar decisiones con respecto a posibles ciberataques y/o detectores.
- Un módulo de infraestructura de *testing*, cuya función es verificar que el código implementado es correcto.

El *framework* se encuentra disponible en el repositorio privado de GitHub https://github.com/DiasporeUnizar/diaspore/. Además, el mismo incluye la documentación necesaria para reproducir todos los experimentos, y la información requerida para extenderlo con nuevos detectores, conjuntos de datos y escenarios de ataque.

1.1. Metodología

El proyecto se ha desarrollado siguiendo principios de las metodologías ágiles [4]. En particular, entregar *software* útil periódicamente, trabajar en cortos periodos de tiempo de la misma duración (iteraciones), refinar el proceso continuamente y priorizar la comunicación síncrona estableciendo reuniones frecuentemente (sin olvidar la comunicación asíncrona a través de documentación e informes, siempre y cuando se considere que son necesarios y útiles para el futuro).

En lo que se refiere a este proyecto, se ha trabajado en iteraciones de una semana, estableciendo una reunión cada miércoles a las 11:00, en la que se analizaba con los directores lo que se había realizado durante esta iteración y se proponían las tareas a realizar para la venidera. Esas tareas se añadían a un tablero *Kanban* de tres columnas *To do, In progress y Done* en *GitHub*, proporcionando así una visión del estado actual de la iteración en todo momento. Además, dentro de cada tarea, la plataforma *GitHub* proporciona un mecanismo de hilo de mensajes, que se ha utilizado para poder comunicar rápidamente cualquier duda, sugerencia o comentario.

En el apéndice B se detallan también aspectos relacionados con la gestión de esfuerzos y de riesgos, inherentes a cualquier proyecto, que, por supuesto, son independientes de la metodología adoptada.

Se decidió seguir esta metodología en contraposición a una metodología tradicional dirigida por planificación desde el inicio puesto que el trabajo a desarrollar pertenece a un dominio complejo e innovador, en el que no se pueden prever todas las necesidades que van a ir apareciendo conforme se va avanzando en él.

Los diferentes experimentos han sido lanzados en el *clúster de supercomputación "Hermes" del I3A*, con 8 núcleos, un Intel Xeon Gold 6254 CPU, 32GB RAM y 1 TB SSD, siendo Ubuntu 20.04 el sistema operativo.

1.2. Estructura del documento

La sección 2 introduce las bases de la experimentación desarrollada. La sección 3 describe el *framework*, detallando todos y cada uno de los módulos que lo constituyen. La sección 4 describe el proceso de experimentación, además de los resultados obtenidos con los experimentos. La sección 5 resume las lecciones aprendidas, el trabajo que queda pendiente para tratar de seguir mejorando la seguridad en estas redes, así como algunas recomendaciones. Para facilitar la lectura de la memoria, la información más detallada se ha incluido en apéndices complementarios a los que se hace referencia desde el texto principal. Estos apéndices pueden ser obviados por el lector si no precisa de la información allí proporcionada.

2. Bases de la experimentación

En este apartado se detallan los tres elementos base de la experimentación: los conjuntos de datos (sección 2.1), los escenarios de consumo (sección 2.2) y los detectores (sección 2.3).

Con la experimentación se busca analizar la calidad y el rendimiento de detectores de ataques de integridad a redes inteligentes y llevar a cabo un análisis comparativo de los mismos.

Para analizar los detectores ha sido necesario obtener datos de consumo. Se decidió obtener datos de consumo provenientes de redes inteligentes reales. El problema que surgió es que estos datos sólo representan escenarios normales de consumo, pues se asume que no presentan ataques de integridad. Este problema se solucionó generando "escenarios sintéticos". Un escenario sintético representa un escenario de ataque, pues es una manipulación de los datos de consumo originales (*escenario normal*) con la finalidad de conseguir un beneficio económico. Para generar estos escenarios sintéticos se han estudiado algunos que han sido propuestos en la literatura anteriormente.

En resumen, para poder llevar a cabo la experimentación es necesario:

- Obtener datos de consumo provenientes de redes inteligentes. Estos datos se consideran libres de ataques (escenario "normal").
- Desarrollar diferentes escenarios de consumo a partir de los datos obtenidos. Esto es, generar escenarios sintéticos a partir de los escenarios obtenidos de las fuentes de datos.
- Implementar varios detectores, que deberán distinguir entre los datos sin modificar (normales) y los datos sintéticos (anómalos).

2.1. Conjuntos de datos utilizados

Para adecuarse al contexto de las redes inteligentes se decidió utilizar datos de consumo cuyo origen fuese una red inteligente real. En concreto, se han utilizado dos conjuntos de datos, ambos proporcionados por el *Irish Social Science Data Archive (ISSDA)* de la Comisión Irlandesa de regulación energética (*Commission for Energy Regulation* – CER) [5].

Para obtenerlos es necesario realizar una solicitud de uso. En este caso se envió una petición de uso para propósitos de investigación. Un conjunto recoge los datos de consumo de electricidad (ISSDA CER-Electricity) de 6.445 contadores y el otro conjunto recoge los datos de consumo de gas (ISSDA CER-Gas) de 1.576 contadores. El muestreo realizado recoge datos durante varias decenas de semanas cada 30 minutos, en kWh.

Los ficheros de datos de uso contienen una línea por lectura, y cada lectura consta de: identificador del contador, un *timestamp* y la cantidad de energía (en kWh) consumida desde la última lectura. En la información suministrada, además de los datos de uso, se incluye información relativa a qué clasificación

pertenece cada contador, puesto que, por ejemplo, hay algunos contadores pertenecientes a hogares y otros a pequeñas y medianas empresas.

Antes de su uso se han tenido que preprocesar los datos puesto que: a) están incompletos, no todos los contadores tienen registros cada media hora; y b) tienen distinta distribución, los ficheros de gas están separados por semanas, mientras que los de electricidad no. Además, la transformación de la información a un formato homogéneo y definido mediante su preprocesamiento supone un buen punto de partida, ya que facilita la inclusión de nuevos conjuntos de datos al plan de experimentos al definir un formato común.

En el apéndice C se describen en detalle los dos *datasets* utilizados.

2.2. Escenarios de consumo

Por un lado se define el escenario "normal" como un escenario de consumo registrado por un contador sin falsificaciones. Por otro lado, se definen los escenarios de consumo que representan ataques, es decir, manipulaciones del consumo registrado por un contador. Estos últimos son construidos a partir de modificaciones del escenario "normal", y su objetivo es generar un beneficio, en términos económicos, a aquellos que los ejecuten. En la Figura 7 puede visualizarse una estimación de los beneficios económicos para cada uno de los ataques. En este trabajo se han definido diferentes escenarios de consumo con la finalidad de poner a prueba los detectores de anomalías.

2.2.1. Normal

El escenario normal contiene los datos tal y como son. No se realiza ningún tipo de modificación en los mismos. Se asume que los *datasets* utilizados no presentan ataques de integridad, pese a que sí presentan anomalías como largos periodos de tiempo sin hacer uso del contador, o periodos de tiempo en los que se hace un uso elevado. Sin embargo, son escenarios que entran dentro de la normalidad del día a día de los seres humanos, como semanas de vacaciones, fiestas en casa, etc. Consecuentemente, los detectores implementados no deberían identificar como anómalos los consumos registrados en un escenario normal, puesto que generarían falsos positivos.

2.2.2. FDI_X

El tipo de ataque *False Data Injection* (FDI) [6] puede dar lugar a diferentes escenarios, puesto que puede tener variantes dependiendo del valor que se le proporcione al parámetro "X". Este ataque consiste en multiplicar todos y cada uno de los registros de un contador por *X/100*, haciendo que se pague un X % de lo que realmente se ha consumido. En los experimentos llevados a cabo en este trabajo se han considerado 5 escenarios distintos: FDI_0, FDI_5, FDI_10, FDI_20 y FDI_30.

2.2.3. Swap

El ataque *swap* se basa en que se paga más o menos dependiendo de la hora de consumo. Por ejemplo, 1 *kWh* puede ser mucho más barato si se consume a las 03:00 de la madrugada (horas valle) que a las 16:00 de la tarde (horas pico). En este trabajo se han definido dos marcos horarios:

- Horas valle: periodo desde las 00:00 hasta las 09:00.
- Horas pico: periodo desde las 09:00 hasta las 24:00.

Teniendo en cuenta esta distinción de horas, la idea principal del ataque es intercambiar en el eje temporal los 18 mayores consumos de las horas pico por los 18 menores consumos de las horas valle.

La utilidad de este ataque reside en que es más probable que se consuma más electricidad o gas en las horas pico que en las horas valle, puesto que en las horas valle, la mayor parte de la población está durmiendo tal y como se han definido las dos franjas consideradas en este trabajo.

2.2.4. RSA

El escenario de ataque *Random Scale Attack* [7] (RSA_α_β) se obtiene multiplicando cada uno de los registros de un contador por un valor aleatorio, generado por una distribución uniforme en el intervalo [α, β] con $0 < \alpha < \beta$. En los experimentos llevados a cabo en este trabajo se han considerado tres variantes: RSA_0.5_1.5, RSA_0.25_1.1 y RSA_0.5_3. Comportamientos de ataque como el RSA_0.5_3 declaran más *kWh* del que realmente consumen. Esto puede ser interesante si se le da la vuelta al razonamiento, puesto que en las redes inteligentes se puede aportar energía, por ejemplo vía paneles solares. Por tanto, se podrían realizar ataques como este, consistentes en reportar más energía de la que realmente se está aportando a la red. Sin embargo, se deja como trabajo futuro analizar detectores para este tipo de ataques considerando *datasets* con datos de energía producida por un cliente.

2.2.5. Avg y Min-Avg

Los escenarios de ataque *Average* (Avg) y *Min-Avg* [7] se basan en el promedio semanal de los consumos registrados por un contador. El objetivo es obtener un beneficio y, al mismo tiempo, evitar ser detectado por el detector "MinAverage" [8], que se describe en el apartado 2.3.1.

En el escenario *Average*, el valor del consumo promedio de una semana se utiliza para sustituir los consumos de esa semana de manera similar a lo descrito en el apartado 2.2.4, es decir, multiplicando dicho valor por un número aleatorio generado por una distribución uniforme en el intervalo [0.5, 1.5].

El escenario de ataque *Min-Average* permite conseguir el máximo beneficio pasando desapercibido al detector "MinAverage", que marca como anómalas aquellas semanas cuyo consumo promedio es menor que la semana de entrenamiento de menor consumo promedio. El ataque se basa en obtener el mínimo de

los promedios de consumo de las semanas de entrenamiento para generar datos de semanas cuyo consumo promedio siempre sea mayor que ese. Para tratar de obtener el máximo beneficio como atacante, la generación de datos se ha planteado como un problema de programación lineal de minimización, en el que además se tiene en cuenta el valor del precio dependiendo de la franja horaria (horas valle/horas pico).

2.3. Detectores

En este apartado se documentan los detectores implementados en el proyecto. Su misión es encontrar escenarios anómalos (ver sección 2.2), es decir, detectar escenarios de consumo distintos al escenario normal. En la Tabla 1 se listan todos los detectores implementados en este trabajo, indicando: su nombre, la técnica de aprendizaje utilizada y una referencia si su desarrollo ha venido inspirado de la literatura.

Nombre	**Técnica**	**Ref.**
MinAverage	Uso del mínimo de los promedios de consumo en el pasado	[8]
ARIMA	Auto-Regressive Integrated Moving Average	[7]
ARIMAX	Auto-Regressive Integrated Moving Average with eXplanatory variable	
PCADBSCAN	Principal Component Analysis, clústering	[7]
KMeans	Clústering	
MiniBatchKMeans	Clústering	
FisherJenks	Clústering	
IsolationForest	Bosque de árboles de decisión	
KLD JSD	Comparación de distribuciones de frecuencia relativa	[7]
TEG	Time-Evolving Graphs, comparación de distribuciones de frecuencia relativa	
NN	Deep learning, clasificación	[6]

Tabla 1: Detectores considerados en el proyecto.

2.3.1. Uso del mínimo de los promedios de consumo

El diseño del detector *MinAverage* [8] es sencillo. A partir de una serie temporal de datos de consumo registrados por un contador durante un período de

entrenamiento de n semanas, el modelo de predicción de este detector se basa en calcular el consumo promedio de cada semana de entrenamiento y el mínimo m de los promedios. El detector marca como semanas anómalas aquellas cuyo promedio de consumo es menor que m.

El detector tiene desventajas claras, puesto que si para entrenar se le ha proporcionado una semana en la que no se ha hecho uso del contador (puede haber motivos, como, por ejemplo, semanas de vacaciones en las que no se reside en el hogar), será incapaz de encontrar anomalías en las semanas de validación ya que ninguna semana tendrá un promedio de consumo inferior a cero. Esto daría lugar a la existencia de muchos falsos negativos. Otra desventaja es que también puede dar lugar a muchos falsos positivos en semanas de validación en las que no se haya consumido energía si se ha entrenado con semanas en las que siempre se ha consumido energía.

2.3.2. Modelos autorregresivos

Se han implementado dos detectores basados en modelos autorregresivos: *ARIMA* [7], y la variante *ARIMAX*, que, a diferencia del primero, tiene en cuenta la periodicidad diaria de consumo.

ARIMA. El detector ARIMA se basa en la construcción de un modelo de predicción estadístico ARIMA (Auto-Regressive Integrated Moving Average) [9] a partir de una serie temporal de datos de consumo registrados por un contador inteligente (los datos de entrenamiento). En general, un modelo ARIMA se caracteriza por una tripleta de parámetros (p, d, q), donde:

- p es el orden de autorregresión,
- q es el orden de media móvil, y
- d es el grado de diferenciación de la serie temporal.

Para crear un modelo ARIMA que se ajuste a la serie temporal es necesario asignar un valor a cada parámetro de la tripleta. Este paso es vital porque la calidad del modelo de predicción es altamente sensible a estos parámetros. Se ha utilizado la API *auto_arima*[2] que facilita el trabajo de búsqueda de los parámetros para finalmente obtener un modelo de predicción óptimo con respecto al criterio de información elegido. Hay diferentes criterios de información que se pueden utilizar como estimadores de la calidad de un modelo de regresión. En este caso, se ha elegido el *BIC (Bayesian Information Criterion)*, que es generalmente preferido para muestras grandes [9] como en el caso de los conjuntos de datos considerados (20160 observaciones para cada serie temporal).

Una vez se ha obtenido el modelo ARIMA a partir de los datos de entrenamiento, este es utilizado para calcular la predicción de consumo cada media hora en un intervalo de confianza del 95 %. El criterio de detección se basa en verificar si el consumo registrado por un contador en el momento t se encuentra

[2]Biblioteca Python `pmdarima`: `http://alkaline-ml.com/pmdarima`.

dentro del intervalo de confianza de la predicción en el momento t. En caso positivo, el consumo será considerado normal, en caso negativo, el consumo será considerado como anómalo.

ARIMAX. Este detector funciona exactamente igual que el detector ARIMA con la salvedad de que se utilizan *variables exógenas de Fourier* para obtener un modelo de predicción ARIMAX (Autoregressive Integrated Moving Average with Explanatory Variable). Las variables exógenas son calculadas considerando una periodicidad diaria de 48 registros (uno cada media hora).

2.3.3. Técnicas de *clústering*

Se han implementado cuatro detectores basados en técnicas de agrupamiento de datos. Uno de ellos es el detector *PCADBSCAN* [7]. El resto son *KMeans*, *MiniBatchKMeans* y *FisherJenks*, de los que no se ha encontrado información en la literatura relacionada con este dominio del problema.

PCADBSCAN. El detector PCADBSCAN combina dos técnicas de minería de datos:

- *PCA (Principal Component Analysis)* [10]: se aplica en el primer paso para reducir la dimensionalidad de los datos de consumo a un espacio bidimensional.
- *DBSCAN (Density-Based Spatial Clustering of Applications with Noise)* [11]: se aplica en el segundo paso al espacio generado por *PCA*.

A diferencia del resto de detectores, este utiliza todo el conjunto de datos para el primer paso. Es decir, no solamente utiliza el histórico de consumo registrado por un solo contador, sino que aprovecha la particularidad de que se trata de una red distribuida. De este modo trata de obtener información más relevante haciendo uso de todo el conjunto de contadores.

A partir del conjunto de datos de consumo utilizado para el entrenamiento, se construye una matriz a la que se le aplica la reducción de dimensionalidad con PCA. La matriz reducida obtenida con PCA permite así caracterizar cada semana de consumo de un contador con un punto en el espacio bidimensional.

En el segundo paso, se aplica DBSCAN a la matriz reducida considerando cada contador de manera independiente. El objetivo es obtener una agrupación de puntos que representen el escenario normal: todos aquellos puntos que estén fuera de esta agrupación serán considerados como anómalos.

Para definir la agrupación con DBSCAN se necesita asignar un valor a dos parámetros: el radio de la región circular alrededor de un punto (*eps*) y el número mínimo de puntos que están a una distancia menor que *eps* del punto. El primer parámetro se calcula con el estimador S_n (estimador de *Rousseeuw and Croux* [12]). El segundo parámetro se calcula mediante mayoría simple, es decir, que si hay 60 puntos, para que un punto sea considerado "normal", ha de tener al menos 31 puntos a una distancia Euclidiana *eps* de él. Esta condición

puede no darse para ninguno de los puntos del espacio, lo que implica que no se genera ninguna agrupación y, por tanto, no se puede aplicar el algoritmo.

KMeans. El detector *KMeans* se basa en el algoritmo de clasificación homónimo [13] que divide muestras en k grupos distintos. El objetivo de este algoritmo es resolver un problema de optimización, cuya función a optimizar es la suma de las distancias cuadráticas de cada una de las muestras al centroide (punto representativo de un *clúster*). Los centroides son inicializados aleatoriamente, y se recalculan siempre que se añade una nueva muestra al conjunto de muestras.

El detector implementado distingue dos grupos. El primero contiene las semanas con usos anómalos del contador. El segundo incluye las semanas de uso normal, por lo que aplica el algoritmo *KMeans* considerando dos *clústers* (k=2). En concreto, se entrena al detector con un resumen de los datos de consumo de un contador, que corresponde con la media de consumo de cada una de las semanas de entrenamiento. Él mismo, al ser un algoritmo de *clústering* de tipo no supervisado, genera los dos grupos mencionados anteriormente con datos de entrenamiento, con la finalidad de, posteriormente, asignar un grupo a cada una de las muestras de validación que le sean proporcionadas.

MiniBatchKMeans. El detector *MiniBatchKMeans* es una versión modificada de *KMeans*, que usa el concepto de *mini lotes (mini batches)* con la finalidad de reducir el tiempo total de cómputo del algoritmo *KMeans*. Los mini lotes están formados por subconjuntos pequeños de los datos de entrenamiento, escogidos aleatoriamente en cada una de las iteraciones empleadas para entrenar el modelo. Se ha probado [14] que *MiniBatchKMeans* converge antes que *KMeans*, pero esta característica va en detrimento de la eficacia del mismo.

FisherJenks. El detector *FisherJenks* se basa en el algoritmo homónimo [15]. No requiere entrenar ningún modelo, por lo que es algo distinto al resto. Es un algoritmo de *clústering* unidimensional cuya función principal es separar los valores numéricos de consumo de un contador que se le pasan en el número de contenedores que se le indiquen. Esta separación la realiza en base al valor numérico de dichas muestras en *kWh*. En este trabajo, se le indica que separe n muestras en tres contenedores: *normal*, *suspicious* y *anomalous*.

El algoritmo devuelve el número de muestras que contiene el contenedor *anomalous*. La desventaja de este detector es que, tal y como está implementado, trata las muestras individualmente. Otra aproximación que se deja como trabajo futuro sería tratar conjuntos de muestras (resumen de días enteros a partir de su media, por ejemplo).

2.3.4. Bosque de árboles de decisión

El detector *IsolationForest* implementa un algoritmo de clasificación no supervisado [16] que es capaz de encontrar anomalías en los datos basándose en el principio de que es fácil detectar datos anómalos, porque son pocos y muy

distintos al resto de datos (raros). Lanza un número de *isolation tree*, cada uno con el mismo conjunto de datos de entrenamiento. Para cada *isolation tree*, se aplica un algoritmo aleatorio que cuanto más anómala es una muestra, más rápido la descarta (menos ramas recorre). Esto implica que conforme se van bajando niveles en el árbol, menos anómalas son las muestras. Finalmente, se llega a consenso entre todos los *isolation tree* y para cada muestra se calcula el número medio de ramas que ha recorrido entre todos los *isolation tree*. Si una muestra ha recorrido pocas ramas en media, se marca como anómala. En caso contrario, si una muestra ha recorrido el árbol desde la raíz hasta las hojas en la mayoría de los árboles, se marca como normal.

La información resumida que se le pasa a este detector para entrenarlo y validarlo es la desviación estándar de los datos de consumo diarios. Con esto se sigue una aproximación algo diferente a las métricas utilizadas en otros detectores en los que se usa la media, por lo que en principio debería ser más eficaz ante los escenarios anómalos construidos en base al consumo promedio.

2.3.5. Comparación de distribuciones de frecuencia relativa

Se han implementado varios detectores que se basan en comparar distribuciones de frecuencia relativa del consumo registrado por un contador. El detector *KLD* fue propuesto por [7], mientras que su variante, *JSD*, y los diferentes detectores *TEG*, se proponen en este trabajo.

KLD y JSD. Ambos detectores etiquetan una semana de consumo como anómala si su distribución se desvía "demasiado" de la distribución de frecuencia relativa de consumo histórica, es decir, de la distribución obtenida de los datos de entrenamiento.

Los detectores se diferencian en el tipo de función utilizada para cuantificar la divergencia (o entropía) de las dos distribuciones [17]: *Kullback-Leibler Divergence (KLD)* y *Jensen-Shannon Divergence (JSD)*, respectivamente.

Un valor de divergencia cercano a cero para una semana de validación significa que los datos de consumo de la misma son similares a los datos de entrenamiento, mientras que si está muy alejado de cero significa que la semana es anómala.

El modelo de predicción se construye en base a la función de divergencia (es decir, KLD o JSD) y al número de contenedores (*bins*) que se van a utilizar para obtener, para cada semana de entrenamiento, un histograma con tantas barras como *bins*. Todos los contenedores son de la misma longitud, y se calculan teniendo en cuenta el consumo mínimo y máximo de todo el conjunto de datos de entrenamiento. Un histograma representa la distribución de frecuencia relativa de consumo semanal y es una aproximación de la distribución de probabilidad.

La función de divergencia se utiliza para calcular la divergencia de las distribuciones de consumo semanales y la distribución de consumo de las n semanas de entrenamiento. Los n valores de divergencia de las semanas de entrenamiento constituyen el modelo de predicción.

El criterio de clasificación se basa en establecer un umbral, percentil $(100-\alpha)$ de los n valores de divergencia, donde α es un número entre 0 y 100. Si la divergencia de la distribución de consumo de una semana A y la distribución de consumo de las n semanas de entrenamiento es mayor que este umbral, el detector marca el consumo de esta semana como anómalo. En caso contrario, el consumo es considerado normal.

TEG. Este detector se basa en construir una serie temporal de grafos (*Time Evolving Graph* [18]) a partir de los datos de consumo registrados por un contador en las semanas de entrenamiento. Un grafo representa el uso del contador durante una semana, donde los nodos modelan el nivel de consumo y las aristas corresponden al cambio de nivel de consumo. Los nodos y las aristas tienen pesos asociados (sus frecuencias de aparición). La generación de los grafos implica una discretización de los datos de consumo, es decir, se necesita pasar de valores en *kWh* a niveles (nivel 0, nivel 1, etc.). Para este propósito, de forma similar a los detectores *KLD* y *JSD*, se definen n *bins* intervalos equidistantes y contiguos incluidos entre el mínimo y el máximo valor de consumo durante el periodo de entrenamiento.

El modelo de predicción de este detector consiste en calcular, para cada semana de consumo, la *diferencia* entre el grafo asociado a la semana y el grafo de todo el periodo de entrenamiento (*grafo global*). Se han implementado 28 variantes de este detector. Cada variante utiliza una métrica diferente para calcular la diferencia entre grafos [17].

El criterio de detección es similar al criterio utilizado en los detectores *KLD* y *JSD*. En concreto, si la diferencia entre el grafo asociado a la semana de validación A y el grafo global es mayor del umbral percentil $(100 - \alpha)$, de los valores en el modelo de predicción, el detector marca el consumo de la semana A como anómalo.

2.3.6. *Deep Learning*

El detector *NN (Neural Network)* es el único que requiere ser entrenado con escenarios distintos del escenario normal, por lo que se deben identificar a priori todos los escenarios que se quieren distinguir. Es decir, es un tipo de aprendizaje supervisado. Esto significa que si se le presenta un escenario que no conoce (para el que no ha sido entrenado), lo confundirá con uno de los que sí conoce. El modelo de predicción está formado por dos capas de neuronas, utilizando 300 épocas[3]:

Primera capa. Capa de 50 neuronas con función de activación *relu*. Esta función transforma las entradas negativas a cero, dejando las positivas en su valor original.

[3]Una época corresponde a una pasada del conjunto de datos de entrenamiento por la red neuronal.

Segunda capa. Capa de N neuronas, siendo N el número de escenarios de consumo distintos que puede encontrarse la red neuronal (es decir, el escenario normal y los escenarios de ataque definidos en la subsección 2.2). Las N neuronas tienen una función de activación *softmax*, que es idónea para realizar clasificaciones múltiples, como es el caso.

Las variables del modelo de predicción, es decir: el número de capas y de neuronas por capa, las funciones de activación y el número de épocas, fueron seleccionadas después de horas de experimentación, teniendo en cuenta el producto vectorial de las listas de todas las variables mencionadas anteriormente.

El proceso de entrenamiento del modelo consiste en separar el conjunto de datos de consumo registrados por un contador durante 60 semanas en subconjuntos de 24 horas para cada uno de los N diferentes escenarios de consumo.

Para entrenar la red neuronal se han considerado $60 * 7 = 420$ muestras para cada escenario. Cada muestra agrupa las 24 horas antes mencionadas, y se resume utilizando los siguientes calificadores estadísticos (*features*):

- Media
- Media^2
- Media^3
- Desv. típica
- Varianza
- Moda
- Rango
- Coef. Variación (cv)
- cv^2
- *Skewness*
- Q1
- Q2
- Q3
- Q3 - Q1
- Último registro - Primer registro

Cuando se le presentan conjuntos de validación, la red neuronal devuelve un porcentaje de similitud de ese subconjunto de validación para cada escenario de consumo para el que fue entrenada. Estos subconjuntos son de 24 horas, al igual que los subconjuntos de entrenamiento. Con esta aproximación no sólo es capaz de identificar escenarios anómalos, sino que además es capaz de clasificar qué tipo de escenario es el que se está encontrando, por ejemplo: ataque *Swap*, ataque *FDI_30* o escenario normal.

Experimentalmente se ha comprobado que este detector proporciona un alto número de falsos positivos. Esto es así porque cuando se le presenta un escenario normal asigna prácticamente la misma probabilidad a los escenarios FDI_30 y RSA_0.5_1.5 que al normal. Por ello, se decidió introducir un "sesgo" en su implementación, aumentando un 4 % la probabilidad de que la red neuronal clasifique una muestra como un escenario normal en todos los casos. Esto no ha afectado sustancialmente al número de falsos negativos, pero sí que ha mejorado el ratio de verdaderos negativos de la red neuronal de manera considerable.

3. *Framework* desarrollado

Como se expone en la introducción, uno de los objetivos del presente trabajo es identificar ataques a las redes inteligentes. Comenzando desde los datos en "crudo" proporcionados por los contadores inteligentes, se ha desarrollado un proceso completo hasta obtener en un *dashboard* un análisis de la experimentación con los mismos, que proporciona un asesoramiento respecto a la calidad de los detectores y la eficacia de los ataques desde un punto de vista económico. La Figura 1 explica dicho proceso o *workflow*. Los óvalos en la figura representan actividades mientras que "los símbolos de página" representan los artefactos necesarios para llevar a cabo una actividad y/o producidos por otra. Para dar soporte a este *workflow*, en este proyecto se ha desarrollado un *framework* software que abarca todas estas etapas o actividades.

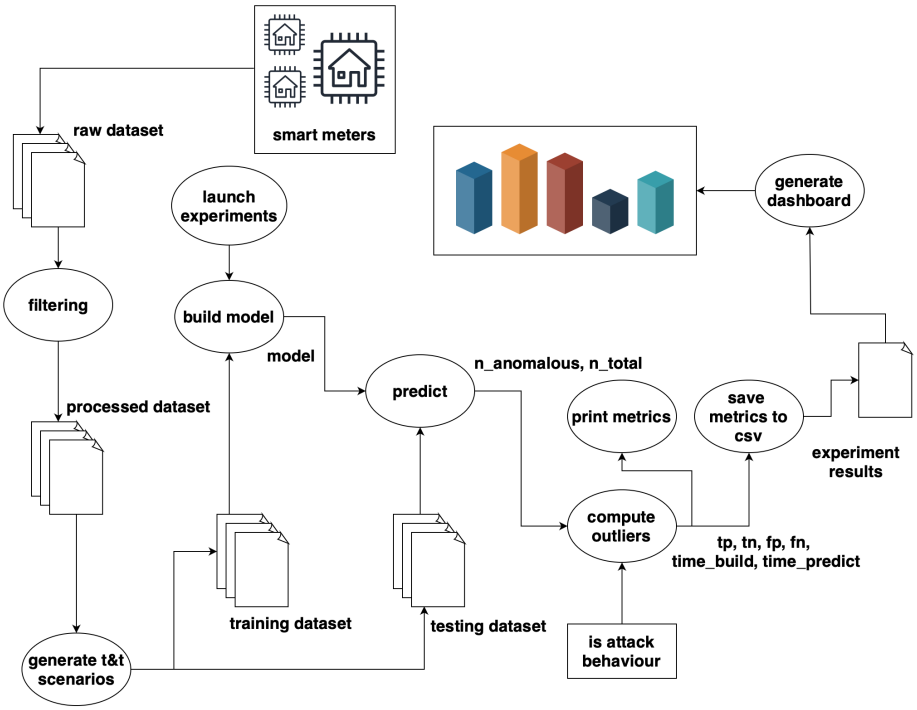


Figura 1: Flujo de actividades y artefactos.

Se comienza por los datos en crudo de los contadores (*smart meters*), que pasan por la actividad *filtering*. Esta actividad preprocesa y homogeneiza los datos que se utilizan en la actividad *generate t&t (training and testing) scenarios*, que es la encargada de generar ficheros de entrenamiento y de validación.

La actividad *launch experiments* desencadena una secuencia de actividades. Primero genera un modelo a partir de la operación *build model* utilizando fiche-

ros de entrenamiento. Después, este modelo es utilizado en la operación *predict* para clasificar el número de muestras anómalas dentro de unos ficheros de validación. A continuación, *compute outliers* crea una matriz de confusión con los datos proporcionados por *predict* (siempre que se le avise de si esa información corresponde a un ataque o no). Finalmente, la matriz de confusión, junto con los tiempos de construcción del modelo y de predicción con el mismo, se proporcionan en la salida estándar (*print metrics*). También se escriben en un fichero CSV (*save metrics to CSV*). Para más información con respecto a la actividad *launch experiments*, consultar el Algoritmo 1.

Los resultados de los experimentos se procesan *a posteriori* para generar un *dashboard* interactivo con diferentes comparativas gráficas y tabulares (*generate dashboard*).

El *framework* consta de cinco módulos diferentes, como muestra la Figura 2. Cada módulo, representado como paquete UML, es responsable de un subconjunto de las actividades mencionadas previamente. Además, cada módulo incluye uno o más ficheros, programados en *Python* y representados en la figura como rectángulos, que llevan a cabo estas actividades. Las siguientes subsecciones están dedicadas exclusivamente a describir en detalle cada uno de los módulos que componen el *framework*.

3.1. Módulo de preprocesado

El módulo de preprocesado (*preprocessing* en la Figura 2) es el responsable de las actividades de filtrado (*filtering*) y de generación de datos para el entrenamiento y la validación de los detectores (*generate t&t scenarios*). En concreto, contiene *programas* en *Python* para:

1. Preprocesar los datos de los *dataset* originales. Consiste en seleccionar los contadores que tienen los datos de consumo completos, es decir, 336 registros para todas las semanas. El preprocesado de datos de electricidad resulta muy costoso (alrededor de 12 horas de ejecución en media en el clúster de operaciones). El preprocesado del de gas se demora alrededor de unos 5 minutos.

2. Generar los datos de entrenamiento (*training dataset*) y de validación (*testing dataset*) para cada *dataset* preprocesado anteriormente. Los datos generados se corresponden con los escenarios mencionados en la subsección 2.2.

Para más información acerca del preprocesado de cada conjunto de datos, consultar el apéndice C.

3.2. Módulo de detectores

El módulo de detectores (*detectors* en la Figura 2) es responsable de varias actividades, cada una representada por un método de la clase abstracta *Detector*, que hereda de la clase *ABC* de *Python*. Esta clase abstracta sirve

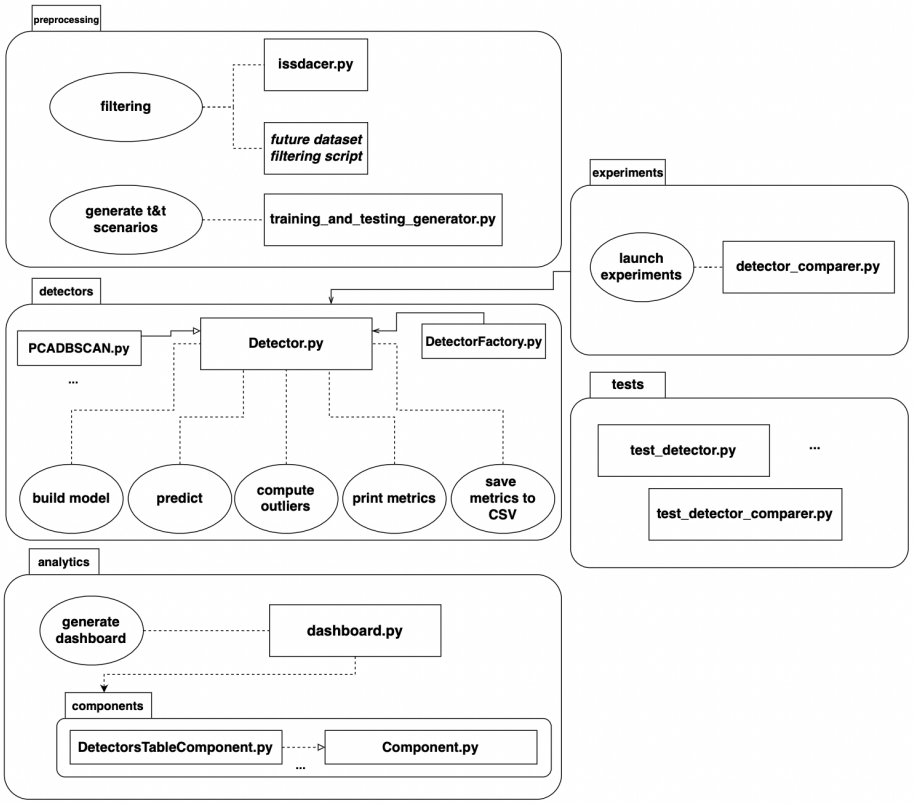


Figura 2: Vista de módulos del *framework* y su relación con las actividades.

como interfaz para facilitar el desarrollo y la inclusión de nuevos detectores, garantizando así la extensibilidad del *framework*. El módulo también contiene las clases hijas de *Detector*. Estas clases implementan los detectores presentados en la subsección 2.3, por ejemplo PCADBSCAN. Además, también proporciona una factoría de detectores.

Con el empleo de la herencia de clases como forma de transferencia de comportamiento se consiguen varios beneficios de diseño que resultaban necesarios para este módulo:

1. Los cambios en la clase abstracta *Detector* se propagan a las clases hijas que heredan de esta clase, favoreciendo así la mantenibilidad.

2. Relacionada con la ventaja anterior, no hay código repetido innecesariamente, lo que también facilita la mantenibilidad.

3. Extensibilidad. Hacer explícita una interfaz para implementar detectores proporciona dos ventajas. Primero, facilita el desarrollo de nuevos detectores, algo crucial en este ámbito. Segundo, facilita la integración con el

resto del *framework*, como puede ser con el módulo de lanzamiento de los experimentos, explotando así el principio de sustitución de Liskov [19].

La clase *Detector* cuenta con dos métodos abstractos (*build_model* y *predict*) que las clases hijas tienen que redefinir para poder ser instanciadas. Además, también posee otros cinco métodos que implementan operaciones de carga en memoria de los datos de entrenamiento y validación (*get_training_dataset* y *get_testing_dataset*, respectivamente), cálculo de la matriz de confusión en base al número de muestras totales y la predicción del detector (*compute_outliers*), escritura por pantalla (*print_metrics*) y en fichero de los resultados de un experimento (*metrics_to_csv*). En el apéndice D se incluye una descripción detallada de cada uno de los métodos.

Hay que observar que los métodos implementados en la clase *Detector* siempre se pueden redefinir, dentro de la semántica de las operaciones. Es decir, dentro de la interfaz, se proporciona un grado de libertad al desarrollador o desarrolladora que vaya a utilizar este módulo. En el caso del presente trabajo, el detector implementado en la clase *PCA-DBSCAN* redefine los métodos *get_training_dataset* y *get_testing_dataset*. Esto es así por sus características inherentes, ya que para el entrenamiento necesita todo el conjunto de datos, y no solo el conjunto de datos del contador en cuestión, mientras que para validarlo, necesita hacer una inserción de una columna al conjunto de datos de validación. Mencionar también que el detector implementado en la clase *MinAverage* redefine el método *compute_outliers* por su implementación del método *predict*, que devuelve un diccionario (así incluye no solo cuántas semanas se han marcado como anómalas, sino que también especifica exactamente qué semanas han sido, definiendo así una postcondición más fuerte). Estos son solo algunos ejemplos de las diferentes decisiones de diseño que ponen de manifiesto el esfuerzo realizado para obtener un *framework* que facilite a futuro el trabajo de los programadores y programadoras que lo vayan a extender o simplemente utilizar.

Como se ha mencionado antes, este módulo contiene una factoría de detectores, *DetectorFactory*. Esta clase ofrece un único método, estático, *create_detector*, que implementa el patrón "factory method". La funcionalidad de este método es muy sencilla. Se le pasa una cadena de texto que representa un identificador de un tipo de detector concreto (por ejemplo, "NN", "JSD" o "Min-Avg") e instancia y devuelve de manera idempotente un objeto del detector de la clase especificada. De esta manera queda desacoplada la forma de construir los detectores, encapsulando esta lógica única y exclusivamente en la factoría. Si se le indica un detector que no existe, lanza una excepción *Key Error*, indicando además cómo se debe proceder para añadir ese detector al conjunto de detectores que la factoría es capaz de construir (ver Figura 19).

3.3. Módulo de lanzamiento de los experimentos

El módulo de lanzamiento de los experimentos (*experiments* en la Figura 2) contiene el *programa detector_comparer* cuya funcionalidad es generar los valores necesarios para calcular las diferentes métricas básicas de calidad y de rendi-

miento correspondientes a los experimentos. Un experimento está definido por: un contador inteligente, un detector y un escenario de consumo relacionado con los registros del contador. Las métricas básicas generadas por un experimento son las mostradas en la Tabla 2: la matriz de confusión q, el *accuracy* y los tiempos de construcción y predicción del modelo asociado al detector.

Métrica básica	**Explicación**
$q \equiv \begin{bmatrix} tp & fp \\ fn & tn \end{bmatrix}$	$\begin{bmatrix} \text{true positives} & \text{false positives} \\ \text{false negatives} & \text{true negatives} \end{bmatrix}$
$accuracy$	$(tp + tn)/(tp + tn + fp + fn)$
$\langle time\_build, time\_predict \rangle$	$\langle$tiempo de construcción, tiempo de predicción$\rangle$

Tabla 2: Métricas básicas de calidad generadas.

El *programa* implementa el Algoritmo 1. Este algoritmo recorre la lista de contadores, de detectores y de escenarios. Para cada contador y detector, se crea un modelo de detección a partir de los datos de entrenamiento disponibles para ese contador. Cada escenario de consumo es un fichero de validación distinto, al que se le aplica la operación de predicción del detector. La predicción se basa en el modelo creado previamente. El resultado de la predicción se utiliza para crear una matriz de confusión que se añade al conjunto de métricas de calidad. Por otro lado, se añaden al conjunto de métricas de rendimiento los tiempos de construcción (del modelo) y de predicción.

Data: $\mathcal{E} = \langle \mathcal{C}, \mathcal{D}, \mathcal{S} \rangle$ (contadores, detectores, escenarios)
Result: $\mathcal{Q}$ (métricas de calidad), $\mathcal{P}$ (métricas de rendimiento)
$\mathcal{Q} = \emptyset$, $\mathcal{P} = \emptyset$;
foreach $c \in \mathcal{C}$ **do**
  foreach $d \in \mathcal{D}$ **do**
    $train = d$.get_training_dataset$(c, dataset)$;
    $\langle model, time\_build \rangle = d$.build_model$(train)$;
    foreach $s \in \mathcal{S}$ **do**
      $test = d$.get_testing_dataset$(s, c, dataset)$;
      $\langle pred, obs, time\_predict \rangle = d$.predict$(d, test, model)$;
      $q = d$.compute_outliers$(pred, obs, s)$;
      $\mathcal{Q} = \mathcal{Q} \cup \{q\}$;
      $\mathcal{P} = \mathcal{P} \cup \{\langle time\_build, time\_predict \rangle\}$;
    end
  end
end

Algoritmo 1: Lanzamiento de los experimentos y generación de métricas básicas.

El *programa* está parametrizado por *dataset*. También se pueden añadir/quitar detectores/escenarios de las listas que contiene en su programa principal, siempre y cuando se haya añadido el detector implementado a la factoría en el primer caso y se haya generado el *fichero que representa el escenario* en el segundo caso. Esto proporciona al desarrollador o desarrolladora la opción de extender el *framework* con total libertad, en lo referido a *datasets*, escenarios y/o detectores.

Destacar que el coste de ejecución de un plan de experimentos es muy alto cuando éste incluye un gran número de contadores, detectores y escenarios. Un plan de experimentos como el $\mathcal{E}_7$ (ver Tabla 4), con 1.000 contadores, 12 escenarios y 28 detectores contiene $1.000 \times 12 \times 28 = 336.000$ experimentos. La ejecución de estos experimentos se demoró durante más de 14 días en el *clúster* de supercomputación "Hermes" del I3A. Es por ello que el programa contiene realimentación, ya que así indica el número de contadores que quedan por procesar, y el tiempo estimado (ver Figura 3). Este último se calcula multiplicando el tiempo medio que costó procesar los anteriores contadores por los restantes. De la Figura 3 podemos calcular fácilmente que el tiempo medio de procesamiento de cada *meterID* (contador) es aproximadamente $23.857/35 = 682$ segundos (en ese *hardware* y con esas condiciones específicas de experimentación, que no son las de $\mathcal{E}_7$). Finalmente también es importante destacar que hay detectores que emplean un tiempo varias órdenes de magnitud superior que otros en crear el modelo o en predecir los resultados, por lo que el tiempo de ejecución es altamente dependiente de los detectores considerados en el plan de experimentos en cuestión.

```
MeterID:                        1366
Detector:                       NN
Attack:                         FDI30
Exec. time of model creation:   7.379162788391113 seconds
Exec. time of model prediction: 1.6260120868682861 seconds
Accuracy:                       1.0
Number of true positives:       53
Number of false negatives:      0
Number of true negatives:       0
Number of false positives:      0
[ 53 0 ]
[ 0 0 ]


35 meterIDs remaining. It will be completed in 23857.365752489335 seconds (aprox.)
```

Figura 3: Ejecución del *programa detector comparer* y tiempo estimado.

3.4. Módulo de análisis de los experimentos

El módulo de análisis de los experimentos (*analytics* en la Figura 2) contiene un *programa* cuya funcionalidad es visualizar en forma de gráficas y tablas diferentes métricas de calidad y de rendimiento (ver Tabla 3). Con estos elementos, se facilita el análisis comparativo de los detectores. Las métricas para compararlos se construyen a partir de las métricas básicas (ver Tabla 2). La otra

funcionalidad del *módulo* es mostrar el impacto de cada escenario de ataque en términos del consumo y del beneficio económico que proporcionan al atacante.

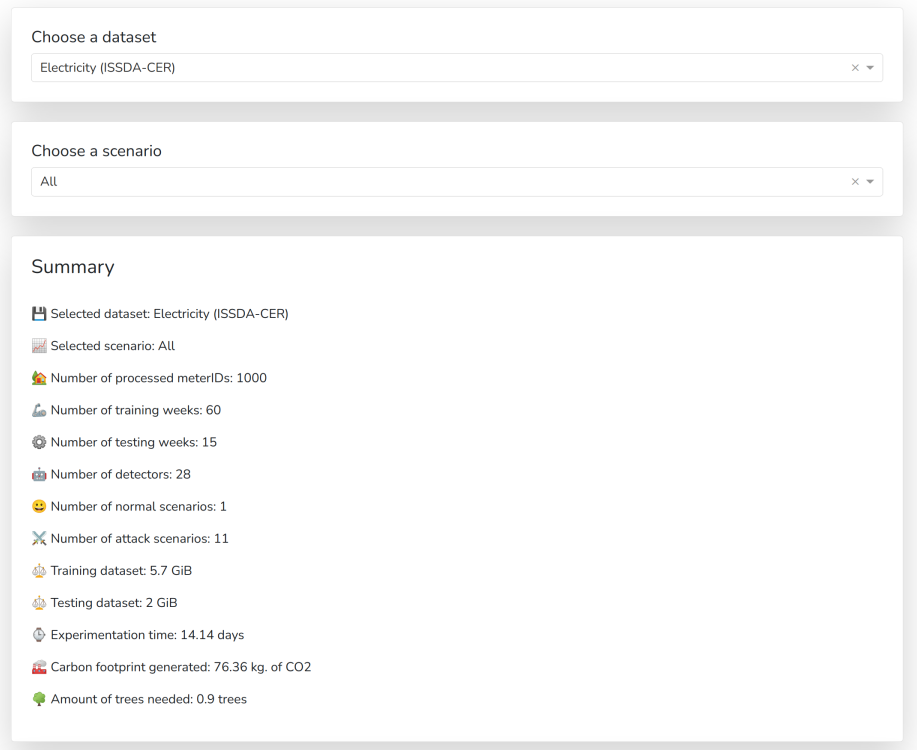
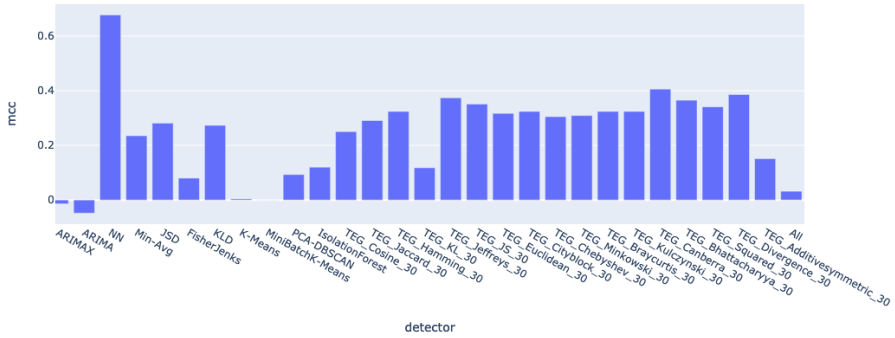


Figura 4: Configuración del *dashboard* y resumen del plan de experimentos.

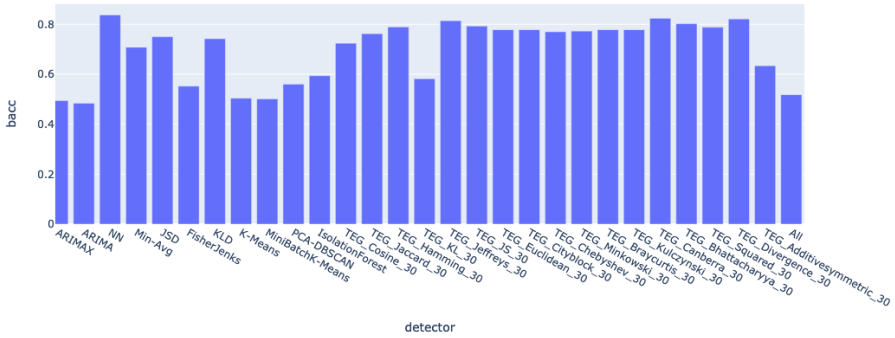
El *módulo* genera una interfaz web en *http://localhost:8050* con la que se puede interactuar. La interfaz muestra la configuración actual del *dashboard*, que puede ser cambiada modificando el *dataset* o el escenario, ver Figura 4).

El *programa* toma como entrada los resultados generados por el módulo *experiments* (subsección 3.3) y calcula las métricas de la Tabla 3 considerando dos niveles de abstracción: *global*, que permite comparar la calidad y el rendimiento de los detectores teniendo en cuenta todos los escenarios, y *unitario*, que permite comparar la calidad y el rendimiento de los detectores para un escenario de consumo concreto. Esta comparación se hace siempre teniendo en cuenta resultados de un plan de experimentos para un *dataset* concreto, es decir, no se mezclan resultados para distintos conjuntos de datos. La Figura 5 muestra las gráficas de dos métricas que se pueden visualizar con la configuración del *dashboard* de la Figura 4.

Los escenarios de ataque se comparan con respecto al consumo original, véase, por ejemplo, la Figura 6. El beneficio obtenido por un atacante se calcula en base a la factura que pagaría llevándolos a cabo durante 60 semanas, en



(a) Matthews Correlation Coefficient



(b) Balanced accuracy

Figura 5: Ejemplos de gráficas mostradas en el *dashboard* para comparar los detectores.

media. Para este cálculo se tienen en cuenta también factores como el valor cambiante del precio a lo largo del día. En el caso del presente trabajo, se ha utilizado una tarifa de *0,04€/kWh* desde las 00:00 hasta las 09:00 y de *0,11€/kWh* desde las 09:00 hasta las 24:00. El *dashboard* incluye una tabla como la mostrada en la Figura 7, que recoge los beneficios obtenidos para cada tipo de ataque considerado en el plan de experimentos.

En el desarrollo del *dashboard* se ha aplicado el patrón arquitectural modelo-vista-controlador [20], separando las gráficas/tablas en componentes, que heredan de una clase abstracta *Component*, y son instanciados a través de una factoría. Para más información acerca de la arquitectura del *dashboard*, consultar el apéndice D.

acc	accuracy	$(tp + tn)/(tp + fn + fp + tn)$
mcc	Matthews correlation coefficient	$\frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}}$
bacc	balanced accuracy	$(tpr + tnr)/2$
tpr	true positive rate	$tp/(tp + fn)$
tnr	true negative rate	$tn/(fp + tn)$
time_build	time to build the model	
time_predict	time to predict with the model	

Tabla 3: Métricas utilizadas para la comparación de detectores.

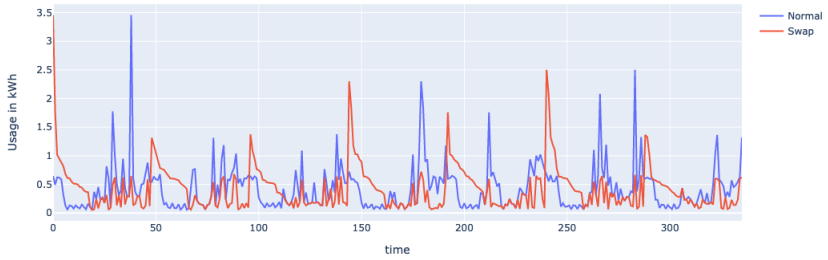


Figura 6: Ataque *Swap* comparado con el escenario normal.

Economic benefit for each scenario 💰

How much money does an average attacker save in 60 weeks?

Scenario	Bill (€)	Benefit (€)
Normal	904.75	0
FDI0	0	904.75
FDI5	45.24	859.51
FDI10	90.48	814.27
FDI20	180.95	723.8
FDI30	271.43	633.32
Swap	598.05	306.7
RSA_0.5_1.5	904.15	0.6
RSA_0.5_3	1578.82	-674.07
RSA_0.25_1.1	612.15	292.6
Avg	790.96	113.79
Min-Avg	322.16	582.59

Figura 7: Comparación económica de escenarios mostrada en el *dashboard*.

3.5. Módulo de infraestructura de *testing*

El módulo de infraestructura de *testing* (*tests* en la Figura 2) contiene varios *programas de testing* desarrollados haciendo uso del *framework Pytest*. Es este módulo del *framework* el que verifica dinámicamente las piezas *software* de los demás módulos: *detectors*, *experiments* y *analytics*. No se han desarrollado *tests* sobre el módulo de *preprocessing* por dos motivos:

- No es un módulo que vaya a cambiar. Cada *programa* del módulo de preprocesado de datos realiza su cometido y no se vuelve a modificar, puesto que tampoco se modifica el formato de los datos obtenidos de los contadores.
- En general, es computacionalmente muy costoso verificar y validar que el preprocesamiento de un conjunto de datos es correcto.

La infraestructura desarrollada no se limita a la ejecución local de los *tests*, sino que éstos se lanzan cada vez que se realiza una actualización (*push*) del repositorio de *GitHub*, utilizando la herramienta *GitHub Actions*. En concreto, los *tests* se lanzan en seis máquinas virtuales, cada una con una versión diferente de *Ubuntu* y de *Python* (ver Tabla 11).

Si en cualquiera de las máquinas virtuales falla uno o más *tests*, se notifica al usuario que ha realizado la actualización del repositorio. La ejecución de los mismos, tanto en local como en "la nube", se realiza utilizando la misma instrucción con unos *flags* para mostrar la cobertura de instrucciones de código de cada módulo, indicando cuáles son las líneas de código que no se han ejecutado a través de los *tests*.

```
pytest --cov-report term-missing --cov=src
```

Este módulo es realmente importante, ya que:

- Facilita refactorizar sin miedo a romper las piezas que se habían construido antes, proporcionando seguridad a cualquier futuro programador o programadora que utilice el *framework*.
- Favorece la mantenibilidad del *framework*.
- Proporciona ejemplos de cómo se podrían implementar *tests* a detectores que se pudieran incluir en un futuro.

Actualmente el *framework* cuenta con 1.411 *tests* (muchos de ellos parametrizados), alcanzando una cobertura de líneas de código de un 74 %.

4. Experimentación

Los experimentos han sido realizados con el *framework* descrito en la sección 3, ejecutándose en el *clúster de supercomputación "Hermes" del I3A*, con 8 núcleos, un Intel Xeon Gold 6254 CPU, 32GB RAM y 1 TB SSD, siendo Ubuntu 20.04 el sistema operativo.

Se planteó una metodología de experimentación iterativa. En concreto, se llevaron a cabo 8 planes de experimentación, $\mathcal{E}_1$ a $\mathcal{E}_8$, tal y como se ve en la Tabla 4. Cada plan de experimentación refina al anterior, y tiene en cuenta un conjunto de datos, de detectores y de escenarios. Las columnas $|\mathcal{C}_i|$, $|\mathcal{D}_i|$ y $|\mathcal{S}_i|$ de la Tabla 4 indican el número de contadores, detectores y escenarios de cada plan $\mathcal{E}_i$, respectivamente. Un plan de experimentación se ejecuta con el *programa detector comparer* del módulo de lanzamiento de experimentos, explicado en la subsección 3.3.

Las iteraciones se prolongaron a lo largo de los meses que ha durado este proyecto. En concreto, durante la experimentación se utilizaron 60 semanas de entrenamiento y 15 de validación para el *dataset* ISSDA CER de electricidad (ver planes $\mathcal{E}_1$, $\mathcal{E}_3$, $\mathcal{E}_5$ y $\mathcal{E}_7$), mientras que para el *dataset* ISSDA CER de gas se emplearon 59 semanas de entrenamiento y 15 de validación (ver planes $\mathcal{E}_2$, $\mathcal{E}_4$, $\mathcal{E}_6$ y $\mathcal{E}_8$).

Plan	Fecha	Duración (días)	*Dataset*	$\lvert\mathcal{C}_i\rvert$	$\lvert\mathcal{D}_i\rvert$	$\lvert\mathcal{S}_i\rvert$
$\mathcal{E}_1 = \langle \mathcal{C}_1, \mathcal{D}_1, \mathcal{S}_1 \rangle$	17/10/20	9	electricidad	500	10	4
$\mathcal{E}_2 = \langle \mathcal{C}_2, \mathcal{D}_2, \mathcal{S}_2 \rangle$	16/11/20	6	gas	500	9	4
$\mathcal{E}_3 = \langle \mathcal{C}_3, \mathcal{D}_3, \mathcal{S}_3 \rangle$	09/02/21	7	electricidad	500	11	10
$\mathcal{E}_4 = \langle \mathcal{C}_4, \mathcal{D}_4, \mathcal{S}_4 \rangle$	16/02/21	4	gas	500	10	10
$\mathcal{E}_5 = \langle \mathcal{C}_5, \mathcal{D}_5, \mathcal{S}_5 \rangle$	18/03/21	7	electricidad	500	14	12
$\mathcal{E}_6 = \langle \mathcal{C}_6, \mathcal{D}_6, \mathcal{S}_6 \rangle$	25/03/21	5	gas	500	11	12
$\mathcal{E}_7 = \langle \mathcal{C}_7, \mathcal{D}_7, \mathcal{S}_7 \rangle$	14/05/21	14,14	electricidad	1.000	28	12
$\mathcal{E}_8 = \langle \mathcal{C}_8, \mathcal{D}_8, \mathcal{S}_8 \rangle$	31/05/21	1,39	gas	1.000	25	12

Tabla 4: Planes de experimentación.

El proceso iterativo ha permitido, a través de las observaciones de los resultados, tomar decisiones significativas desde el inicio. Esto ha supuesto una mejora continua del propio proceso. En concreto, las decisiones han sido con respecto al *framework* y al propio enfoque de la experimentación. A continuación se resumen los aspectos más importantes.

Inicialmente, durante $\mathcal{E}_1$, el *framework* permitía preprocesar conjuntos de datos, crear detectores y lanzar experimentos. Sin embargo, no era posible analizar los resultados de los experimentos, ya que no había ningún módulo especializado

en procesarlos para obtener y mostrar *información útil* a partir de ellos. Es por eso que se necesitó introducir un nivel de abstracción más alto, diseñando e implementando un *dashboard* para interpretar los resultados y poder así comparar la calidad de los detectores.

Durante los cuatro primeros planes, $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ y $\mathcal{E}_4$, se detectó que algunos escenarios de ataque que no reportaban grandes beneficios solían y suelen ser confundidos con el escenario normal. Sin embargo, se decidió restarle importancia a estos casos, priorizando escenarios en los que el impacto económico fuese mayor.

Durante la realización de los planes $\mathcal{E}_5$ y $\mathcal{E}_6$, cuyos resultados han sido publicados en el trabajo [1], surgió la necesidad de mejorar el *dashboard* desarrollado hasta entonces. Esto implicó evaluar las métricas que se estaban utilizando (*accuracy, recall, precision*) para analizar si era necesario incluir alguna nueva. Se llegó a la conclusión de que la métrica *accuracy* podía no ser del todo fiable, ya que puede dar lugar a interpretaciones erróneas. Por ejemplo, no es una buena métrica cuando hay 11 escenarios de ataque y 1 escenario normal, ya que los de ataque corresponden a $\approx 91\,\%$ de los casos, por lo que clasificar todas las muestras como anómalas proporcionaría un *accuracy* $\approx 91\,\%$. Se investigaron alternativas y se añadieron las métricas *Matthews correlation coefficient (mcc)* y *Balanced Accuracy (bacc)* al *dashboard*, que contemplan este tipo de casos no balanceados.

Para los planes $\mathcal{E}_7$ y $\mathcal{E}_8$ se implementó la versión *web* del *dashboard*, que recoge todas las métricas de la Tabla 3.

Después de lanzar el plan $\mathcal{E}_7$ se decidió que se deberían eliminar del siguiente plan de experimentos aquellos detectores computacionalmente más costosos y de baja calidad, por ejemplo ARIMA(X). Las razones fueron los pobres resultados de algunos detectores, además de que se empezó a pensar en el coste energético de los propios experimentos. Esto impulsó a que finalmente se añadiera en el *dashboard* información correspondiente a la *huella de carbono* generada por la experimentación, la cual disminuyó $\approx 90\,\%$ en el plan $\mathcal{E}_8$ con respecto al plan $\mathcal{E}_7$.

4.1. Conclusiones de la experimentación

Los resultados de la experimentación se proporcionan en el apéndice E. A partir de dichos resultados se obtienen las siguientes conclusiones de interés.

Con respecto a los detectores. El detector basado en redes neuronales parece el más prometedor con respecto a los datos de electricidad. Se obtiene un 0,677 en la métrica *mcc* y un 0,837 de *bacc*, tal y como se puede apreciar en la Figura 25. Sin embargo, funciona algo peor detectando el consumo normal ya que tiene una ratio de 0,702 de verdaderos negativos. Además, al pertenecer a la categoría de aprendizaje supervisado, requiere ser entrenado con todos los escenarios de consumo que se desee que sea capaz de detectar (algo complicado en este área). Esto hace que el detector TEG_Canberra, con 30 *bins*, sea también una opción interesante, con mayor ratio de verdaderos negativos y con unos

tiempos de construcción y de predicción varias órdenes de magnitud inferiores. Por contra, hay varios detectores, como el K-Means o el MiniBatchK-Means, que tienen muy baja calidad. Estos obtienen valor ≈ 0 en la métrica *mcc* y $\approx 0{,}5$ en la métrica *bacc*. Esto significa que son igual de útiles que lanzar una moneda al aire. Finalmente, los detectores ARIMA(X) son muy poco eficientes en términos de tiempos de ejecución. En concreto, ARIMAX es unas 15 veces más lento generando el modelo que el basado en redes neuronales, que, realmente no destaca por su velocidad, pues suele tardar 1 minuto en generarlo.

Con respecto a los escenarios de ataque. Algunos son fáciles de detectar a simple vista analizando sus curvas de consumo, como el ataque *swap* (ver Figura 6). Sin embargo, otros como los *RSA* son más complicados de detectar de esta manera, ya que sus curvas de consumo son muy similares a las de un consumo normal (ver Figura 8), además de que sería muy costoso inspeccionar manualmente las curvas de consumo de cada contador, una a una. Evidentemente, el ataque más rentable es el FDI_0. Otro muy rentable es el Min-Avg, permitiendo ahorrar en media hasta un 35 % de la factura, para el conjunto de datos eléctrico (ver Figura 7).

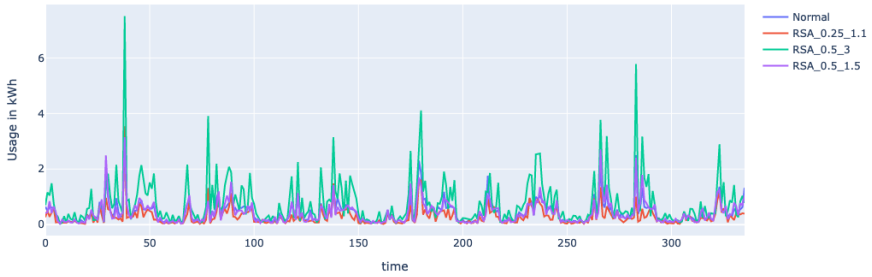


Figura 8: Ataques *RSA* comparados con el escenario normal.

5. Conclusiones

Este proyecto ha sido llevado a cabo con éxito, puesto que se han cumplido todos los objetivos fijados en la propuesta inicial del Trabajo Fin de Grado. A continuación se exponen las conclusiones alcanzadas con la realización de este proyecto, que van a ser divididas en: lecciones aprendidas, trabajo futuro y recomendaciones.

5.1. Lecciones aprendidas

Con la realización del trabajo se ha aprendido a desarrollar un proyecto de investigación en un ámbito real y de vanguardia.

La necesidad de implementar un *framework* para facilitar el trabajo futuro en este sector ha supuesto un reto, puesto que se quería desarrollar un *software* reusable, mantenible y extensible. También se ha trabajado con multitud de herramientas que no se conocían, lo que ha requerido estudiarlas previamente. La dificultad del trabajo ha residido principalmente en que no se tenían conocimientos previos en el dominio del problema, ni tampoco de muchos de los conceptos matemáticos necesarios para desarrollar los detectores y crear escenarios de ataque verosímiles. Esto ha llevado a investigar a fondo, leyendo multitud de contribuciones que definen el estado del arte en la materia (ver apéndice A).

5.2. Trabajo futuro

A continuación se citan algunas propuestas de trabajo futuro en este área:

- Utilizar otros conjuntos de datos abiertos, como el de datos de energía solar *(Ausgrid dataset)*, que incluye también datos de generación de energía.
- Analizar el impacto que podría tener el ataque *RSA_0.5_3* en un conjunto de datos de generación y venta de energía como el de *Ausgrid*.
- Analizar el rendimiento del detector *Fisher Jenks* tratando conjuntos de muestras (resúmenes de los conjuntos). Actualmente en el *framework* trata muestras individuales.
- Analizar el rendimiento del detector *Isolation Forest* utilizando otras métricas. Actualmente está utilizando únicamente la desviación estándar del consumo del conjunto de datos de entrada.
- Analizar el rendimiento de las variantes del detector *TEG* utilizando diferentes periodicidades. Actualmente está usando muestras semanales.
- Investigar más en detalle los resultados actuales para tratar de combinar detectores a través de funciones *booleanas*. Ejemplo: solo marcar como anómalos aquellos registros que los detectores X e Y han marcado como anómalos (función *AND*), aunque se podrían realizar combinaciones mucho más complicadas, especialmente si se utilizan detectores de bajo coste computacional.

- Analizar detenidamente si la clasificación de los contadores (diferentes tipos de clientes o de contratos) afecta al rendimiento de los detectores y/o de los ataques.

5.3. Recomendaciones

A continuación se proponen recomendaciones para trabajar en este área:

- Utilizar el *framework* desarrollado en el presente trabajo. Adaptarse a él lo máximo posible, pues él se adapta realmente bien a este dominio del problema.
- Leer la documentación del *framework*. Está disponible en diferentes ficheros *README.md*, además de la propia documentación de cada uno de los ficheros de código.
- Contar con un servidor virtual privado *VPS* potente para realizar las tareas de experimentación, pues si se quiere realizar un plan de experimentos completo, se va a requerir buena capacidad de cómputo.
- Desechar de los planes de experimentos aquellos detectores que no aporten buenos resultados, especialmente si tardan mucho en crear el modelo y/o predecir con el modelo. El cambio climático es un problema real, y todos y todas podemos aportar nuestro granito de arena para intentar frenarlo.
- Si esto se utilizara masivamente por grandes instituciones o empresas, comprometerse a plantar el número de árboles indicado en el resumen del *dashboard*. Esto podría ayudar a contrarrestar la huella de carbono generada por las costosas ejecuciones.

Referencias

[1] Simona Bernardi, Raúl Javierre, José Merseguer, and José Ignacio Requeno. Detectors of Smart Grid Integrity Attacks: An Experimental Assessment. In *17th European Dependable Computing Conference, September 13-16, 2021*, 2021. Aceptado para su publicación en las actas del congreso. Versión *preprint*: `https://github.com/DiasporeUnizar/smartest`.

[2] Smart Energy International. Energy Theft and Fraud Reduction. `https://www.smart-energy.com/industry-sectors/energy-grid-management/energy-theft-and-fraud-reduction/`, June 2020. (Último acceso: 18/06/2021).

[3] Python. *Python*. URL: `https://www.python.org/` (Último acceso: 18/06/2021).

[4] Agile Manifesto. *Manifesto for Agile Software Development*. URL: `https://agilemanifesto.org/` (Último acceso: 18/06/2021).

[5] Irish Social Science Data Archive, Commission for Energy Regulation. *CER Smart Metering Project*. URL: `https://www.ucd.ie/issda/data/commissionforenergyregulationcer/` (Último acceso: 18/06/2021).

[6] Pablo Aznar Delgado. Development of a Deep Learning based attack detection system for Smart Grids. Master's thesis, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid, 2019. URL: `http://oa.upm.es/55617/` (Último acceso: 18/06/2021).

[7] Varun Badrinath Krishna. *Data-driven methods to improve resource utilization, fraud detection, and cyber-resilience in Smart Grids*. PhD thesis, Graduate College of the University of Illinois at Urbana-Champaign, 2018.

[8] Daisuke Mashima and Alvaro A. Cárdenas. Evaluating Electricity Theft Detectors in Smart Grid Networks. In Davide Balzarotti, Salvatore J. Stolfo, and Marco Cova, editors, *Research in Attacks, Intrusions, and Defenses*, pages 210–229, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications – With R Examples*. Springer Texts in Statistics. Springer, 2016. Fourth edition.

[10] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Phil.Trans.R.Soc.A*, 374, April 2016.

[11] Kelvin Salton do Prado. How DBSCAN works and why should we use it?, April 2017. Towards Data Science, URL: `https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80` (Último acceso: 18/06/2021).

[12] P.J. Rousseeuw and C. Croux. Alternatives to the Median Absolute Deviation. *Journal of the American Statistical Association*, 88:1273—1283, 1993.

[13] Imad Dabbura. K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks, September 2018. Towards Data Science, URL: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a (Último acceso: 18/06/2021).

[14] scikit learn.org. *Mini Batch K-Means*. URL: https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans (Último acceso: 16/03/2021).

[15] Chris Moffitt. Finding Natural Breaks in Data with the Fisher-Jenks Algorithm, December 2019. Practical Business Python, URL: https://pbpython.com/natural-breaks.html (Último acceso: 18/06/2021).

[16] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[17] Sung-Hyuk Cha. Comprehensive survey on distance/similarity mea-sures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.

[18] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph Based Anomaly Detection and Description: A Survey. *Data Min. Knowl. Discov.*, 29(3):626–688, May 2015.

[19] eduesqui. Cómo cumplir con el Principio de Sustitución de Liskov, September 2018. Medium, URL: https://medium.com/eduesqui/como-cumplir-con-el-principio-de-sustituci%C3%B3n-de-liskov-1eddcdb0cb3 (Último acceso: 18/06/2021).

[20] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.

[21] *NumPy: The fundamental package for scientific computing with Python*. URL: https://numpy.org/ (Último acceso: 18/06/2021).

[22] *Pandas: a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language*. URL: https://pandas.pydata.org/ (Último acceso: 18/06/2021).

[23] *Matplotlib: Python plotting*. URL: https://matplotlib.org/ (Último acceso: 18/06/2021).

[24] *Scikit-learn: Machine Learning in Python*. URL: https://scikit-learn.org/ (Último acceso: 18/06/2021).

[25] Google. *TensorFlow: Una plataforma de extremo a extremo de código abierto para el aprendizaje automático*. URL: https://www.tensorflow.org/?hl=es-419 (Último acceso: 18/06/2021).

[26] *Keras: the Python deep learning API*. URL: https://keras.io/ (Último acceso: 18/06/2021).

[27] Pypi. *pmdarima: a statistical library designed to fill the void in Python's time series analysis capabilities*. URL: https://pypi.org/project/pmdarima/ (Último acceso: 18/06/2021).

[28] *SciPy: a Python-based ecosystem of open-source software for mathematics, science, and engineering*. URL: https://www.scipy.org/ (Último acceso: 18/06/2021).

[29] National Aeronautics and Space Administration. *NASA Risk Management Handbook*, November 2011. NASA/SP-2011-3422, Version 1.0.

Lista de Figuras

1. Flujo de actividades y artefactos. 14
2. Vista de módulos del *framework* y su relación con las actividades. 16
3. Ejecución del *programa detector comparer* y tiempo estimado. . . 19
4. Configuración del *dashboard* y resumen del plan de experimentos. 20
5. Ejemplos de gráficas mostradas en el *dashboard* para comparar los detectores. 21
6. Ataque *Swap* comparado con el escenario normal. 22
7. Comparación económica de escenarios mostrada en el *dashboard*. 22
8. Ataques *RSA* comparados con el escenario normal. 26
9. Horas pendientes, estimadas e invertidas. 38
10. Distribución del tiempo invertido por categorías. 39
11. Concentración de trabajo. 39
12. Evolución de las horas dedicadas. 39
13. Hoja de cálculo de seguimiento de esfuerzos. 40
14. Tablero *kanban* utilizado. 41
15. *Issues* de *GitHub*. 41
16. Proporción de *meterIDs* en el *dataset* filtrado de electricidad. . . 46
17. Proporción de *meterIDs* en el *dataset* filtrado de gas. 47
18. Clase abstracta Detector. 48
19. README del módulo de detectores. 50
20. Primer boceto del *dashboard*. 52
21. Estructura de clases del *dashboard*. 53
22. Actualización correcta en las 6 máquinas virtuales. 55
23. Configuración de la integración continua. 56
24. Resultados de los *tests* en la integración continua. 57
25. Resultados del plan de experimentos $\mathcal{E}_7$. 60
26. Resultados del plan de experimentos $\mathcal{E}_8$. 61

Lista de Tablas

1. Detectores considerados en el proyecto. 7
2. Métricas básicas de calidad generadas. 18
3. Métricas utilizadas para la comparación de detectores. 22
4. Planes de experimentación. 24
5. Descripción de las categorías. 38
6. Riesgos identificados. 42
7. Descripción de las clasificaciones de probabilidades e impacto. . . 43
8. Matriz NASA. 43
9. Riesgos de mayor exposición y estrategias de mitigación. 43
10. Distribución de *meterIDs* utilizados en la experimentación. . . . 47
11. Máquinas virtuales utilizadas para ejecutar los *tests*. 55

A. Estado del arte y herramientas

En este apéndice se realiza un estudio del estado del arte de las redes inteligentes, *smart grid* (subsección A.1): qué son, de qué elementos se componen, cuáles son sus ventajas y sus desventajas, a qué ataques están expuestas y qué enfoques se proponen actualmente para la detección de fraudes. También se detallan las herramientas de apoyo a la minería de datos/aprendizaje automático utilizadas en el *framework* desarrollado (subsección A.2).

A.1. Las redes eléctricas inteligentes

Las redes eléctricas inteligentes, redes de distribución eléctricas inteligentes o *smart grid* son una evolución de la red eléctrica tradicional. Esta evolución es posible en gran medida gracias a los dispositivos que constituyen las mismas: los contadores inteligentes. Los contadores inteligentes son unos aparatos cuya función es transmitir a la red en todo momento el consumo o la producción de energía. Son realmente parecidos a los contadores tradicionales, con la diferencia de que estos últimos no están conectados conformando una gran red de comunicaciones bidireccional. Esta interconexión permite manejar grandes cantidades de información en tiempo real. Además, se prevé que las *smart grid* pasen a ser la norma en unos años.

A continuación se resumen las principales ventajas de las redes eléctricas inteligentes:

- Mayor sostenibilidad, es decir, incentivan el uso de generadores de energías renovables.
- Mayor capacidad para que el usuario monitorice sus propios consumos.
- Mayor capacidad para que el usuario desconecte su contador de la red (por ejemplo, en horas pico), pasando a consumir energía de sus propios generadores ecológicos (paneles solares, etc.).
- Mayor participación de los usuarios en el mercado energético, puesto que pueden vender energía a la propia red.
- Mayor capacidad para investigar usos fraudulentos por parte de algunos usuarios.
- Mayor capacidad para analizar el mercado energético; por ejemplo, cuáles son las horas con demanda elevada, etc.

Sin embargo, estas redes están expuestas a ataques, ya que son un enorme sistema distribuido conformado por una gran cantidad de elementos electrónicos que pueden contener vulnerabilidades (*hardware* o *software*). Se puede afirmar que las desventajas de las redes eléctricas inteligentes van ligadas a los ataques a los que están expuestas. Algunos de los ataques pueden ser los identificados por Pablo Aznar en [6]:

- Posibilidad de espiar/controlar los horarios de una persona basándose en la información recogida por los contadores inteligentes.
- El crecimiento sin control del número de contadores inteligentes puede acabar significando que se fabriquen de muchos tipos, cada uno con diferentes vulnerabilidades que pueden ser explotadas con ciberataques.
- El acceso físico a los dispositivos que conforman la red puede implicar un control absoluto sobre los mismos.
- Los contadores acabarán siendo sistemas legados si no se realizan labores de mantenimiento y de actualización periódica sobre los mismos. Son especialmente importantes las actualizaciones de seguridad.
- Todos los dispositivos se comunican a través de la red, por lo que hay que prestar atención a ataques inherentes al mundo de las comunicaciones: interrupción, modificación, suplantación... Además de que los protocolos de red (ARP, IP, etc.) no fueron concebidos teniendo en cuenta la seguridad. Esta ha venido tiempo después, a base de parchear protocolos y *tunelizar*, entre muchas otras medidas.

Este trabajo se basa parcialmente en las contribuciones de Varun Badrinath Krishna [7] y Pablo Aznar Delgado [6]. Éstos han realizado estudios sobre ataques de integridad a la infraestructura de medición avanzada (*Advanced Metering Infrastructure*, en inglés), proponiendo diferentes tipos de detectores de consumos anómalos y diferentes escenarios de ataque.

Badrinath en su tesis de doctorado demuestra que usando técnicas de *machine learning* y métodos estadísticos para construir modelos empíricos de generación y consumo se mejora tanto el uso de los recursos, como la detección de fraude como la ciber resiliencia en las *smart grid*. En el presente trabajo se ha querido comprobar el rendimiento de algunos de los detectores que propuso él. Para ello ha sido necesario implementarlos desde cero siguiendo sus indicaciones.

Por otro lado, Pablo Aznar enfoca su TFM, entre otras cosas, a demostrar que su módulo (al que él denomina *Anomaly Detection Module*) es capaz de detectar ataques de integridad a contadores de la *smart grid*. En este TFG se ha desarrollado un detector basado en aprendizaje profundo, como en su TFM, pero utilizando algunas características (calificadores estadísticos) más que él. Los ataques *FDI* están inspirados en los que presenta en su trabajo.

A.2. Herramientas de minería de datos

En el *framework* (ver sección 3) desarrollado en este trabajo se han utilizado varias bibliotecas de *Python* para automatizar las tareas relacionadas con minería de datos. Algunas de ellas son: *NumPy*, *Pandas*, *Matplotlib*, *Scikit-learn*, *TensorFlow*, *Keras*, *pmdarima* y *SciPy*. En este apartado se realiza una breve descripción de las mismas.

NumPy [21] es una biblioteca de *software* con licencia BSD que extiende la funcionalidad básica matemática que ofrece *Python*, dotando al programador o programadora de estructuras de datos como vectores y matrices grandes multidimensionales (junto con sus operaciones) para facilitar trabajos analíticos y algebraicos al utilizar este lenguaje de programación.

Pandas [22] es una biblioteca de *software* con licencia BSD que extiende a *NumPy*, dotando al programador o programadora de estructuras de datos (junto con sus operaciones) para facilitar el trabajo con tablas numéricas, ficheros CSV y series temporales.

Matplotlib [23] es una biblioteca de *software* con licencia basada en PSF (licencia *custom*) orientada a la visualización de datos: genera todo tipo de gráficos útiles para analizar fácilmente grandes colecciones de datos. Está diseñada para integrarse con el uso de la biblioteca numérica *NumPy*.

Scikit-learn [24] es una biblioteca de *software* con licencia BSD orientada al aprendizaje automático. Proporciona gran variedad de algoritmos de clasificación, regresión y *clustering*. Está diseñada para integrarse con el uso de las bibliotecas numéricas y científicas *NumPy* y *SciPy*.

TensorFlow [25] es una biblioteca de *software* con licencia Apache 2.0 desarrollada por la empresa *Google*. Su primera versión fue lanzada el 9 de noviembre de 2015 y fue concebida para satisfacer las necesidades de la conocida empresa de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones análogos al aprendizaje y razonamiento usado por los humanos.

Keras [26] es una biblioteca de *software* con licencia MIT. Es capaz de ejecutarse sobre *TensorFlow*, entre otros, y está diseñada para facilitar el desarrollo de programas que trabajan con redes neuronales.

pmdarima [27] es una biblioteca de *software* con licencia MIT. Proporciona utilidades para trabajar más fácilmente con datos que constituyen series temporales.

SciPy [28] es una biblioteca de *software* con licencia BSD. Es muy útil para trabajar en ámbitos relacionados muy estrechamente con análisis matemático. Se integra muy bien con *NumPy*.

B. Gestión del proyecto

En este apartado se realiza un estudio de dos aspectos relacionados con la gestión de proyectos:

- Gestión de esfuerzos (subsección B.1): donde se detallan las horas estimadas y las horas empleadas, desglosándolas por categorías, y se muestran en una línea temporal para apreciar la distribución del esfuerzo desde el inicio hasta el final del proyecto.
- Gestión de riesgos (subsección B.2): donde se describe el análisis de los riesgos identificados (cálculo de la exposición a cada riesgo), la evaluación de los mismos (se priorizan y grafican los riesgos en una matriz NASA [29]) y la estrategia de mitigación para los cuatro riesgos de mayor exposición.

B.1. Gestión de esfuerzos

Gestionar el tiempo empleado en cualquier tarea es realmente importante, ya que como dijo William Thomson Kelvin: *"Lo que no se define no se puede medir. Lo que no se mide no se puede mejorar. Lo que no se mejora, se degrada siempre"*.

En este trabajo se ha creado una hoja de cálculo (ver Figura 13) en la que se registran las tareas: se les da un nombre, un estado (*to-do, in progress, done, discarded*), unas horas estimadas, una fecha de inicio y una categoría de las definidas en la Tabla 5. Al finalizar una tarea, se le asigna una fecha de fin y el número de horas realmente invertidas en la misma.

Esta información puede parecer simple, pero al tomarla periódicamente se pueden resolver preguntas como:

- ¿Se han realizado unas buenas estimaciones?
- ¿A qué tipo de tareas se dedica más tiempo?
- ¿Se sigue un ritmo sostenido de trabajo?

En total, se han registrado 346 tareas, resultando en 490 horas estimadas y 474,25 horas invertidas realmente, como muestra la Figura 9. Las estimaciones por tanto han sido bastante buenas, puesto que solo ha habido una desviación (sobreestimación) de 15,75 horas.

Tal y como se refleja en la Figura 10, la implementación abarca un 32,3 % del tiempo invertido (150 horas), seguida de la documentación, con un 22,5 % (104,7 horas). Las tareas relativas a la gestión del proyecto han costado 69,35 horas, mientras que a las de verificación y validación del *software* (*testing*) se les han dedicado 46,5 horas. Las cuatro categorías restantes (reunión, aprendizaje, análisis y diseño) suman en total un 20,3 % del tiempo invertido.

En las Figuras 11 y 12 se puede observar que las aportaciones a este trabajo se llevan realizando desde mayo del año 2020. Sin embargo, se aprecia un parón desde finales de ese año hasta principios de febrero del 2021 (ver Figura 11), que

Categoría	Descripción
Implementación	Desarrollo de código
Documentación	Redacción del trabajo realizado
Gestión	Gestión del proyecto
Testing	Desarrollo de pruebas para verificar y validar el *software*
Reunión	Reuniones desarrolladas
Aprendizaje	Estudio de tecnologías, técnicas y estado del arte
Análisis	Análisis de problemas, soluciones y resultados
Diseño	Diseño de soluciones a problemas complicados

Tabla 5: Descripción de las categorías.

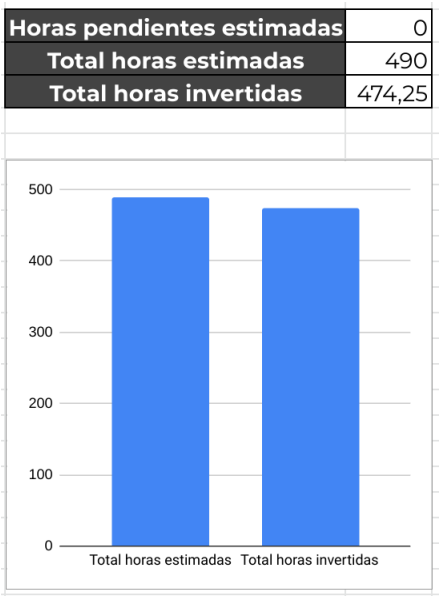


Figura 9: Horas pendientes, estimadas e invertidas.

coincide con las fechas de los exámenes del séptimo cuatrimestre. El resto del tiempo, se ha llevado un ritmo sostenido, pues en la mayoría de días de trabajo se han dedicado unas 3-4 horas y solo han sido unos pocos en los que se han dedicado más de 6 horas al proyecto (ver Figura 12).

Por último, en la Figura 14 se visualiza el tablero *Kanban* utilizado, mientras que en la Figura 15 se muestran algunos de los *issues* del proyecto.

TAREA	HORAS
REUNIÓN	25,5
APRENDIZAJE	33,45
ANÁLISIS	19
DISEÑO	16,25
IMPLEMENTACIÓN	150
TESTING	46,5
GESTIÓN	69,35
DOCUMENTACIÓN	104,7

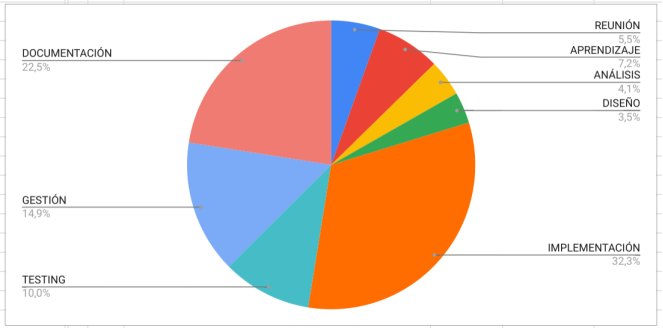


Figura 10: Distribución del tiempo invertido por categorías.

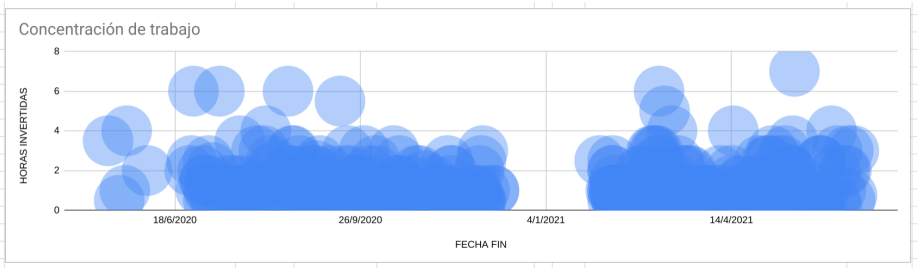


Figura 11: Concentración de trabajo.

Figura 12: Evolución de las horas dedicadas.

Detección de Ataques de Integridad en Redes Inteligentes mediante el uso de Técnicas de Aprendizaje Automático						
DESCRIPCIÓN DE LA TAREA	HORAS ESTIMADAS	ESTADO	FECHA INICIO	FECHA FIN	HORAS INVERTIDAS	TIPO TAREA
Curso de Python (Código Facilito)	3,5	DONE	12/5/2020	13/5/2020	3,5	APRENDIZAJE
Estudio código GitHub + pizarra Google Jamboard	0,5	DONE	19/5/2020	19/5/2020	0,5	APRENDIZAJE
Reunión inicial (Simona y José)	1	DONE	22/5/2020	22/5/2020	1	REUNIÓN
Primeros tests unitarios desarrollados	4	DONE	23/5/2020	23/5/2020	4	TESTING
Cambiada la estructura del proyecto. Tests integración	2	DONE	3/6/2020	3/6/2020	2	TESTING
Planteamiento tests integración. Análisis fuentes de datos	2	DONE	26/6/2020	26/6/2020	2	TESTING
Especificación y ejecución de tests de integración	2,5	DONE	27/6/2020	27/6/2020	2,5	TESTING
Tests de integración, code coverage, inspección estática PyCharm	6	DONE	28/6/2020	28/6/2020	6	TESTING
Lectura de documentos de Nacho	1	DONE	2/7/2020	2/7/2020	1	APRENDIZAJE
Reunión Nacho, Simona y José	2	DONE	3/7/2020	3/7/2020	2	REUNIÓN
CI Python	1,5	DONE	4/7/2020	4/7/2020	1,5	TESTING
CI Python. Problemas con los módulos	1,5	DONE	4/7/2020	4/7/2020	1,5	TESTING
Investigación codecov	0,5	DONE	6/7/2020	6/7/2020	0,5	TESTING
Reunión Simona y José	1	DONE	6/7/2020	6/7/2020	1	REUNIÓN
Solucionar problemas tests de integración	2,5	DONE	6/7/2020	6/7/2020	2,5	TESTING
Encontrado problema aritmética	1,5	DONE	7/7/2020	7/7/2020	1,5	TESTING
Refactorización, TODOs, mantenimiento repositorio	1	DONE	7/7/2020	7/7/2020	1	TESTING
Estudio PCA.py y redacción correo dudas	2,15	DONE	8/7/2020	8/7/2020	2,15	APRENDIZAJE
Reunión con Simona y José	0,75	DONE	10/7/2020	10/7/2020	0,75	REUNIÓN
Organización directorios	0,5	DONE	10/7/2020	10/7/2020	0,5	GESTIÓN
Preprocesado gas/energía, refactorización	6	DONE	12/7/2020	12/7/2020	6	IMPLEMENTACIÓN
Arreglar problemas CI	1,5	DONE	17/7/2020	17/7/2020	1,5	TESTING
Correo situación	0,5	DONE	17/7/2020	17/7/2020	0,5	GESTIÓN
Badrinath capítulo 4	1	DONE	18/7/2020	18/7/2020	1	APRENDIZAJE
Badrinath capítulo 5	1	DONE	19/7/2020	19/7/2020	1	APRENDIZAJE
Badrinath capítulo 5	1,5	DONE	20/7/2020	20/7/2020	1,5	APRENDIZAJE
Repaso PCA-DBSCAN	1	DONE	21/7/2020	21/7/2020	1	APRENDIZAJE
Repaso general Badrinath	1	DONE	23/7/2020	23/7/2020	1	APRENDIZAJE
Reunión Simona y José	1	DONE	23/7/2020	23/7/2020	1	REUNIÓN
Testing preprocessing	0,5	DONE	23/7/2020	23/7/2020	0,5	TESTING
Refactorización preprocessing	1	DONE	23/7/2020	23/7/2020	1	IMPLEMENTACIÓN
Diseño análisis consumidores (stationarity)	3,5	DONE	23/7/2020	23/7/2020	3,5	DISEÑO
Arreglar problemas con los tests (cambio de data_all_filtered)	1,5	DONE	24/7/2020	24/7/2020	1,5	TESTING
Pruebas y correo enviando resultados	1	DONE	24/7/2020	24/7/2020	1	TESTING
customer_analysis.py	3	DONE	1/8/2020	1/8/2020	3	IMPLEMENTACIÓN
Badrinath 5.5	1	DONE	2/8/2020	2/8/2020	1	APRENDIZAJE
Investigar diferencias meterIDs Energy	1,25	DONE	2/8/2020	2/8/2020	1,25	ANÁLISIS
Script para consultar número de meterIDs y su identificación	1,25	DONE	3/8/2020	3/8/2020	1,25	IMPLEMENTACIÓN
Refactorización script meterIDs	0,25	DONE	3/8/2020	3/8/2020	0,25	IMPLEMENTACIÓN
Envío correo actualización	0,75	DONE	3/8/2020	3/8/2020	0,75	GESTIÓN
customer_analysis.py (ADF, KPSS)	1,5	DONE	5/8/2020	5/8/2020	1,5	IMPLEMENTACIÓN
preprocessing último paso	3	DONE	5/8/2020	5/8/2020	3	IMPLEMENTACIÓN
Arreglar problemas preprocessing (EnergyDataWeek 17 crash)	4	DONE	6/8/2020	6/8/2020	4	IMPLEMENTACIÓN
Comprobación preprocessing, customer_analysis	1	DONE	7/8/2020	7/8/2020	1	TESTING
Documentación: overleaf, correo	0,5	DONE	7/8/2020	7/8/2020	0,5	DOCUMENTACIÓN
https://www.youtube.com/watch?v=9f-GarcDY58	2	DONE	8/8/2020	8/8/2020	2	APRENDIZAJE
https://www.youtube.com/watch?v=9f-GarcDY58	2,5	DONE	9/8/2020	9/8/2020	2,5	APRENDIZAJE
SARIMAX + Badrinath (5.4.)	0,5	DONE	10/8/2020	10/8/2020	0,5	APRENDIZAJE

Figura 13: Hoja de cálculo de seguimiento de esfuerzos.

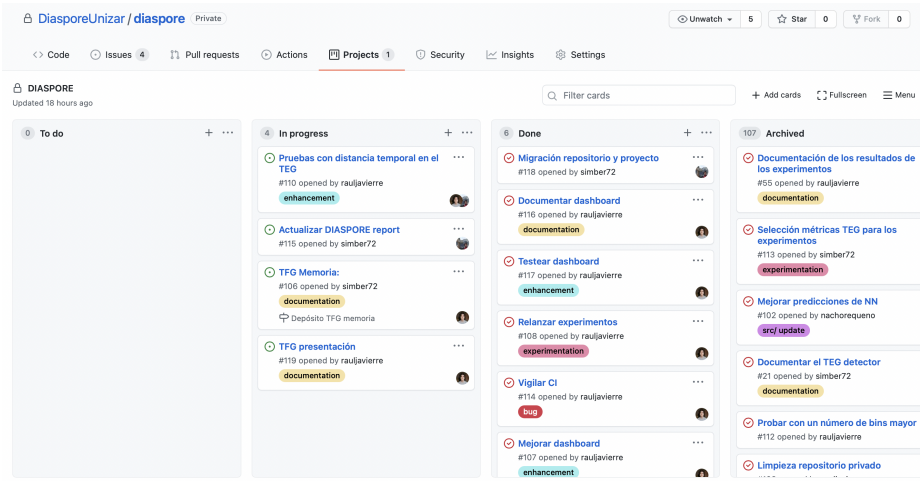


Figura 14: Tablero *kanban* utilizado.

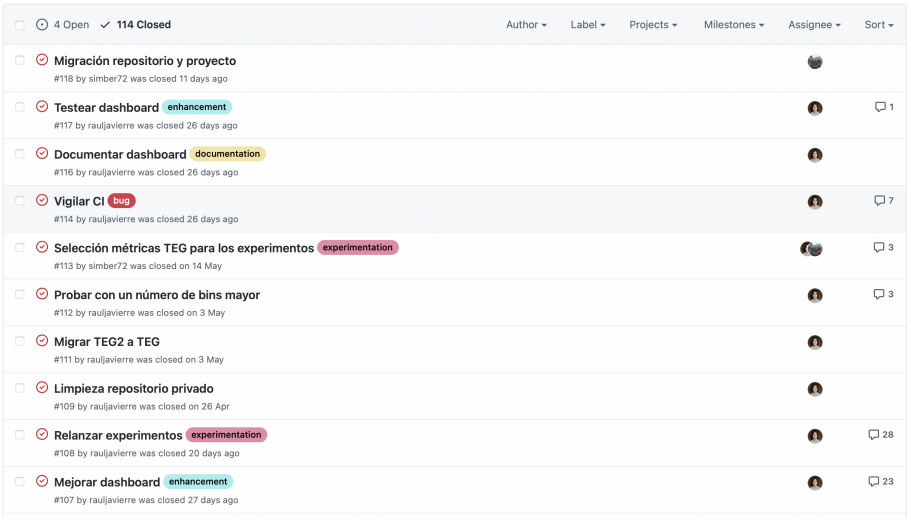


Figura 15: *Issues* de *GitHub*.

B.2. Gestión de riesgos

En todo proyecto hay un conjunto de riesgos que pueden materializarse (y un proyecto software no es una excepción). Son más o menos importantes dependiendo de la probabilidad de que se materialicen y del impacto que provoquen al hacerlo. Por ello, es necesario priorizarlos en base a dicha importancia, puesto que la materialización de algún riesgo podría complicar seguir adelante con el proyecto (en muchos casos suele suponer el fin del mismo).

Partiendo de la base de que nunca se puede reducir ningún riesgo a cero, la idea principal es reducir los más preocupantes a niveles aceptables y monitorizar su estado periódicamente. Es por ello que se detalla a continuación el plan de gestión de riesgos del presente trabajo.

El plan incluye una tabla (ver Tabla 6) donde se han identificado 22 riesgos, clasificados en 4 grandes categorías. Por orden, las categorías son: riesgos del proceso, riesgos relacionados con la Ingeniería del Software, riesgos técnicos y riesgos del personal. Para cada de uno los riesgos identificados, se ha evaluado la probabilidad de que se materialice y el impacto que tendría al hacerlo. Al multiplicar estos dos valores se genera un producto, denominado *exposición* (probabilidad x impacto = exposición). Tanto para cuantificar la probabilidad como para cuantificar el impacto se utilizan números entre 1 y 5 (Tabla 7), distribuyendo así los riesgos en una matriz NASA [29], mostrada en la Tabla 8.

ID	DESCRIPCIÓN	PROBABILIDAD	IMPACTO	EXPOSICIÓN
1	El proceso no está bien definido	1	5	5
2	El proceso no está soportado por herramientas	1	3	3
3	El proceso no está gestionado mediante el uso de métricas	1	3	3
4	El proceso no se revisa (mejora continua)	2	2	4
5	No se utiliza un VCS	1	5	5
6	No se utilizan herramientas de diseño	2	3	6
7	No se prueba el código (explícitamente con tests)	2	2	4
8	No se utiliza una herramienta de integración continua	1	3	3
9	Posibilidad de aparición de deuda técnica	2	2	4
10	**El código no está correctamente documentado**	**3**	**3**	**9**
11	La arquitectura no está correctamente documentada	1	4	4
12	**El codebase no sigue ningún tipo de organización**	**3**	**3**	**9**
13	El objetivo final no está claro, es cambiante	3	2	6
14	No se utiliza tecnología asentada y documentada	1	3	3
15	No se utiliza tecnología conocida por el equipo de desarrollo	2	1	2
16	El dominio del problema no es conocido por el equipo de desarrollo	2	1	2
17	El software requiere ser ejecutado en hardware heterogéneo	3	1	3
18	**Se requiere mucho tiempo para ejecutar el software**	**5**	**3**	**15**
19	No se tiene una combinación correcta de habilidades	1	4	4
20	El personal no está comprometido	1	4	4
21	**El personal trabaja en muchos proyectos a la vez**	**4**	**3**	**12**
22	El personal no está capacitado	1	3	3

Tabla 6: Riesgos identificados.

PROBABILIDAD	
CLASIFICACIÓN	**DESCRIPCIÓN**
1	Remota o inexistente
2	Poca
3	Media
4	Alta
5	Muy alta

IMPACTO	
CLASIFICACIÓN	**DESCRIPCIÓN**
1	Remoto o inexistente
2	Necesidad de invertir hasta un 5% de recursos adicionales si el riesgo se materializa
3	Necesidad de invertir hasta un 5% de recursos adicionales si el riesgo se materializa
4	Necesidad de invertir hasta un 15% de recursos adicionales si el riesgo se materializa
5	Necesidad de invertir más de un 15% de recursos adicionales si el riesgo se materializa

Tabla 7: Descripción de las clasificaciones de probabilidades e impacto.

IMPACTO						
5	2					
4	3					
3	5	1	2	1	1	
2		3	1			
1		2	1			
	1	2	3	4	5	**PROBABILIDAD**

Tabla 8: Matriz NASA.

ID	ESTRATEGIA DE MITIGACIÓN
10	Comprobar periódicamente (cada 4 iteraciones) que el código tiene comentarios suficientes como para entender en todo momento qué es lo que está haciendo
12	Establecer pautas de organización del codebase en el VCS. Comprobar periódicamente (cada 4 iteraciones) que el código está correctamente organizado. Si no lo está, proceder a reorganizarlo en base a las pautas establecidas
18	Procurar, dentro de lo posible, minimizar el coste computacional de los scripts. Analizar si es necesario lanzar los experimentos con tantos meterIDs
21	Mantener comunicación fluida y de banda ancha a través de reuniones semanales por videollamada. Evitar comunicación por correo electrónico/GitHub dentro de lo posible (está bien para documentar observaciones/resultados). Esto desemboca en una optimización del tiempo

Tabla 9: Riesgos de mayor exposición y estrategias de mitigación.

C. Conjuntos de datos utilizados

En este anexo se entra más en detalle en cuanto a los dos *datasets* de uso de electricidad (subsección C.1) y de gas (subsección C.2). Se describe cada uno de ellos: qué datos guarda, cuántos datos contiene, bajo qué circunstancias se pueden utilizar estos datos, etc. También se describe el preprocesado de estos datos, es decir: cuál es el proceso desde que los datos se descargan de la fuente de datos hasta que pueden utilizarse directamente en el módulo de experimentación del *framework* implementado.

Los dos conjuntos de datos utilizados en este trabajo se obtienen del *Comission for Energy Regulation (CER)*, concretamente desde el centro de adquisición de datos *Irish Social Science Data Archive (ISSDA)* [5].

Se ha realizado una petición de uso al *ISSDA*, puesto que son ellos los que proporcionan el permiso de utilización de los datos. En concreto, el *ISSDA* permite usar los *datasets*, siempre y cuando:

1. Se les envíe por correo electrónico un formulario de petición de uso (disponible en su sitio web).

2. La petición enviada se acepte: es entonces cuando proporcionan el acceso al repositorio de datos.

3. Se utilicen los *datasets* para propósitos de investigación o educacionales. Muy excepcionalmente permiten su uso para proyectos comerciales.

C.1. ISSDA CER (Electricidad)

Este *dataset* recoge los datos de uso de electricidad de 6.445 contadores en Irlanda durante varias decenas de semanas, realizando un muestreo cada 30 minutos. Estos contadores pueden pertenecer a tres tipos: hogares (*residential*), pequeñas y medianas empresas (*small and medium enterprises (SME))* y no clasificados (*unclassified*), como se muestra en la Figura 16.

El *dataset* contiene una cantidad de datos importante: 6 ficheros de texto que en total suman 2,5 GiB. Estos ficheros contienen una línea por lectura. Cada lectura consta del identificador del contador, del tiempo de lectura y de la cantidad de energía (en kWh) consumida desde la última lectura. Además, hay disponibles unos ficheros de documentación que indican a qué tipo/clasificación pertenece cada contador.

La cantidad de información contenida en el *dataset* es abrumadora. Sin embargo, no es completa. En principio, debería haber un registro cada 30 minutos para cada uno de los contadores, pero no es así, por lo que ha sido necesario pasar por un proceso de filtrado de datos (*data filtering*), además de reestructurar los originales para adaptarlos a las necesidades del trabajo.

Esta es la secuencia de pasos que se ha diseñado e implementado para pasar del conjunto de datos original de electricidad a unos datos que se pueden utilizar en el *framework*:

1. Utilizar el *dataset* original (`data_original`) para generar un nuevo *dataset* (`data_sorted`) en el que todos los registros de los contadores estén ordenados por *meterID (identificador del contador)* y por el *timestamp* de lectura del contador.

2. Transformar `data_sorted`, convirtiéndolo a un conjunto de datos llamado `data_all`, en el que se separan los datos semanalmente. Este *dataset* está formado por tantos ficheros como semanas de información tenía el conjunto de datos original (`data_original`).

3. Filtrar `data_all`, eliminando todos aquellos *meterIDs* que no tengan todos los registros esperados (336 por semana para todas las semanas). Este paso genera el conjunto de datos `data_all_filtered` (ver Figura 16).

4. Generar los ficheros de datos de entrenamiento *(training dataset)* y de validación *(testing dataset)* utilizando `data_all_filtered`. En concreto, se generan ficheros cuyo nombre se describe con la sintaxis BNF:

   `meterID_[nombreDelAtaque_]`[4]`semanaInicio_semanaFin.csv`.

Las semanas[5] de inicio y de fin para el *training dataset* son la 0 y la 60, respectivamente, mientras que para el *testing dataset* son la 61 y la 75. Los ficheros que representan comportamientos de ataque se generan modificando los datos de consumo originales conforme al tipo de ataque considerado, por lo que han de pasar por un paso *extra*.

Por último, remarcar que se generan tantos ficheros de entrenamiento y de validación como el producto de escenarios de consumo y *meterIDs* considerados. Por ejemplo, considerando 5 escenarios de consumo y 20 *meterIDs*, obtendremos 100 ficheros de entrenamiento y 100 ficheros de validación. Para cada *meterID*, 5 ficheros de entrenamiento incluirán sus consumos de las semanas 0 a la 60 (modificados o no en función de si el escenario es de ataque o no, respectivamente) y 5 ficheros de validación incluirán sus consumos de las semanas 61 a la 75 (modificados o no). Destacar que se necesitan generar ficheros de entrenamiento (semanas 0 a la 60) de todos los escenarios para entrenar la red neuronal (aprendizaje supervisado).

Con el filtrado (paso 3) el número de *meterIDs* con datos completos en las semanas consideradas se redujo de 6.445 a 4.626. Sin embargo, se decidió que 1.000 *meterIDs* eran suficientes para realizar los últimos experimentos detallados en la sección 4, por lo que solo se generaron los datos necesarios para entrenar y para validar con 1.000 del total filtrado (paso 4). Badrinath [7], consideró 500 *meterIDs* en su estudio: 404 *residentials*, 36 *SMEs* y 60 *unclassified*. No se sabía exactamente cuáles escoger, por lo que se decidió respetar su proporción y utilizar sí o sí el contador 1330, ya que él seguro que lo utilizó en sus estudios puesto que aparece en su tesis (así podían compararse los resultados con ese contador en concreto). Los 999 restantes fueron escogidos de forma aleatoria, siguiendo la proporción mencionada.

[4] Si no representa ningún ataque, se omite.

[5] La semana 36 del conjunto de datos original no se tuvo en cuenta por falta de datos.

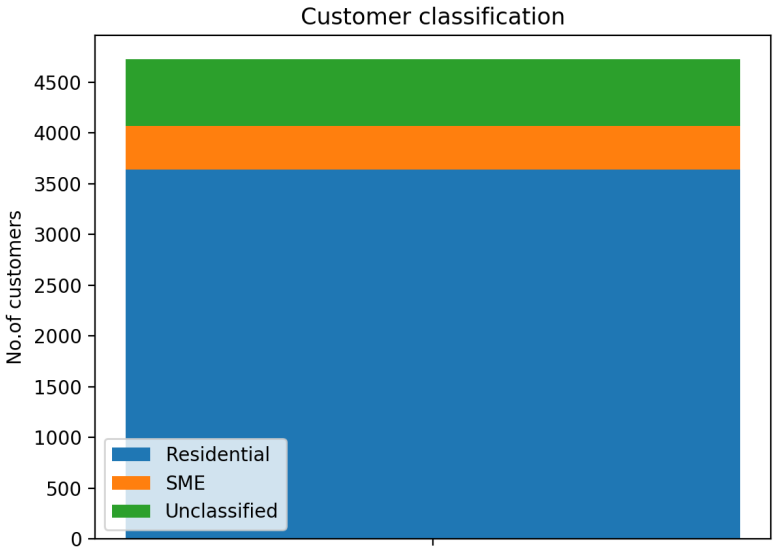


Figura 16: Proporción de *meterIDs* en el *dataset* filtrado de electricidad.

C.2. ISSDA CER (Gas)

Los datos de consumo de gas provienen exactamente de la misma fuente que los de la electricidad, por tanto, comparten las mismas restricciones en cuanto a permisos de utilización, etc. Sin embargo, existen unas cuantas diferencias en cuanto a lo mencionado en la subsección C.1.

En primer lugar, este *dataset* es más pequeño y contiene información sobre un menor número de contadores (1.576 contadores). Por otro lado, de forma similar al *dataset* de electricidad, los contadores están clasificados. En este caso se clasifican en base al tipo de contrato asociado al contador, es decir: factura bimensual, factura mensual, factura bimensual + dispositivo extra, factura mensual + dispositivo extra, control (ver Figura 17). Finalmente, el conjunto de datos original ya está agrupado por semanas, dando lugar a 78 ficheros de texto (702,2 MiB en total). Por consiguiente, puesto que ya están agrupados por semanas, se simplifica la secuencia de pasos necesaria para transformar el conjunto de datos original a unos datos que se pueden utilizar en el *framework* implementado. Concretamente, solamente se siguen los últimos dos pasos (3 y 4) de la secuencia mencionada en la subsección C.1. En el último paso, no hay diferencias con respecto al *dataset* de electricidad, excepto que se utilizan los datos de las semanas[6] 0 a la 60 para datos de entrenamiento y los datos de

[6]Las semanas 40, 42, 68 y 76 del conjunto de datos original no se tuvieron en cuenta por falta de datos.

las semanas 61 a la 77 para datos de validación. Para este *dataset* también se utiliza un subconjunto de *meterIDs* para la experimentación, como se detalla en la Tabla 10.

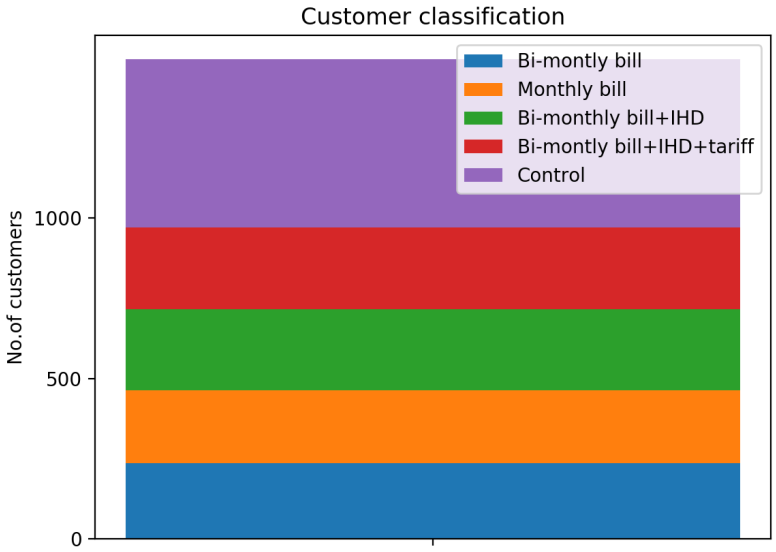


Figura 17: Proporción de *meterIDs* en el *dataset* filtrado de gas.

Tipo contrato	Número *meterID*
factura bimensual	158
factura mensual	152
factura bimensual + dispositivo extra	168
factura mensual + dispositivo extra	170
control	352
Total	1.000

Tabla 10: Distribución de *meterIDs* utilizados en la experimentación.

D. *Framework* desarrollado

En este anexo se entra más en detalle en algunos de los módulos pertenecientes al *framework* presentado en la sección 3.

D.1. Módulo de detectores

Todos los detectores desarrollados en este trabajo heredan de la clase abstracta *Detector* con la finalidad de uniformar su implementación, favoreciendo así su mantenibilidad, su extensibilidad y su integración con el resto del *framework*.

La clase abstracta *Detector* cuenta con 7 métodos cuya signatura se muestra en la Figura 18 y que se describen a continuación:

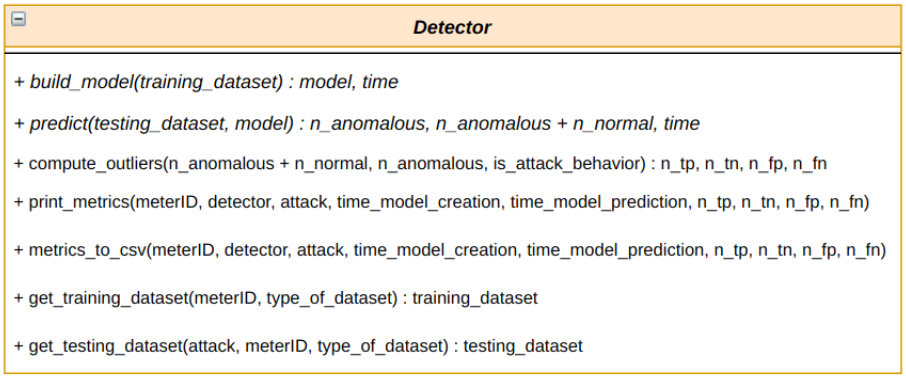


Figura 18: Clase abstracta Detector.

1. *build_model:* método abstracto, lo que implica que las clases hijas deben hacer *override* del método (sobreescribirlo/redefinirlo) obligatoriamente para poder ser instanciadas. La operación consiste en que se cree un modelo (*model*) a partir de un conjunto de datos de entrenamiento (*training_dataset*) y se devuelva tanto el modelo como el tiempo empleado en realizar la operación (*time*).

2. *predict:* método abstracto, lo que implica que las clases hijas deben hacer *override* del método obligatoriamente para poder ser instanciadas. La operación consiste en que dado un conjunto de datos de validación (*testing_dataset*) y un modelo (*model*), se devuelva: el número de muestras que el modelo considera anómalas (*n_anomalous*), el número de muestras totales (*n_anomalous + n_normal*) y el tiempo empleado en realizar la operación (*time*).

3. *compute_outliers:* método que calcula la matriz de confusión a partir del número de muestras totales (*n_anomalous + n_normal*), el número de

muestras marcadas como anómalas (*n_anomalous*) y un *booleano* que especifica si el escenario considerado es de ataque o no (*is_attack_behavior*). El método devuelve los elementos de la matriz de confusión, es decir: el número de verdaderos positivos (*n_tp*), verdaderos negativos (*n_tn*), falsos positivos (*n_fp*) y falsos negativos (*n_fn*). Notar que en los experimentos llevados a cabo en este trabajo un mismo conjunto de datos de validación no incluye datos normales y datos de ataque mezclados, por lo que la matriz de confusión generada al validar un escenario que corresponde con un ataque nunca podrá tener verdaderos negativos ni falsos positivos. Ocurre lo mismo al validar un comportamiento normal, nunca tendrá verdaderos positivos ni falsos negativos.

4. *print_metrics:* método que imprime por salida estándar información relativa a un experimento. En concreto: el contador que ha registrado los datos de entrenamiento/validación utilizados (*meterID*), el detector a validar (*detector*), el tipo de escenario de consumo (que puede ser de *ataque* o *normal*), el tiempo de creación del modelo de predicción (*time_model_creation*), el tiempo empleado para realizar la predicción (*time_model_prediction*) y los elementos de la matriz de confusión (*n_tp, n_tn, n_fp, n_fn*).

5. *metrics_to_csv:* método que escribe en un fichero información relacionada con un experimento en formato *Comma-Separated Values* (`CSV`). En concreto: el contador que ha registrado los datos de entrenamiento/validación utilizados (*meterID*), el detector a validar (*detector*), el tipo de escenario de consumo (*ataque*), el tiempo de creación del modelo de predicción (*time_model_creation*), el tiempo empleado para realizar la predicción (*time_model_prediction*) y los elementos de la matriz de confusión (*n_tp, n_tn, n_fp, n_fn*).

6. *get_training_dataset:* método que, dado un contador (*meterID*) y el tipo de conjunto de datos utilizado (*type_of_dataset*), devuelve un conjunto de datos de entrenamiento (*training_dataset*).

7. *get_testing_dataset:* método que, dado un contador (*meterID*) y el tipo de conjunto de datos utilizado (*type_of_dataset*), devuelve un conjunto de datos de validación (*testing_dataset*).

Excepto *build_model* y *predict* (que son abstractos), todos los demás métodos están implementados en la clase *Detector*. Siempre se puede hacer *override* del resto de métodos (dentro de la semántica de las operaciones), es decir, dentro del *molde*, se proporciona un grado de libertad al desarrollador o desarrolladora que vaya a utilizar este módulo.

Como este módulo tiene bastantes reglas y es una pieza software básica del *framework* desarrollado, está acompañado de un fichero `README` (véase Figura 19) para facilitar trabajar con él a futuros desarrolladores y desarrolladoras.

How to add and test a new detector

1. Create a class that encapsulates your detector in the directory src/detectors/. This class must inherit from the abstract class Detector defined in this directory and needs to override its abstract methods "build_model" and "predict".

```
from .Detector import Detector

class MyCustomDetector(Detector):

    def build_model(self, training_dataset):
        t0 = time()
        # Use the training dataset
        return model, time() - t0

    def predict(self, testing_dataset, model):
        t0 = time()
        # Use the testing dataset and the model
        return number_of_anomalous, number_of_anomalous + number_of_normal, time() - t0
```

   Note that there are another five methods ("compute_outliers", "print_metrics", "metrics_to_csv", "get_training_dataset" and "get_testing_dataset") declared and implemented in Detector (they are not abstract methods). You can override any of them in your detector class if you need it, but respect the semantics and the inputs/outputs, or it will crash later! You can see some examples:

   - src/detectors/PCADBSCAN.py: I needed to modify the training/testing dataset.
   - src/detectors/MinAverage.py: I needed to modify the compute_outliers because I return my predictions in a dict instead in a numerical value.

   However, I recommend only overriding "build_model" and "predict" methods if you start designing your class using this schema.

2. Create a new "elif" branch in the static method DetectorFactory.create_detector

```
class DetectorFactory:

@staticmethod
def create_detector(detector):
    if detector == "Min-Avg":
        return MinAverage()
    elif detector == "JSD":
        return JSD()
    elif detector == "MyCustomDetectorName":
        return MyCustomDetector()
    else:
        raise KeyError("Detector " + detector + " not found. You must add a conditional branch in /src/detectors/Dete
```

   Don't forget to import the module you've just created in the step 1!

```
from .MyCustomDetector import MyCustomDetector
```

3. Add the name that you've indicated in the DetectorFactory.create_detector elif branch to the list named "list_of_detectors" in the file detector_comparer

```
list_of_detectors = ["MyCustomDetectorName", ..., "ARIMA", "ARIMAX", "JSD", "KLD", "Min-Avg"]
```

4. And run...

```
export PYTHONPATH=.
python3 src/experiments/detector_comparer.py <type_of_dataset>
```

Figura 19: README del módulo de detectores.

D.2. Módulo de lanzamiento de los experimentos

Este módulo es el encargado de orquestar el lanzamiento de los experimentos (ver Algoritmo 1). Está formado por un *programa*, cuyas precondiciones son:

- Que se haya declarado una lista de contadores con los que experimentar para el conjunto de datos indicado.
- Que los escenarios indicados estén generados para el conjunto de datos. Esto es: si se pide que se realicen experimentos con el escenario *Avg* del *dataset ISSDA-CER de electricidad*, los ficheros que representan dicho escenario deben haber sido generados previamente.
- Que exista una implementación de los detectores indicados y que la factoría de detectores sepa cómo construirlos.

Si se cumplen todas esas precondiciones, se generará un fichero de resultados de experimentos, que contendrá las métricas de la Tabla 2 para cada uno de ellos.

Este *programa* se apoya (está muy acoplado) al módulo *detectors*, puesto que, en secuencia:

1. Obtiene el detector utilizando la factoría.
2. Utiliza la operación *get_training_dataset* del detector para obtener el conjunto de datos de entrenamiento.
3. Utiliza la operación *build_model* del detector para obtener el modelo entrenado y el tiempo de creación del modelo. Para ello, le pasa el conjunto de datos del anterior paso.
4. Utiliza la operación *get_testing_dataset* del detector para obtener el conjunto de datos de validación.
5. Utiliza la operación *predict* del detector para obtener el número de muestras marcadas como anómalas, el número de muestras consideradas en la predicción y el tiempo empleado para realizar dicha predicción. Para ello, le pasa tanto el modelo como el conjunto de datos de validación.
6. Utiliza la operación *compute_outliers* del detector para obtener las cuatro componentes de la matriz de confusión. Para ello, le pasa el número de muestras marcadas como anómalas, el número de muestras consideradas en la predicción (número de muestras marcacas como anómalas + número de muestras marcadas como normales) y un parámetro para indicar si se trataba de un escenario de ataque o no.
7. Utiliza la operación *print_metrics* del detector para imprimir por pantalla las métricas básicas de la Tabla 2.
8. Utiliza la operación *metrics_to_csv* del detector para escribir en un fichero de formato CSV las métricas básicas de la Tabla 2.

D.3. Módulo de análisis de los experimentos

Este módulo tiene la finalidad de mostrar un *dashboard* con información de resultados de experimentos. Sin embargo, la tarea de elegir qué hay que mostrar y cómo hay que mostrarlo no es baladí, por lo que se dibujó un primer boceto (ver Figura 20) que se discutió en una de las reuniones semanales. Al final se sustituyeron los diagramas de caja (*boxplot*) por diagramas de barras, pero el resto de componentes siguen siendo los mismos, es decir, se reutilizan para cambiar entre análisis global y análisis unitario. La separación entre estos dos niveles de análisis se consigue mediante la interacción del usuario (éste elige todos los escenarios o uno en concreto, además del conjunto de datos a analizar para modificar la configuración del *dashboard*). Las Figuras 4, 5, 6, 7, 8, 25 y 26 muestran componentes de la interfaz final.

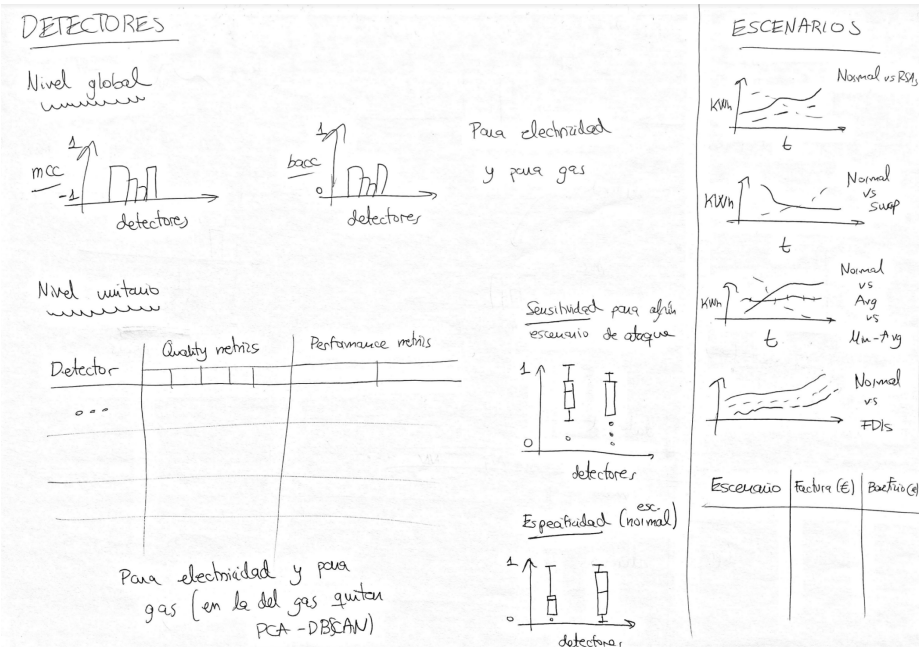


Figura 20: Primer boceto del *dashboard*.

La arquitectura de este módulo sigue el patrón modelo-vista-controlador (MVC). Por un lado, se ha desarrollado una clase por cada componente. Esto es, cada gráfica y tabla es una clase, encargada de gestionar su propio estado. Todas estas clases heredan de una clase abstracta *Component*, para así facilitar su uso, tratándolas todas de la misma manera desde el programa *dashboard*. También se ha añadido un nivel de abstracción adicional, construyendo una factoría de componentes para poder construirlos de forma idempotente a partir de sus nombres, independientemente de lo complicada que sea su instanciación. Esta estructura se puede apreciar en la Figura 21.

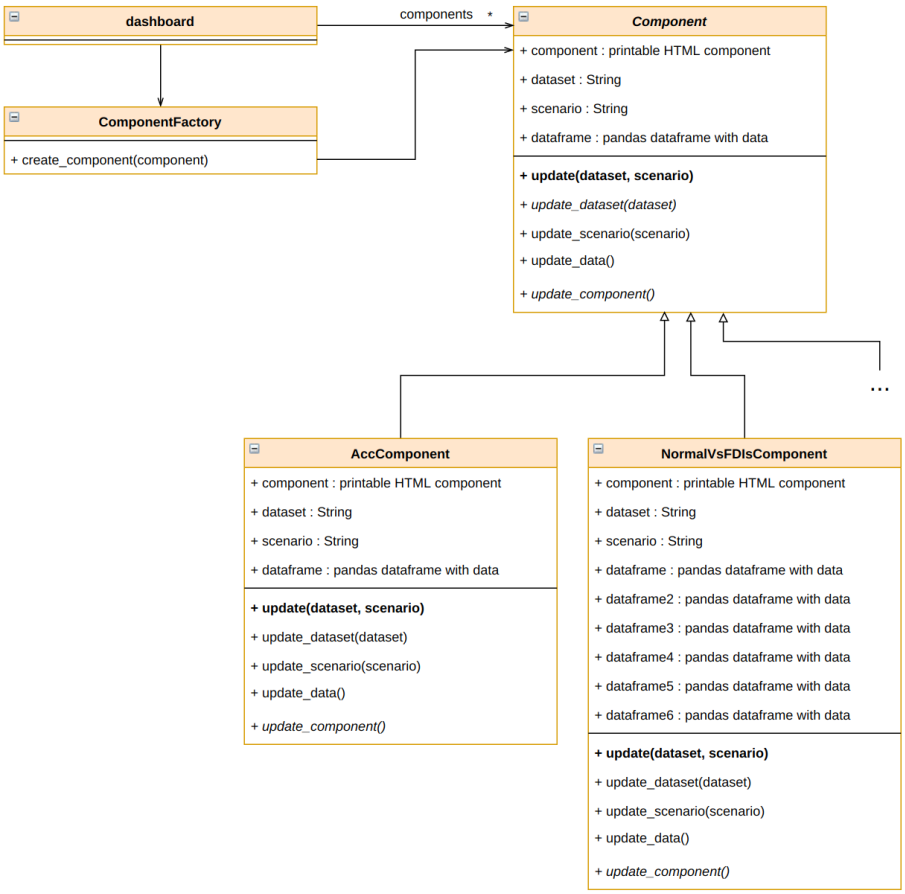


Figura 21: Estructura de clases del *dashboard*.

Cada componente guarda:

- El propio componente imprimible en HTML en el atributo *component*, que es de la clase que se prefiera (siempre y cuando pueda imprimirse en este formato). En el presente trabajo se han utilizado los tipos *dash_core_components.Graph* y *dash_table.DataTable*.
- El nombre del *dataset* seleccionado actualmente: atributo *dataset*.
- El nombre del *escenario* seleccionado actualmente (puede ser *All*): atributo *scenario*.
- Uno o varios *dataframes* de la biblioteca *Pandas*, para almacenar los datos correspondientes a los resultados para el *dataset* y *escenario* seleccionados.

Para actualizar el estado del componente, solo hay que invocar a su operación *update*, pasándole un *dataset* y un *escenario*. El resto de operaciones son

llamadas por ésta, cambiando el nombre del *dataset*, el nombre del *escenario* y los datos actuales. La operación *update_component* es abstracta, puesto que es dependiente de cada componente: diferente implementación si es una tabla, gráfica, etc.

El programa *dashboard*, que es el que lanza el servidor *web*, permite visualizar una serie de componentes (vista). Como se ha comentado antes, cada uno de estos es el encargado de mantener su estado (modelo). Los controladores son los dos desplegables para seleccionar un conjunto de datos y/o un escenario, desde la interfaz (ver Figura 4). Cuando se dispara la acción de uno de estos controladores porque se ha cambiado el conjunto de datos o el escenario, el programa *dashboard* recorre la lista de todos los componentes que guarda, y llama a su operación *update*, provocando una actualización secuencial del modelo de todos los componentes, que se refleja en la vista de la interfaz.

D.4. Módulo de infraestructura de *testing*

Las 6 máquinas virtuales utilizadas para ejecutar los *tests en la nube* con el apoyo de la herramienta de integración continua *GitHub Actions* están configuradas con 3 versiones diferentes de *Ubuntu* y 2 versiones diferentes de *Python*. La Tabla 11 muestra la configuración de cada una.

Máquina virtual (MV)	Sistema Operativo	Versión Python
MV_1	Ubuntu 20.04	Python 3.7
MV_2	Ubuntu 20.04	Python 3.8
MV_3	Ubuntu 18.04	Python 3.7
MV_4	Ubuntu 18.04	Python 3.8
MV_5	Ubuntu 16.04	Python 3.7
MV_6	Ubuntu 16.04	Python 3.8

Tabla 11: Máquinas virtuales utilizadas para ejecutar los *tests*.

Si los *tests* en cualquiera de estas máquinas virtuales falla, se notifica al usuario que haya realizado la actualización del repositorio. Además, la actualización se marca como fallida en la historia del repositorio. En la Figura 22 se muestra un ejemplo en el que los *tests* pasaron satisfactoriamente en las seis máquinas virtuales.

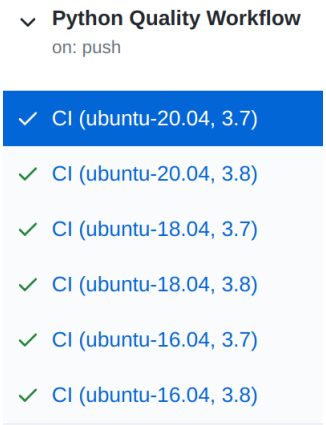


Figura 22: Actualización correcta en las 6 máquinas virtuales.

La configuración del *pipeline* de integración continua se ha desarrollado con el *script* detallado en la Figura 23, mientras que los resultados de los *tests* se imprimen como se muestra en la Figura 24, mostrando, para cada fichero: el número de líneas que contiene, el número de líneas no cubiertas por los *tests*, el porcentaje de líneas cubiertas y el listado de líneas no cubiertas.

```yaml
name: Python Quality Workflow

on: [push]

jobs:

  CI:

    runs-on: ${{ matrix.os }}

    strategy:
      matrix:
        os: [ubuntu-20.04, ubuntu-18.04, ubuntu-16.04]
        include:
          - os: ubuntu-20.04
          - os: ubuntu-18.04
          - os: ubuntu-16.04
        python-version: ['3.7', '3.8']

    steps:
      # Clone the repo
      - uses: actions/checkout@v2

      # Set up Python
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}

      # Install requirements and run tests
      - name: Install requirements and run tests
        run: |
          export PYTHONPATH=.
          pip install -r requirements.txt
          pytest --cov-report term-missing --cov=src
```

Figura 23: Configuración de la integración continua.

```
Name                                                  Stmts   Miss  Cover   Missing
-------------------------------------------------------------------------------------
src/__init__.py                                           2      0   100%
src/analytics/MetadataInformation.py                     78     78     0%   1-120
src/analytics/__init__.py                                72     61    15%   16-21, 25-40, 44-46, 50-52, 55-107
src/analytics/components/AccComponent.py                  6      0   100%
src/analytics/components/BaccComponent.py                 6      0   100%
src/analytics/components/Component.py                    24      1    96%   42
src/analytics/components/ComponentFactory.py             42      0   100%
src/analytics/components/DetectorsTableComponent.py       5      0   100%
src/analytics/components/MccComponent.py                  6      0   100%
src/analytics/components/NormalVsAveragesComponent.py    26      0   100%
src/analytics/components/NormalVsFDIsComponent.py        41      0   100%
src/analytics/components/NormalVsRSAsComponent.py        31      0   100%
src/analytics/components/NormalVsSwapComponent.py        21      0   100%
src/analytics/components/ScenariosTableComponent.py      53      0   100%
src/analytics/components/TbComponent.py                   6      0   100%
src/analytics/components/TnrComponent.py                  6      0   100%
src/analytics/components/TpComponent.py                   6      0   100%
src/analytics/components/TprComponent.py                  6      0   100%
src/analytics/components/__init__.py                      0      0   100%
src/analytics/dashboard.py                               26     26     0%   11-226
src/detectors/ARIMA.py                                   26     15    42%   21-22, 25-28, 35-81, 85-88, 94-99
src/detectors/ARIMAX.py                                  31     18    42%   24-25, 28-31, 38-91, 96-101, 107-112
src/detectors/Detector.py                                58      2    97%   27, 31
src/detectors/DetectorFactory.py                         41      0   100%
src/detectors/FisherJenks.py                             15      0   100%
src/detectors/IsolationForest.py                         29      0   100%
src/detectors/JSD.py                                     66      0   100%
src/detectors/KLD.py                                     66      0   100%
src/detectors/KMeans.py                                  32      0   100%
src/detectors/MinAverage.py                              43      0   100%
src/detectors/MiniBatchKMeans.py                         32      0   100%
src/detectors/NN.py                                      63      0   100%
src/detectors/PCADBSCAN.py                              182      5    97%   89-90, 98, 119, 287
src/detectors/RD.py                                      19     19     0%   10-39
src/detectors/TEG.py                                    111      0   100%
src/detectors/TEGModules/__init__.py                      0      0   100%
src/detectors/TEGModules/graph_comparison.py            347      4    99%   63, 67, 532, 626
src/detectors/TEGModules/graph_discovery.py              30      0   100%
src/detectors/__init__.py                                 0      0   100%
src/experiments/__init__.py                               2      0   100%
src/experiments/detector_comparer.py                     35      2    94%   94-95
src/preprocessing/__init__.py                             0      0   100%
src/preprocessing/issdacer.py                           192    192     0%   37-361
src/preprocessing/training_and_testing_generator.py     159    159     0%   18-264
src/tests/__init__.py                                     0      0   100%
src/tests/test_component.py                              14      1    93%   367
src/tests/test_component_factory.py                      12      1    92%   42
src/tests/test_detector.py                               82      1    99%   1166
src/tests/test_detector_comparer.py                      17      1    94%   34
src/tests/test_detector_factory.py                       52      1    98%   78
-------------------------------------------------------------------------------------
TOTAL                                                  2219    587    74%

================ 1411 passed, 1 warning in 868.91s (0:14:28) ==================
```

Figura 24: Resultados de los *tests* en la integración continua.

E. Resultados

En este anexo se explican las métricas de calidad utilizadas en el análisis comparativo de los detectores y se detallan los resultados globales de los dos últimos planes de experimentos ($\mathcal{E}_7$ y $\mathcal{E}_8$ de la Tabla 4) en base a estas métricas.

La métrica *acc (accuracy)* tiene un dominio del 0 al 1 y mide cómo de frecuente es que las predicciones de un modelo sean correctas. En el caso del presente trabajo, en el que solo hay dos clases posibles (comportamiento normal/anómalo), un *accuracy* de 0,5 es igual de útil que lanzar una moneda al aire, mientras que un *acurracy* cercano a 1 sería un buen resultado.

La métrica *bacc (balanced accuracy)* tiene el mismo dominio, es decir, su valor está entre 0 y 1, siendo los valores cercanos a 1 los que representan unos buenos resultados. Esta métrica aporta incluso más valor, puesto que tiene más sentido tal y como se ha planteado la metodología de experimentación, en la que se prueba con más muestras anómalas que normales (11 anómalas por cada 1 normal).

La métrica *mcc (Matthews Correlation Coefficient)*, con un dominio en el intervalo [-1, 1], también es una buena métrica para tratar experimentos con clases no balancedas. En este caso, un *mcc* de 0 sería el equivalente a lanzar una moneda al aire, un *mcc* cercano a -1 significa que el detector clasifica al revés y un *mcc* cercano a 1 significa que la calidad del detector es alta.

La métrica *tpr (true positive rate)* está en el intervalo [0, 1] y en este caso representa cómo de frecuente es que se marque como anómala una muestra anómala. Se buscan valores cercanos a 1.

La métrica *tnr (true negative rate)* está en el intervalo [0, 1] y en este caso representa cómo de frecuente es que se marque como normal una muestra normal. Se buscan valores cercanos a 1.

En el plan de experimentos $\mathcal{E}_7$ (Tabla 4) se han considerado datos de electricidad para 1000 contadores distintos, 28 detectores:

1. ARIMAX
2. ARIMA
3. NN
4. Min-Avg
5. JSD
6. FisherJenks
7. KLD
8. K-Means
9. MiniBatchK-Means
10. PCA-DBSCAN
11. IsolationForest
12. TEG_Cosine_30
13. TEG_Jaccard_30
14. TEG_Hamming_30
15. TEG_KL_30
16. TEG_Jeffreys
17. TEG_JS_30
18. TEG_Euclidean_30
19. TEG_Cityblock_30

20. TEG_Chebyshev_30
21. TEG_Minkowvski_30
22. TEG_Braycurtis_30
23. TEG_Kulczynski_30
24. TEG_Canberra_30
25. TEG_Bhattacharrya_30
26. TEG_Squared_30
27. TEG_Divergence_30
28. TEG_Additivesymetric_30

y 12 escenarios de consumo:

1. Normal
2. Swap
3. RSA_0.25_1.1
4. RSA_0.5_3
5. RSA_0.5_1.5
6. Avg
7. Min-Avg
8. FDI_0
9. FDI_5
10. FDI_10
11. FDI_20
12. FDI_30

Los resultados de las métricas de calidad han sido bastante buenos en general (Figura 25), sobre todo los de la red neuronal (NN), pues es el mejor detector utilizando cuatro de las cinco métricas comentadas al inicio del anexo. En la única métrica en la que no destaca es en la de *tnr*, pues este detector tiende a marcar comportamientos normales como anómalos. Otros detectores como el *TEG_Canberra_30* proporcionan también muy buenos resultados en todas las métricas de calidad.

En el plan de experimentos $\mathcal{E}_8$ (Tabla 4) se han considerado datos de gas para 1000 contadores distintos, 25 detectores (los mismos que para $\mathcal{E}_7$, excepto ARIMAX, ARIMA y PCA-DBSCAN) y 12 escenarios (los mismos que para $\mathcal{E}_7$). Tal y como se puede observar en la Figura 26, los resultados han sido peores que los del plan de experimentos $\mathcal{E}_7$ (Figura 25). Una de las razones puede ser que muchos de los registros para este conjunto de datos eran de 0 kWh (no se utiliza el gas todo el tiempo), por lo que las curvas de consumo son bastante distintas a las del conjunto de datos de electricidad. Sin embargo, los detectores que son buenos para detectar escenarios con el conjunto de datos de electricidad, como el detector basado en redes neuronales y algunas variantes del *TEG*, siguen funcionando relativamente bien con el conjunto de datos de gas.

detector	dataset	scenario	acc	mcc	bacc	tpr	tnr	tb (s.)	tp (s.)
ARIMAX	Electricity (ISSDA-CER)	All	0.133	-0.014	0.494	0.061	0.926	908.376	0.151
ARIMA	Electricity (ISSDA-CER)	All	0.11	-0.048	0.484	0.035	0.932	64.755	0.076
NN	Electricity (ISSDA-CER)	All	0.951	0.677	0.837	0.973	0.702	59.79	3.449
Min-Avg	Electricity (ISSDA-CER)	All	0.479	0.235	0.708	0.433	0.983	0.087	0.02
JSD	Electricity (ISSDA-CER)	All	0.65	0.281	0.75	0.63	0.87	0.008	0.002
FisherJenks	Electricity (ISSDA-CER)	All	0.23	0.08	0.552	0.165	0.94	0	0.045
KLD	Electricity (ISSDA-CER)	All	0.661	0.273	0.742	0.645	0.838	0.008	0.002
K-Means	Electricity (ISSDA-CER)	All	0.452	0.004	0.504	0.442	0.566	0.099	0.022
MiniBatchK-Means	Electricity (ISSDA-CER)	All	0.449	0	0.501	0.439	0.562	0.096	0.022
PCA-DBSCAN	Electricity (ISSDA-CER)	All	0.226	0.093	0.56	0.159	0.961	16.144	0.004
IsolationForest	Electricity (ISSDA-CER)	All	0.732	0.12	0.594	0.76	0.429	1.587	0.644
TEG_Cosine_30	Electricity (ISSDA-CER)	All	0.617	0.25	0.724	0.595	0.854	0.181	0.049
TEG_Jaccard_30	Electricity (ISSDA-CER)	All	0.626	0.291	0.762	0.599	0.924	0.179	0.049
TEG_Hamming_30	Electricity (ISSDA-CER)	All	0.657	0.324	0.789	0.63	0.948	0.175	0.048
TEG_KL_30	Electricity (ISSDA-CER)	All	0.795	0.118	0.582	0.838	0.325	0.181	0.049
TEG_Jeffreys_30	Electricity (ISSDA-CER)	All	0.752	0.374	0.814	0.74	0.887	0.181	0.049
TEG_JS_30	Electricity (ISSDA-CER)	All	0.751	0.351	0.793	0.742	0.844	0.182	0.05
TEG_Euclidean_30	Electricity (ISSDA-CER)	All	0.687	0.317	0.778	0.668	0.887	0.18	0.049
TEG_Cityblock_30	Electricity (ISSDA-CER)	All	0.717	0.324	0.778	0.705	0.852	0.179	0.049
TEG_Chebyshev_30	Electricity (ISSDA-CER)	All	0.665	0.305	0.77	0.644	0.897	0.184	0.05
TEG_Minkowski_30	Electricity (ISSDA-CER)	All	0.675	0.309	0.773	0.655	0.891	0.181	0.049
TEG_Braycurtis_30	Electricity (ISSDA-CER)	All	0.717	0.324	0.778	0.705	0.852	0.18	0.049
TEG_Kulczynski_30	Electricity (ISSDA-CER)	All	0.717	0.324	0.778	0.704	0.852	0.18	0.049
TEG_Canberra_30	Electricity (ISSDA-CER)	All	0.794	0.406	0.824	0.788	0.861	0.182	0.05
TEG_Bhattacharyya_30	Electricity (ISSDA-CER)	All	0.763	0.365	0.802	0.756	0.847	0.179	0.049
TEG_Squared_30	Electricity (ISSDA-CER)	All	0.74	0.341	0.788	0.731	0.844	0.18	0.049
TEG_Divergence_30	Electricity (ISSDA-CER)	All	0.763	0.386	0.821	0.752	0.89	0.18	0.049
TEG_Additivesymmetric_30	Electricity (ISSDA-CER)	All	0.45	0.151	0.634	0.413	0.854	0.18	0.049
All	Electricity (ISSDA-CER)	All	0.177	0.032	0.518	0.109	0.926	37.93	0.191

Figura 25: Resultados del plan de experimentos $\mathcal{E}_7$.

detector	dataset	scenario	acc	mcc	bacc	tpr	tnr	tb (s.)	tp (s.)
NN	Gas (ISSDA-CER)	All	0.924	0.509	0.758	0.957	0.558	54.263	3.49
Min-Avg	Gas (ISSDA-CER)	All	0.218	0.113	0.57	0.148	0.993	0.088	0.023
JSD	Gas (ISSDA-CER)	All	0.523	0.254	0.728	0.482	0.975	0.008	0.002
FisherJenks	Gas (ISSDA-CER)	All	0.234	0.08	0.554	0.17	0.937	0	0.046
KLD	Gas (ISSDA-CER)	All	0.476	0.225	0.7	0.432	0.968	0.008	0.002
K-Means	Gas (ISSDA-CER)	All	0.502	0.005	0.504	0.501	0.508	0.1	0.025
MiniBatchK-Means	Gas (ISSDA-CER)	All	0.471	-0.004	0.497	0.466	0.527	0.098	0.025
IsolationForest	Gas (ISSDA-CER)	All	0.758	0.105	0.578	0.794	0.363	1.599	0.677
TEG_Cosine_30	Gas (ISSDA-CER)	All	0.218	0.106	0.566	0.149	0.983	0.179	0.048
TEG_Jaccard_30	Gas (ISSDA-CER)	All	0.357	0.166	0.635	0.302	0.968	0.177	0.048
TEG_Hamming_30	Gas (ISSDA-CER)	All	0.43	0.209	0.68	0.38	0.981	0.173	0.047
TEG_KL_30	Gas (ISSDA-CER)	All	0.531	-0.067	0.44	0.55	0.33	0.179	0.048
TEG_Jeffreys_30	Gas (ISSDA-CER)	All	0.671	0.315	0.779	0.649	0.909	0.179	0.048
TEG_JS_30	Gas (ISSDA-CER)	All	0.342	0.16	0.628	0.285	0.97	0.18	0.048
TEG_Euclidean_30	Gas (ISSDA-CER)	All	0.313	0.153	0.616	0.252	0.981	0.178	0.048
TEG_Cityblock_30	Gas (ISSDA-CER)	All	0.193	0.097	0.555	0.121	0.989	0.177	0.048
TEG_Chebyshev_30	Gas (ISSDA-CER)	All	0.318	0.155	0.619	0.258	0.98	0.182	0.049
TEG_Minkowski_30	Gas (ISSDA-CER)	All	0.318	0.155	0.619	0.258	0.981	0.179	0.048
TEG_Braycurtis_30	Gas (ISSDA-CER)	All	0.193	0.097	0.555	0.121	0.989	0.178	0.048
TEG_Kulczynski_30	Gas (ISSDA-CER)	All	0.193	0.097	0.555	0.121	0.989	0.178	0.048
TEG_Canberra_30	Gas (ISSDA-CER)	All	0.662	0.32	0.784	0.638	0.931	0.18	0.048
TEG_Bhattacharyya_30	Gas (ISSDA-CER)	All	0.433	0.197	0.671	0.386	0.956	0.177	0.048
TEG_Squared_30	Gas (ISSDA-CER)	All	0.286	0.138	0.6	0.223	0.978	0.178	0.048
TEG_Divergence_30	Gas (ISSDA-CER)	All	0.628	0.303	0.773	0.6	0.946	0.178	0.048
TEG_Additivesymmetric_30	Gas (ISSDA-CER)	All	0.321	0.12	0.594	0.266	0.923	0.178	0.048
All	Gas (ISSDA-CER)	All	0.266	0.086	0.562	0.207	0.917	2.368	0.204

Figura 26: Resultados del plan de experimentos $\mathcal{E}_8$.