Desarrollo de algoritmos de aprendizaje automático para la anticipación de eventos en la Unidad Auxiliar de Potencia (APU) de la flota del A380

Álvaro Mesa Castellanos

Máster en Ingeniería de Telecomunicación

MÁSTERES
DE LA UAM
2018 - 2019

Escuela Politécnica Superior

UAM Universidad Autónoma de Madrid

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Desarrollo de algoritmos de aprendizaje automático para la anticipación de eventos en la Unidad Auxiliar de Potencia (APU) de la flota del A380

Máster Universitario en Ingeniería de Telecomunicación

Autor: MESA CASTELLANOS, Álvaro
Director: MARTÍNEZ SANCHO, Alberto
Co-director: TORRE TOLEDANO, Doroteo

Noviembre 2018

Desarrollo de algoritmos de aprendizaje automático para la anticipación de eventos en la Unidad Auxiliar de Potencia (APU) de la flota del A380

AUTOR: Álvaro Mesa Castellanos
Director: Alberto Martínez Sancho
Co-director: Doroteo Torre Toledano

Dpto. TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Noviembre de 2018

Agradecimientos

Me gustaría agradecer en primer lugar a Alberto y Carolina por haberme dado la oportunidad, y confiar en mí para realizar este proyecto, gracias al cual he aprendido a conocer cómo funciona y cómo se trabaja en un entorno profesional y en una empresa tan grande como Airbus. Agradecer a Alberto por su paciencia desde el primer día y disposición para resolver y explicarme cualquier cuestión o duda, ya que nunca había trabajado en el mundo aeronáutico y no sabía incluso lo que era o hacía un APU. También me gustaría dar las gracias a todos mis compañeros porque me han hecho sentir como uno más.

Por otra parte, agradecer el trato y disposición de Doroteo, siempre abierto a comentar lo que fuera y a ayudar, gracias.

Finalmente, agradecer a mis seres queridos por estar ahí siempre, animándome y apoyándome, sobre todo a mis padres.

Resumen

Este trabajo fin de máster se enmarca en un contexto empresarial, en el que las actividades se desarrollarán en el departamento *APU Design office* de *Airbus Operations S.L*, en la planta de Getafe, bajo una beca. El trabajo se llevará a cabo en una modalidad de codirección, en primer lugar, con un tutor empresarial, Alberto Martínez Sancho, conocedor del mundo aeronáutico y cuyas actividades se centran en el desarrollo de algoritmos de mantenimiento predictivo, y, en segundo lugar, con un tutor universitario, Doroteo Torre Toledano, investigador dedicado a tareas de reconocimiento de voz y procesamiento de audio.

El mantenimiento predictivo es un marco de actuación en el que elementos físicos se monitorizan con el fin de desarrollar modelos y algoritmos que puedan detectar fallos y síntomas de ellos antes de que éstos ocurran, realizando de esta forma las tareas de mantenimiento que procedan con el fin de evitar cualquier tipo de fallo que pueda ocasionar algún perjuicio o interrupción operacional. Gracias a la proliferación, desarrollo y madurez de plataformas que manejan una gran cantidad de datos, Airbus desarrolló la plataforma *Skywise*, en colaboración con la empresa americana *Palantir Technologies*, por ello, se buscaba en el departamento un perfil concreto de científico de datos el cual pudiera sacar partido de esa gran cantidad de información y desarrollar algoritmos para la anticipación de un evento de fallo en la unidad auxiliar de potencia, APU, de la flota del A380, el avión comercial de pasajeros más grande del mundo.

Para el desarrollo de un primer prototipo de sistema de anticipación de fallos, se estudiará en primer lugar el ente físico con el que trabajaremos, el APU, desde un punto mecánico y de señal. Posteriormente, se propondrá, basándonos en el estado del arte, una serie de técnicas, algoritmos, procedimientos y protocolo experimental para afrontar nuestro problema, además, se evaluarán distintos tipos de algoritmos de aprendizaje automático mostrando finalmente las métricas y curvas que permiten evaluar su rendimiento. Para concluir el trabajo, se presentarán los resultados al equipo técnico para el estudio de su rendimiento y viabilidad de cara a una posible implantación en el sistema de alertas de Airbus.

Palabras clave

Mantenimiento predictivo, *health monitoring*, *Skywise*, Airbus, IoT, aprendizaje automático, aviónica, APU, anticipación de eventos, detección de anomalías, tratamiento de series temporales, *imbalanced learning*.

Abstract

This Master´s thesis is linked to a business context, in which the different activities will be carried out in the APU Design office at Airbus Operations S.L, in Getafe plant, during an internship. The work will be co-directed firstly, by Alberto Martínez Sancho, business supervisor, who has a deep knowledge of aeronautical world and whose activities are focused on the development of predictive maintenance algorithms and health monitoring, secondly, by Doroteo Torre Toledano, academic supervisor, researcher and expert in audio processing and voice recognition.

Predictive maintenance is a framework in which physical elements are monitored in order to develop models and algorithms whose aim is to detect failures and its symptoms before they actually occur, carrying different tasks out so as to avoid any kind of failure that could damage the element and cause operational interruptions. Thanks to the proliferation, development and maturity of platforms that deal with a great amount of data, Airbus launched the Skywise platform, in partnership with Palantir Technologies, as a result, the APU Design Office was looking for a data scientist who could make value from that data in order to develop algorithms for a failure event anticipation in the auxiliary power unit, APU, of A380 fleet, the world´s biggest passenger commercial aircraft.

In order to develop a first prototype of a failure event anticipation system, the APU will be studied from a mechanical and signal point of view. Afterwards, based on the state of the art, different techniques, algorithms, procedures and experimental protocols will be proposed to face our problem, in addition, some machine learning algorithms will be evaluated, presenting the different metrics and curves that allow us to evaluate its performance. Finally, the results will be presented to the technical team for its performance and viability study, in pursuit of a possible implementation in Airbus alert system.

Keywords

Predictive maintenance, health monitoring, Skywise, Airbus, IoT, machine learning, aircraft, APU, event anticipation, anomaly detection, time series analysis, imbalanced learning.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

El siguiente trabajo de fin de máster se enmarca en un contexto empresarial, en el que las actividades se llevarán a cabo en el departamento '*Auxiliary Power Unit (APU) Design office'*, en la planta de Getafe de Airbus Operaciones S.L bajo una beca de un año de duración.

En los últimos años, debido principalmente al desarrollo de entornos y herramientas de análisis de datos, han surgido numerosas posibilidades en cuanto a la explotación de ellos, permitiendo mejorar el servicio ofrecido y mejoras en el desempeño de productos, concretamente, se dispone en la empresa de una plataforma de extracción, procesamiento y análisis de datos [1] y que de ahora en adelante nos referiremos a ella como *Skywise*. Por ello, se buscaba en el departamento un perfil muy concreto de científico de datos, el cual, mediante el desarrollo de algoritmos de aprendizaje automático, pudiera explorar este mundo, así como proponer nuevas aproximaciones y posibilidades de mejora de sus productos en el marco de actuación de *health monitoring* y mantenimiento predictivo, ámbito en el que mi director empresarial, Alberto Martínez, trabaja actualmente, desarrollando algoritmos para la detección de fallos en el APU. Como consecuencia de los resultados obtenidos con estas aproximaciones, el equipo veía interesante adentrarse en el desarrollo de algoritmos de aprendizaje máquina con el objetivo de resolver problemas reales existentes en la operación del APU, más concretamente en la del A380. En relación con esto último, existe un determinado fallo durante las maniobras en tierra que genera problemas operacionales, derivando en retrasos, costes y que será objeto de nuestro trabajo.

Por todo lo anterior, a través de algoritmos de *machine learning*, se pretende crear un prototipo que sea capaz de anticipar esos fallos, permitiendo a los agentes correspondientes realizar las tareas de mantenimiento que correspondan con el fin de mitigar el fallo con suficiente antelación.

Además, actualmente existe un equipo bastante maduro de científicos de datos en la planta de Toulouse, cuyas aproximaciones de aprendizaje máquina a problemas de anticipación de eventos han resultado bastante prometedoras, tal y como se puede ver en el siguiente trabajo [2], con lo que nos anima a seguir esta senda.

Con todo ello, la principal motivación para llevar a cabo este trabajo es la posibilidad de desarrollar y estudiar soluciones nunca antes vistas en el departamento, que puedan suponer mejoras en el desempeño de sus actividades, así como por el contenido propio del trabajo, colaborando de forma paralela con mi tutor empresarial y mi tutor universitario, Doroteo Torre Toledano, el cual posee grandes conocimientos en el ámbito de aprendizaje automático aplicado a voz y audio.

1.2 Objetivos

Dado el contexto empresarial en el que se enmarca este trabajo de fin de máster, los objetivos propuestos por el departamento para cumplir con la tarea encomendada en el transcurso de la beca son los siguientes:

1. Entender el problema a resolver, así como su adecuación y viabilidad marcado por el entorno empresarial en el que se desarrolla el trabajo.
2. Familiarización con la Unidad Auxiliar de Potencia (APU), entender las bases de su funcionamiento desde un punto de vista mecánico, así como de su sistema de sensores desde un punto de vista de señal, comprendiendo la adquisición y naturaleza de éstas.
3. Proponer, fundamentándose en el estado de arte, un marco de actuación que dé solución a un problema real al que se enfrenta el departamento.
4. Saber manejar la herramienta *Skywise*, basada en *Spark*™, entender el paradigma de la computación distribuida y ser capaz de manejar datos a gran escala.
5. Desarrollar un *pipeline* de extracción, pre-procesamiento de datos, entrenamiento y validación de modelos de aprendizaje automático lo más escalable posible, es decir, que pueda ser utilizado para abordar otros problemas similares existentes y que sea capaz de procesar una determinada cantidad de información, marcada por las necesidades del problema.
6. En última instancia, si el rendimiento es el esperado, integrar todo el trabajo realizado en forma de prototipo, el cual pueda ser utilizado por los operadores para prevenir el fallo y con ello realizar las tareas de mantenimiento predictivo que correspondan.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- *Estado del arte:* analizaremos y describiremos con detalle el APU, el evento que se pretende anticipar, los fundamentos matemáticos, metodología y algoritmos empleados, así como la herramienta *Spark*™.
- *Diseño:* describiremos la plataforma *Skywise*, así como la BBDD de eventos de fallo, la flota analizada y herramientas empleadas para la resolución del problema.
- *Desarrollo:* describiremos con detalle el procedimiento y experimentos que se han seguido para la concepción del sistema de aprendizaje automático.
- *Pruebas y resultados:* presentaremos los resultados de los experimentos, desglosándolos y analizándolos por algoritmos, proponiendo los mejores resultados obtenidos en este trabajo.
- *Conclusiones:* desarrollaremos las conclusiones de los experimentos y resultados obtenidos, además de proponer líneas de trabajo futuro.

2 Estado del arte

2.1 Auxiliary Power Unit (APU)

La unidad auxiliar de potencia, APU, es una turbo máquina [3] que se instala en la parte trasera del avión y cuyas funciones son las de proporcionar energía en diferentes formas para realizar funciones distintas a las de propulsión de éste, tarea de la que se encarga los motores principales. Con ello, los principales propósitos de esta máquina son:

1. Proporcionar energía eléctrica a los distintos componentes eléctricos y electrónicos del avión, bien cuando no estén en funcionamiento los motores principales o no exista una fuente de energía externa.
2. Generar presión neumática para el arranque de motores principales y el sistema de aire acondicionado en cabina.
3. En el caso de fallo de motores y pérdidas de potencia en vuelo, mantener los sistemas críticos funcionando, tales como luces, instrumentación y sistemas neumáticos.

Figura 2-1: En la figura de la izquierda, APU en su fase de instalación para realización de ensayos, en la de la derecha, montado en el avión final

Una vez presentadas las principales funciones del APU, nos podemos percatar de la importancia de este elemento en la operación del avión. Para evidenciar esto último, procederemos a continuación a describir un ejemplo general de funcionamiento, teniendo en cuenta que el APU se puede emplear, siempre que sea posible, en cualquier fase de operación del avión, tanto en vuelo como en tierra, sin embargo, el objeto de estudio de este trabajo se centrará en las operaciones en tierra, como ya presentamos en la introducción de este trabajo.

En la operación de una aeronave comercial se dan distintas fases, en la primera de ellas, el avión se encuentra en fase de *parking*, esperando en la puerta correspondiente al embarque de los pasajeros, realizando al mismo tiempo acciones de repostaje, comprobaciones y carga de equipaje. Durante esta fase, si se requiere energía neumática o eléctrica, por ejemplo, para mantener el aire acondicionado de cabina mientras embarcan los pasajeros o mantener ciertos instrumentos en funcionamiento, se puede facilitar mediante el APU, sin embargo, debido a las restricciones en ruido, emisiones y teniendo en cuenta de que se trata de una máquina con un gran consumo de combustible, se opta generalmente, y si se dispone de ello, por una toma eléctrica externa en tierra. Posteriormente, se da la fase de *taxi-out*, en la que el avión se dirige a la pista para el despegue, fase en la que entra en escena nuestro evento de fallo. Como ya explicamos anteriormente, un uso importante del APU es la del arranque de motores principales, los cuales se arrancan con presión neumática que genera la máquina.

Con ello, mientras el avión se dirige a la pista, el piloto procede al arranque de los motores y si todo ocurre como es debido se arrancan satisfactoriamente, si no, el APU se apaga y no logra arrancarlos, éste será de ahora en adelante nuestro evento de fallo o fallo simplemente, el cual se pretende anticipar. Como consecuencia de ello, se puede o bien volver a arrancar el APU e intentar de nuevo la secuencia de arranque de motores, o solicitar una fuente externa de energía para conseguirlo, todo ello deriva en importantes retrasos, costes y el descontento del pasaje.

Debido a cuestiones de confidencialidad y propiedad industrial no podemos dar una explicación detallada de por qué y cómo se produce este fallo exactamente, por lo que de ahora en adelante lo tomaremos como un fallo que sería deseable predecir con antelación para que los agentes correspondientes recibieran una alerta, y así poder realizar las tareas de mantenimiento correspondientes y evitar todas las consecuencias expuestas anteriormente.

Para finalizar este apartado, describiremos un elemento importante en la operación del APU, la *Electronic Control Box* o ECB. La ECB es un computador a bordo que integra toda la lógica de control de la máquina y realiza la monitorización y adquisición de las señales procedentes de los distintos sensores de ésta a través de un canal dedicado que provee una cierta QoS, conecta distintos componentes del avión y cuyo nombre es AFDX *(Avionics Full-Duplex Switched Ethernet),* propiedad de Airbus. En cualquier caso, no es objetivo de este trabajo la descripción y estudio detallado del tipo de red y protocolo, ya que en nuestra aplicación final nos abstraemos de todo ello y accedemos a los datos disponibles a través de la plataforma *Skywise*, sin embargo, si el lector tiene interés en conocer más detalles, encontrará una referencia bibliográfica [4] al final de este trabajo.

Por último, la ECB toma un papel esencial en el control de la máquina en el caso de que se den situaciones de potencial riesgo, por ejemplo, valores fuera de lo normal de algún sensor, resultando en un *shutdown* o apagado, cuyo objetivo es el de proteger el APU. En relación con esto último, nuestro evento de fallo se da precisamente por un apagado comandado por la ECB.

2.2 Sistema de aprendizaje automático

Una vez explicado el funcionamiento y características del APU, así como el evento a anticipar, desarrollaremos en las siguientes secciones y subsecciones las distintas herramientas, metodologías, formulación y algoritmos propuestos para la resolución del problema.

A continuación, mostramos un esquema simplificado de sistema de aprendizaje automático, el cual desarrollaremos con más detalle en el apartado de desarrollo:

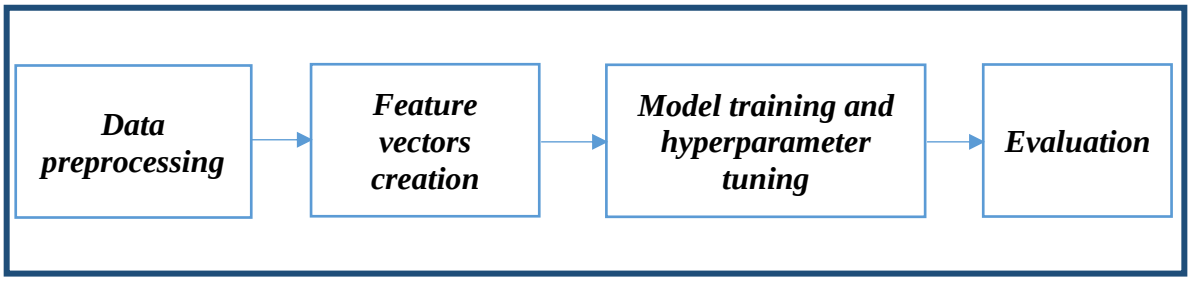

Figura 2-2: Esquema genérico de sistema de aprendizaje automático

Con ello, nuestro esquema básico integra cuatro bloques funcionales bien diferenciados entre sí, los cuales tienen las siguientes características:

1. *Data preprocessing:* los datos procedentes de fenómenos de la vida real pueden no estar en la forma adecuada para el resto de bloques de procesamiento, en nuestro caso, trabajamos con series temporales multivariadas procedentes de sensores del APU, es por ello que se hace necesario un procesamiento previo. Teniendo en cuenta que los datos se toman con una frecuencia de muestreo dada, *fs*, se puede dar el caso en el que no dispongamos de lecturas de uno o varios sensores en un instante de tiempo dado, por lo que o bien podemos descartar esas muestras o realizar algún tipo de interpolación. Por otra parte, debido a la heterogeneidad de los rangos de valores de los distintos sensores, por ejemplo, presión, temperatura, señal binaria de control, es necesario realizar una normalización previa, de media y varianza en nuestro caso, para que los rangos sean comparables entre sí. Además, se facilita significativamente el entrenamiento de los algoritmos que emplean descenso por gradiente y se mejora el rendimiento en general de cualquier algoritmo. Por último, hasta ahora hemos tenido en cuenta variables continuas, sin embargo, nos podemos encontrar con variables categóricas, como por ejemplo, modelo de APU o aeropuerto de destino, por ello, es necesario cuantificar numéricamente de alguna forma estas características, dado que conviene unificar la naturaleza de los datos de entrada para permitir una mayor flexibilidad en los algoritmos utilizados. Con ello, en la literatura se proponen diferentes aproximaciones, siendo una de las más utilizadas *one-hot encoding*, la cual asocia un número binario único a un valor concreto de la característica, tal y como vemos en la tabla a continuación,

Airport	O-H encoded airport	Engine manufacturer	O-H encoded em
MAD	*0001*	***PWC***	*01*
DBX	*0010*	***HW***	*10*
CDG	*0100*	***HW***	*10*
LHR	*1000*	***PWC***	*01*

Tabla 2-1: Ejemplo de codificación de variables categóricas siguiendo el esquema *one-hot*

2. *Feature vectors creation:* en esta fase, se toman los datos ya en la forma adecuada del bloque anterior y se conforman lo que llamamos vectores de características, que servirán como entrada al entrenamiento del modelo. En esta fase, se pueden optar por técnicas de reducción de dimensionalidad, reducción de atributos o adición de nuevas características a partir de las originales, por ejemplo, la derivada de una señal o correlación de señales. Además, hay que tener en cuenta el tipo de paradigma de aprendizaje que estemos empleando, supervisado (disponemos de etiquetas para esos vectores), no supervisado (no disponemos de etiquetas), por lo que, para el primer caso, habría que añadir la etiqueta a cada vector según corresponda, en nuestro problema, por ejemplo, fallo o no fallo, ya que disponemos de las etiquetas.
3. *Entrenamiento de modelos y ajuste de hiperparámetros:* una vez disponemos de los vectores de características, procedemos a entrenar el modelo de *machine learning* escogido. Con ello, debemos decidir primero el modelo que mejor se puede adecuar al problema, qué conjunto de entrenamiento, validación y test empleamos, las métricas de evaluación a optimizar y valores de parámetros del modelo a probar. Como resultado, obtenemos un modelo con un conjunto de parámetros optimizado según la métrica escogida.

4. *Evaluation:* en esta última fase, se parte del modelo entrenado en la etapa anterior y se evalúa frente a un conjunto de datos de prueba, con el objetivo de ver como se comportaría ante datos que no han sido utilizados previamente. Con ello, se calculan las diferentes métricas necesarias para evaluarlo y se toman decisiones.

2.2.1 Series temporales, análisis y pre-procesamiento

Una vez desarrollados los diferentes bloques que integran un sistema genérico de aprendizaje automático, pasaremos a continuación al estudio y caracterización de las señales presentes en el problema. Como ya anticipamos anteriormente, en nuestro problema nos encontramos con series temporales multivariadas, procedentes de un sistema que arroja muestras con una frecuencia dada, *fs*, en cada instante *t*. Con ello, si tenemos lecturas de *d* sensores y registros a partir de un origen de tiempos T_1 y final T_2, podemos expresar matemáticamente la serie temporal multivariada de la siguiente forma:

$$X_t \in \mathbb{R}^d, \qquad T_1 < t < T_2$$

De esta manera, en cada instante *t*, disponemos de un vector de dimensión *d*. Volviendo a la implementación de un sistema de aprendizaje automático, la selección del número de sensores empleados, así como el rango temporal seleccionado para el entrenamiento y prueba de los modelos, depende de la naturaleza del problema al que nos estemos enfrentando, por lo que en la sección de desarrollo analizaremos de forma más profunda todo ello, justificando el porqué de una elección u otra.

2.2.1.1 Normalización de media y varianza

Como ya adelantamos anteriormente, debido a que disponemos de un conjunto heterogéneo de características en cuanto a su margen de valores se refiere, es conveniente realizar una normalización para transformar el espacio de características y que los datos sean comparables unos con otros. Esta normalización, ampliamente empleada en el estado del arte, consiste en sustraer la media y dividir por la varianza de cada característica, resultando en características de media 0 y varianza unidad. Con ello conseguimos que la convergencia sea óptima y más rápida durante el entrenamiento en modelos que emplean descenso por gradiente y en general logramos que el tiempo de entrenamiento sea menor y se mejore el rendimiento de los algoritmos. Por último, cabe destacar, que esta transformación asume que los datos siguen una distribución gaussiana.

Sea una matriz de datos *D*, que recoge el registro de *d* sensores determinados hasta el momento *T*, $D = \langle X_t \rangle_{t=1}^{T}$ de dimensión *T*×*d*, en la que cada columna representa los valores de un sensor en concreto y las filas un instante de muestreo, la columna *j* de la nueva matriz normalizada $\overline{D}$ resultaría:

$$\overline{D_{:,J}} = \frac{D_{:,j} - mean(D_{:,j})}{std(D_{:,j})},$$

donde *:* se refiere a todos los valores de la columna, *mean* a la media muestral y *std* a la desviación típica muestral.

2.2.1.2 Principal Component Analysis, PCA

En este apartado nos centraremos en el estudio de la técnica de reducción de dimensionalidad empleada ampliamente en el estado del arte, PCA. En numerosos artículos científicos [5] [6] se ha estudiado el impacto negativo que tiene en el desempeño de los modelos el emplear vectores de características de dimensión muy alta, conocido como *curse of dimensionality*, con lo que es conveniente recurrir a técnicas de reducción de dimensionalidad. Concretamente, en nuestro problema trataremos con vectores de muy alta dimensión, como podremos ver en el desarrollo del trabajo, por lo que emplearemos esta técnica para intentar paliar el fenómeno antes nombrado.

Esta técnica analiza un *set* de datos en el que las variables están correladas de alguna manera, suponiendo una distribución gaussiana de ellas, con ello, el objetivo es el de extraer la información más importante con el fin de representarla en un espacio de características distinto y reducido al original, con variables ortogonales entre sí llamadas componentes principales [4]. Matemáticamente se basa en la descomposición en valores singulares de la matriz de covarianza de las características involucradas, tal y como veremos a continuación.

Sean dos variables aleatorias *X* e *Y*, su covarianza viene dada por la siguiente expresión:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{(n-1)},$$

la matriz de covarianza de *n* variables aleatorias quedaría expresada como:

$$C = \begin{pmatrix} cov(X_1,X_1) & \cdots & cov(X_1,X_n) \\ \vdots & \ddots & \vdots \\ cov(X_n,X_1) & \cdots & cov(X_n,X_n) \end{pmatrix},$$

con ello, el problema se reduce a descomponer en autovectores la anterior matriz. Sean λ_n los autovalores asociados a los autovectores p_n, la descomposición de la matriz *C* queda de esta forma:

$$Cp_n = \lambda_n p_n,$$

donde p_n representa la nueva base ortonormal del espacio transformado, y λ_n indica la contribución de cada componente. Como disponemos de la importancia λ_n de cada componente, la reducción de dimensionalidad se basa entonces en tomar las *l* componentes con λ mayor, siendo $l<n$, y proyectar los datos en el nuevo espacio transformado.

Por último, debemos tener en cuenta que PCA asume que los datos son combinación lineal de cierta base y como ya dijimos anteriormente, supone una distribución de datos gaussiana, por lo que este método podría no ser aplicable dependiendo del tipo y naturaleza de datos de nuestro problema.

2.2.1.3 Visualización de datos de alta dimensión, t-SNE

En el apartado anterior hemos analizado una técnica para la reducción de dimensionalidad de *datasets* con un gran número de dimensiones, el cual transforma el *dataset* original para ser empleado por los siguientes bloques de la arquitectura de nuestro sistema. Sin embargo, como ya comentamos, esta técnica asume relaciones lineales entre las variables y distribución de datos gaussiana, con lo que puede no ser del todo óptima si nuestra intención es ver las relaciones no lineales existentes entre ellas con el fin de visualizarlas en un espacio

de menor dimensión, por ejemplo, en un gráfico de dispersión para comprender la estructura de los datos. Para este efecto, existe una técnica ampliamente utilizada en la comunidad científica para visualizar *datasets* de gran dimensión cuyo nombre es *t-Stochastic Neighbor Embedding o t-SNE* [7]. El propósito principal de esta técnica es representar en un espacio de dimensión reducido la estructura del *dataset* original, intentándola preservar lo máximo posible. Concretamente, en nuestro caso, buscamos encontrar cierta separabilidad entre datos de fallo y datos de funcionamiento correcto y con ello, ver cierta estructura en ellos. A continuación, desarrollaremos matemáticamente algunos puntos importantes para entender este algoritmo.

Stochastic Neighbor Embedding (SNE), tiene como objetivo convertir las distancias euclídeas de alta dimensión entre un par de puntos en probabilidades condicionales que representan similitudes. La similitud se define como la probabilidad de que al tomar un punto x_i, se tome el punto x_j como su vecino, teniendo en cuenta una distribución gaussiana centrada en x_i y varianza σ_i, al contrario que *t-SNE*, el cual emplea una distribución *t-student* para evaluar las probabilidades en lugar de la distribución gaussiana. La similitud se expresa matemáticamente como:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2{\sigma_i}^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2{\sigma_i}^2)}$$

paralelamente, la similitud entre dos puntos y_i e y_j la podemos definir en el espacio de dimensión reducida como:

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq i}\exp(-\|y_i - y_k\|^2)}$$

Con ello, volviendo a la idea principal del algoritmo, el cual busca preservar la estructura del *dataset* original en la visualización en un espacio reducido, el objetivo es entonces conseguir que las dos probabilidades anteriormente mencionadas sean lo más parecidas entre sí. Con ello, para minimizar el desajuste entre estas dos, se recurre a una optimización por descenso del gradiente en la que la función de coste C y expresión a minimizar vienen dadas por las siguientes fórmulas:

$$C = \sum_I KL(P_i||Q_i) = \sum_i\sum_j p_{j|i} log\frac{p_{j|i}}{q_{j|i}}$$

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

donde la función *KL* es la divergencia de *Kullback-Leibler* (similar a la medida de entropía cruzada). Finalmente, obtenemos la representación en menor dimensión de los datos Υ de manera iterativa, como:

$$\Upsilon^{(t)} = \Upsilon^{(t-1)} + \eta\frac{\delta C}{\delta \Upsilon} + \alpha(t)(\Upsilon^{(t-1)} - \Upsilon^{(t-2)}),$$

siendo t la iteración actual, por lo que hay que fijar un número de iteraciones en el algoritmo, η la tasa de aprendizaje y $\alpha(t)$ la perplejidad, que actúa como medida efectiva del número de vecinos, en [7] se recomienda emplear entre 5 y 50.

2.2.2 Entrenamiento de modelos: Random-search y validación cruzada

El objetivo de un modelo de aprendizaje automático A es encontrar una función f que minimice cierta pérdida esperada $\mathcal{L}(x, f)$ [8]. De forma general, se parte de un conjunto de parámetros θ, con la finalidad de encontrar el subconjunto óptimo $\hat{\lambda}$, entendiendo como óptimo los parámetros que minimizan cierta métrica durante el entrenamiento y que generalicen lo mejor posible frente a datos desconocidos. Esta problemática de elegir el conjunto óptimo de parámetros para un modelo dado se conoce en la literatura como ajuste de hiperparámetros o *hyperparameter tuning*. Existen numerosas aproximaciones y propuestas en el estado del arte para encontrar los parámetros óptimos de un modelo, en nuestro trabajo emplearemos la conocida como *random-search* [8].

Con el problema ya descrito, podemos pensar en primer lugar y de forma directa en encontrar el subconjunto de parámetros óptimos probando combinaciones de θ y tomando aquella que minimiza cierta métrica durante el entrenamiento, esta técnica se conoce como *grid-search*. Sin embargo, esta aproximación puede no ser óptima si tenemos un gran número de parámetros a probar, lo que añade gran coste computacional ya que el número total de combinaciones es de $S = \prod_{k=1}^{K} |L^{(k)}|$, siendo K el número de parámetros y L el vector de valores a probar del parámetro k correspondiente. Por otro lado, esta técnica sufre de la llamada *curse of dimensionality*, con lo que puede no obtener buenos resultados en espacios de alta dimensión, caso en el que nos encontramos en nuestro problema.

Motivados por las razones expuestas anteriormente, nos decantamos por emplear *random-search*. Este método se basa en el muestreo del espacio de parámetros que nosotros designamos, suponiendo una distribución uniforme de éste para parámetros numéricos. Con ello, su objetivo es buscar de manera más inteligente el mejor conjunto de parámetros en regiones 'prometedoras', tal y como se ilustra en la siguiente figura, donde se puede ver que al emplear *grid-search* podemos estar obviando regiones que no contempla nuestro *grid* de parámetros:

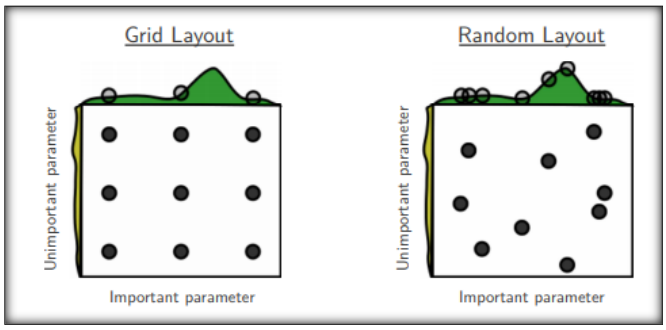

Figura 2-3: Comparativa entre *grid-search* y *random-search*, extraída de [8]

Con ello, *random-search* es una técnica iterativa, en la que se busca las zonas donde, de acuerdo con la métrica establecida para el entrenamiento, ésta se maximice o minimice, dependiendo de su naturaleza, y se centra en un espacio alrededor de ellas para encontrar el mejor conjunto de parámetros. En términos computacionales, este método reduce drásticamente la carga de procesamiento, ya que establecemos un número máximo de iteraciones a priori respecto a *grid-search*, además, según [8], se necesita en media un número menor de iteraciones para obtener resultados similares a *grid-search*. Por otra parte, *random-search* no ve reducido su rendimiento en espacios de alta dimensión, por lo que no le afecta demasiado la llamada *curse of dimensionality*.

Por último, y no por ello menos importante, estudiaremos la técnica llamada *k fold cross-validation*, fundamentando su importancia en el entrenamiento de modelos. Como comentamos anteriormente, el objetivo general de un modelo de aprendizaje automático es el de minimizar una función dada, la cual depende del problema y tipo de modelo que estemos empleando. Con ello, se parte de un conjunto de entrenamiento que sirve como entrada para el aprendizaje, más adelante, cuando procedemos a evaluar el rendimiento del modelo, lo hacemos con datos no empleados antes, llamado conjunto de prueba, calculamos las métricas de evaluación determinadas y extraemos conclusiones. Este rendimiento va a medir en cierta manera la capacidad que tiene el modelo de generalizar frente a datos nuevos no empleados en el entrenamiento, sin embargo, al emplear un conjunto de datos de entrenamiento fijo podemos no estar generalizando adecuadamente y no obtener los resultados esperados. Por ello, es importante emplear un conjunto de validación durante la fase de entrenamiento, el cual servirá a modo de prueba para evaluar la calidad del entrenamiento del modelo, con ello evitaremos el sobreajuste u *overfitting*.

Ligado a la idea del empleo de un conjunto de validación, surge la técnica introducida anteriormente la cual pretende generar modelos más robustos frente a futuros datos de prueba, esta técnica se conoce como *k-fold cross validation*, siendo *k* un número entero a fijar previamente, el cual indica el número de particiones iguales del conjunto de entrenamiento. Partiendo de un conjunto de entrenamiento y prueba dados, que representan una proporción del *dataset* original y un número *k* determinado, esta técnica funciona de la siguiente manera:

1. Se realizan *k* particiones del conjunto de entrenamiento y se dejan *k-1* para entrenamiento y una partición para la validación del modelo donde la proporción de datos empleado es de *(k-1)/k* y *1/k* respectivamente, con ello, se producen *k* iteraciones o entrenamientos de un modelo determinado, en las que se calculan *k* métricas a partir del conjunto de validación.
2. En función de la métrica escogida para evaluar el modelo, se escoge aquel modelo cuyo rendimiento sea mejor, por ejemplo, la mayor precisión en una clasificación.

Sin embargo, en relación con el último punto, la precisión puede no ser la métrica más adecuada, ya que ésta depende de la naturaleza de nuestro problema y de la manera en la que vamos a evaluar nuestro sistema final, es por ello por lo que existen diferentes métricas las cuales explicaremos en el siguiente apartado.

2.2.3 Medidas de rendimiento

Para evaluar el rendimiento de un modelo de aprendizaje automático existen diversas métricas que nos permiten valorar si es adecuado y cómo se comporta en la tarea encomendada, por ello, a continuación, pasaremos a describir aquellas que emplearemos en nuestro trabajo.

Concretamente, como ya hemos anticipado en el apartado anterior, nos enfrentamos a un problema de clasificación de dos clases, fallo o éxito, en el que la salida de nuestro sistema es la probabilidad de que dado un vector de entrada este pertenezca a una clase u otra. Con ello, la conocida como matriz de confusión, nos ayuda a evaluar en primera instancia cómo se está comportando el modelo en la clasificación, tal y como podemos ver en la siguiente tabla, donde *0* indica éxito y *1* fallo:

		Predicted labels	
		0	**1**
Ground truth	**0**	*# True Negatives: TN*	*# False Positives: FP*
	1	*# False Negatives: FN*	*# True Positives: TP*

Tabla 2-2: Descripción de la matriz de confusión para una clasificación de dos clases

Las filas indican la etiqueta real de las muestras y las columnas la etiqueta predicha, por ello, en una clasificación perfecta sin errores, la diagonal principal estaría formada por el número de éxitos y fallos reales respectivamente, siendo cero el resto de posiciones de la matriz. Sin embargo, en un sistema real se dan errores en la predicción, debido a ello, recurrimos a las siguientes métricas que parten de los datos de la matriz de confusión para evaluar el rendimiento:

1. *Precisión o precision:* se calcula como $TP/(TP+FP)$, expresa la habilidad del clasificador de no predecir como positiva una muestra que es en realidad negativa.
2. *Exhaustividad o recall:* se calcula como $TP/(TP+FN)$, indica la habilidad del clasificador de encontrar todas las muestras positivas.
3. *Puntuación F o f-score:* esta medida combina las otras dos anteriores y se puede interpretar como una media ponderada de la precisión y exhaustividad. Se calcula como $F_\beta = (1+\beta^2) * (Precision * Recall)/((\beta^2 * Precision) + Recall)$, en la que el factor β expresa la importancia de la exhaustividad respecto a la precisión. Esta medida ponderada puede ser interesante si por ejemplo nos importa en mayor medida la precisión de nuestro sistema frente a su capacidad de encontrar las muestras positivas, esto es, queremos pocos falsos positivos y nos importa en menor medida la cantidad de verdaderos positivos que clasifiquemos. Si particularizamos para $\beta = 1$, f_1*-score*, estaríamos dando la misma importancia a ambas medidas.

Basándonos en las métricas anteriores, podemos representar de forma gráfica el rendimiento de un clasificador mediante la curva ROC *(Receiver Operating Characteristic)* y la curva de precisión y exhaustividad de la clase positiva, los siguientes ejemplos son resultados obtenidos a partir de un mismo clasificador para nuestro problema:

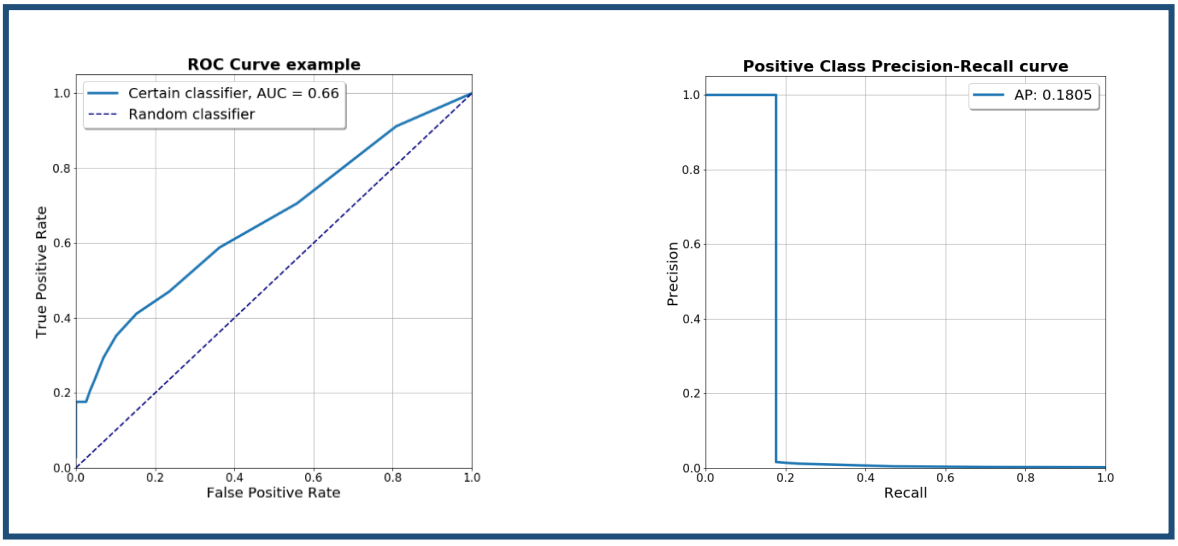

Figura 2-4: Ejemplo de resultados de una clasificación concreta en nuestro problema: curva ROC y curva de *precision-recall* de la clase minoritaria

En primer lugar, la curva ROC nos indica la habilidad del clasificador para distinguir entre verdaderos y falsos positivos, donde la métrica AUC *(Area Under Curve)* refleja el área que engloba la curva por debajo de ella, siendo el máximo valor 1, correspondiente a una clasificación perfecta, y 0 el mínimo, lo cual indicaría que nada ha sido clasificado correctamente. La salida de un clasificador representa la probabilidad, puntuación o *score*, entre 0 y 1, de que la muestra de entrada pertenezca a una clase u otra, teniendo esto en cuenta, la curva se construye variando el umbral de probabilidad en la decisión para la clasificación final, por lo que para cada uno tendríamos un par de puntos correspondientes al número de verdaderos y falsos positivos. Por otro lado, la recta representada con guiones indica el rendimiento de un clasificador aleatorio, el cual predeciría una clase u otra con la misma probabilidad, por ello, supone un límite adecuado para evaluar el rendimiento a primera vista de nuestro clasificador, es decir, las curvas por encima de esa recta clasificarían mejor que un clasificador aleatorio.

En segundo lugar, la curva de precisión y exhaustividad se construye de igual manera a la curva ROC, sin embargo, en ésta representamos para cada valor de umbral, la precisión y la exhaustividad para la clase positiva, donde la métrica AP, *Average Precision*, calculada como $AP = \sum_n (R_n - R_{n-1}) P_n$, es una media ponderada de las precisiones por el incremento de *recall* en cada umbral, ϑ. A lo largo del trabajo, pondremos especial énfasis en el análisis de esta curva, ya que nos indica cómo discriminamos entre verdaderos y falsos positivos y además, de entre los verdaderos positivos reales cuántos es capaz el modelo de clasificar, lo cual será de especial importancia en nuestro problema, ya que nos interesará evaluar el rendimiento del clasificador en esta clase, fallos, en la que lo ideal sería no equivocarnos a la hora de clasificar uno y ser capaz de predecir todos ellos. Esto último es importante ya que el coste de no predecir todos los fallos correctamente es alto, debido a las consecuencias del problema que desarrollamos en el apartado de descripción del APU. Siguiendo con esta idea, se puede dar el caso en el que tengamos una AUC muy alta cercana a 1 y una AP muy baja, comportamiento típico con un conjunto de test muy desbalanceado, por lo tanto no estaríamos prediciendo correctamente los fallos existentes, aunque la precisión sea muy alta, por consiguiente, es importante tener en cuenta ambas curvas en nuestro problema.

Finalmente, en conexión con el siguiente apartado, podemos adelantar que nos encontraremos ante un problema de desbalanceo de clases, en el que la proporción de la clase de éxito respecto a la de fallo es alta, con ello, esperamos tener una precisión media de la clase minoritaria baja y un comportamiento de ambas gráficas similar a los ejemplos vistos anteriormente, característico en problemas con clases desbalanceadas.

2.2.4 Imbalanced learning

En el apartado anterior hemos introducido un fenómeno al que nos enfrentaremos en nuestro problema, el llamado desbalanceo de clases o *imbalanced learning,* en el que la proporción en la que se dan las distintas clases son bastante dispares, de uno o varios órdenes de magnitud mayor una que otra, esto conlleva diferentes consecuencias que estudiaremos a continuación.

En nuestro problema tratamos con una turbo máquina cuya probabilidad de fallo se espera a priori baja, ofreciendo un alto porcentaje de disponibilidad durante su vida útil, ya que hay que tener en cuenta que en el mundo aeronáutico los retrasos y fallos operacionales se cuantifican con una gran cantidad de dinero. Sin embargo, cualquier ente físico es susceptible de experimentar fallos, ya sea por la degradación según el paso del tiempo o diseño de la propia máquina, en nuestro caso concreto, la frecuencia de fallo es

significativamente más baja a la de éxito, pero cuando la máquina falla conlleva problemas significativos, los cuales sería deseable anticipar con suficiente antelación, es por ello que a la hora de diseñar un sistema de aprendizaje automático que pretenda anticipar estos fallos haya que tener en cuenta el desbalanceo de clases. Con ello, este fenómeno afecta directamente al rendimiento de un algoritmo en lo que a la clase minoritaria se refiere, ya que éste carece de suficientes muestras de entrenamiento de ella y en consecuencia tendremos un buen desempeño en la clasificación de las muestras mayoritarias, pero uno muy deficiente en las minoritarias, este fenómeno se conoce en la literatura como *curse of imbalanced datasets*. Como primera aproximación para intentar paliar estas consecuencias se han propuesto las siguientes técnicas en el estado del arte:

- *Submuestreo o undersampling aleatorio de la clase mayoritaria*: con ello conseguimos equiparar el número de muestras de cada clase, submuestreando de manera aleatoria la clase mayoritaria, sin embargo, podemos perder información importante sobre los distintos patrones existentes en la clase submuestreada.
- *Añadir pesos para cada clase*: partiendo del número original de muestras para cada clase, podemos indicar en el entrenamiento del algoritmo un peso determinado para que pondere de distinta forma cada dato de una clase u otra, mayor en la minoritaria y menor en la minoritaria.
- *Sobremuestreo u oversampling de la clase minoritaria:* consiste en repetir un número determinado de veces los datos de la clase minoritaria para que su presencia sea equivalente en el entrenamiento, aun así, no estamos añadiendo información que pueda ser determinante en cuanto a la separabilidad de las clases se refiere.

A continuación, pasaremos a describir y analizar otras técnicas de *oversampling* y *undersampling* que pretenden mejorar los resultados obtenidos por los clasificadores respecto a las aproximaciones anteriores.

2.2.4.1 Synthetic Minority Over-sampling Technique: SMOTE

Como ya hemos explicado antes, el desbalanceo de clases se da en numerosos problemas de la vida real, como por ejemplo en detección de llamadas fraudulentas, detección de derrames de petróleo en imágenes satelitales o detección de fraude, donde nos podemos encontrar un desbalanceo de 100 frente a 1 [9]. Motivado por ésto, existe una técnica de *oversampling* o sobremuestreo de la clase minoritaria llamada SMOTE [9] [10], la cual pretende reducir los efectos de no disponer de suficientes muestras de la clase menos representada, por ello, consideramos conveniente su uso en nuestro problema, ya que como veremos en la sección de desarrollo, existe un gran desbalanceo entre clases de fallo y éxito.

Para crear las nuevas muestras sintéticas, el algoritmo parte de la base de que ellas se deben encontrar en el entorno de la muestra real con sus *k* vecinos más próximos, con ello, sus fases son las siguientes:

1. La entrada al algoritmo son tres variables, los *k* vecinos que vamos a considerar, en [9] emplean cinco, el número de muestras *N* sintéticas que queremos generar y el número de muestras de la clase minoritaria *T*.
2. Se calculan los *k* vecinos más próximos de cada muestra minoritaria x_i.
3. Se comienza con la muestra x_1 y por cada una se elige un vecino al azar, la nueva muestra sintética se calcula como la diferencia de cada muestra y su vecino multiplicada por una perturbación, un número aleatorio entre 0 y 1. Si se llega a la

muestra x_T sin haber conseguido las N muestras requeridas se vuelve a empezar por la primera.

Para tener una representación visual de cómo se generan las muestras, mostramos en esta figura una muestra x_i y sus cinco vecinos más próximos con sus nuevas muestras generadas [$x_{i1o;\ldots;}\, x_{i5o}$]:

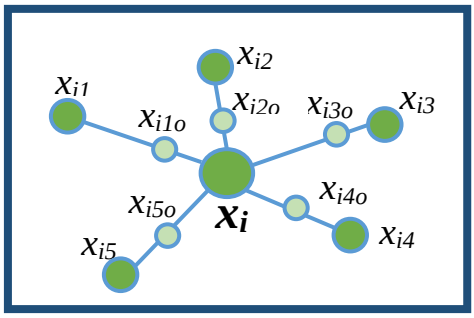

Figura 2-5: Ejemplo de *oversampling* de una muestra genérica x_i con el algoritmo SMOTE

2.2.4.2 Combinación de sobremuestreo y submuestreo: SMOTE-ENN

En el apartado anterior hemos analizado SMOTE como algoritmo de sobremuestreo de la clase minoritaria. Paralelamente, existen técnicas de submuestreo de la clase mayoritaria, las cuales pueden ayudar sustancialmente [11] al rendimiento de los algoritmos, en este trabajo emplearemos *Edited Nearest Neighbour o* ENN.

El algoritmo EEN se basa, como ya lo hacía SMOTE, en criterios y reglas de vecindad para eliminar las muestras de la clase mayoritaria, con lo que aquellas que no cumplen ciertos criterios y que por lo tanto no son susceptibles de pertenecer a cierto vecindario, se eliminan. Con ello, las fases del algoritmo son las siguientes:

1. Se establece y se calcula el número de vecinos *k* que se van a tener en cuenta para cada muestra de la clase minoritaria.
2. Teniendo en cuenta cada muestra y sus *k* vecinos más próximos, se mantiene y no se elimina en función del criterio de vecindad escogido. En el primero, todos los vecinos deben pertenecer a la clase mayoritaria, en el segundo, la mayoría.

Finalmente, motivados por los resultados obtenidos en [11] y para aunar las bondades de ambas aproximaciones, emplearemos en nuestro trabajo una combinación de sobremuestreo y submuestreo, en el que primero se aplica el algoritmo SMOTE y después ENN.

2.3 Mantenimiento predictivo y formulación del problema

El mantenimiento predictivo o *health monitoring* es un marco de actuación en el que un elemento o elementos de un sistema físico que dan cierto servicio se monitorizan para diagnosticar posibles problemas y fallos de rendimiento lo antes posible con el fin de ofrecer una alta disponibilidad. En el apartado de descripción de APU desarrollamos e introdujimos el problema al que nos vamos a enfrentar en nuestro trabajo, siendo el principal objetivo el desarrollo de algoritmos para intentar anticipar el evento de fallo y así generar una alerta, gracias a la cual operadores y agentes determinados puedan realizar las tareas de mantenimiento predictivo correspondientes con el fin de garantizar la continuidad del servicio ofrecido. El evento de fallo provoca retrasos, gran impacto económico y el

descontento de los clientes, por lo que una vez más, nos gustaría recalcar su importancia para tenerla en cuenta a lo largo del trabajo.

Un avión es un ente físico el cual dispone de numerosos y complejos sistemas que interactúan entre sí. Conforme el paso del tiempo, ciertos elementos de éste pueden ir degradándose, dando síntomas de fallo para finalmente desembocar en un fallo que puede tener grandes consecuencias, y que sería conveniente monitorizar con el fin de garantizar su correcto funcionamiento. Por esta razón, en los últimos años, y gracias sobre todo al desarrollo de entornos que manejan grandes cantidades de datos, el interés en el desarrollo de algoritmos de mantenimiento predictivo ha ido creciendo ya que se abre un nuevo marco que ofrece infinitud de posibilidades y aplicaciones. Para este propósito, propondremos a continuación la formulación matemática necesaria para afrontar nuestro problema, siguiendo el esquema propuesto en [2].

Sea una serie temporal multivariada X_t , cuyos parámetros se empiezan a monitorizar a partir de cierto instante de tiempo, $X_t \in \mathbb{R}^d, t \geq 0$, donde cada instante *t* es igual a un múltiplo de *1/fs*, siendo *fs* la frecuencia de muestreo, y un histórico denotado como $D = \langle X_t \rangle_{t=0}^{T}$, con *t*<*T*, el objetivo es el de predecir los nuevos eventos $Y_t \in [0,1], t > T$, donde Y_t indica si el evento se va a producir, $Y_t = 1$, o no, $Y_t = 0$. Por ello, si seguimos un esquema de aprendizaje supervisado, emplearemos las etiquetas de fallo, 1, o éxito, 0, para el entrenamiento de los modelos.

Para la aproximación supervisada, sea el *dataset* histórico $\{X_t, Y_t\}_{t=0}^{T}$, el modelo predictivo se construye como:

$$f: X_{T-l-h}, X_{T-l+1-h} \dots, X_{T-h} \rightarrow [0,1],$$

donde *l* es el llamado *lag* o número de muestras que tomamos para realizar la predicción y *h* el horizonte de predicción, esto es, el número de unidades temporales que nos queremos anticipar. Con ello, dadas nuevas observaciones $X_t , t > T$, obtenemos la probabilidad de un nuevo evento mediante el modelo $p_t = f(X_{t-l-h}, X_{t-l+1-h} \dots, X_{t-h}), t > T$, prediciendo fallo o éxito en función de un umbral de alerta ϑ, donde $p_t > \vartheta$ indicaría fallo, con $\vartheta \in (0,1)$.

2.3.1 Creación de vectores de características

En el apartado anterior introdujimos los fundamentos matemáticos para resolver nuestro problema, con ello, pasaremos en este apartado, a analizar cómo se crean los vectores de características que van a servir de entrada a nuestros modelos de aprendizaje automático, los cuales describiremos más adelante. Antes de la creación de vectores de características, debemos tener en cuenta las siguientes cuestiones:

- Como ya indicamos anteriormente, el evento se produce en la fase de arranque de motores principales, la cual se indica mediante unas señales de control determinadas, por lo tanto, restringimos los datos empleados por el modelo a una ventana concreta entorno a los flancos de subida de esas señales.
- El horizonte de predicción *h* indica cuántos arranques hacia atrás respecto del actual nos queremos anticipar. Generalmente, *h* es un indicador de instantes de tiempo, los cuales pueden ser, por ejemplo, horas, vuelos, días o semanas, sin embargo, dividimos el espacio temporal en arranques de motores principales, ya que es lo que nos interesa predecir, esto es, dentro de cuántos arranques el APU es probable que falle.

- El *lag* indica el número de muestras o instantes de muestreo hacia atrás que tomamos a partir del arranque perteneciente al instante *t-h*, donde el arranque actual se produce en *t*.

Con todo ello, los vectores de características se crean de forma iterativa de la siguiente manera:

- Cada fase de arranque de motores tiene un identificador único, con ello, dividimos el espacio temporal en unidades de arranque.
- Disponemos de una etiqueta para cada arranque, 1 ó 0, la manera en la que se obtienen se presentará en la sección de desarrollo del trabajo.
- Dados un horizonte de predicción *h* y un *lag*, *l*, por cada arranque se toman *l* muestras hacia atrás a partir del *h* arranque anterior, el número de vectores de características será entonces igual al número de arranques.
- Si disponemos de un *set* de características de tamaño *F*, el tamaño de cada vector de características tendrá dimensión $F * lag + 1$, siendo 1 la contribución de la etiqueta. La matriz final de vectores de características, siendo *N* el número de arranques, tendrá *N* filas y *F*lag+1* columnas.

Para ejemplificar todo lo anterior, mostramos en la siguiente figura 5 arranques de APU, dispuestos en el eje temporal *t*, nótese que se trata de un esquema simplificado, en el que los arranques se dan consecutivamente uno inmediatamente después que otro, hecho que no sucede en un ejemplo real. Por ello, a partir del quinto arranque se ilustra la manera en la que se crea el vector de características asociado a él, tomando, en este ejemplo, un horizonte de predicción igual a 2 arranques y un *lag* con *l* muestras, además, debemos añadir a ese vector la etiqueta en función de si el arranque ha sido fallido, 1, o satisfactorio, 0, ya que disponemos de ella.

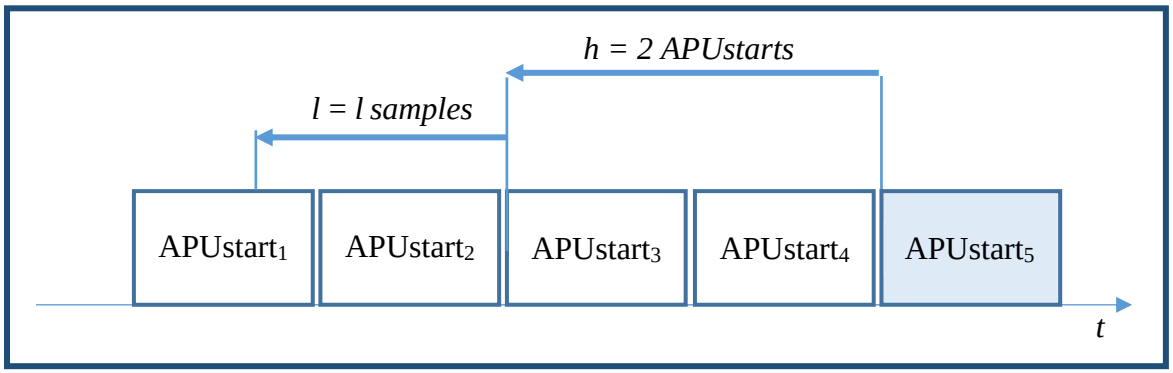

Figura 2-6: Esquema simplificado de creación de vectores de características para un arranque en concreto, con un horizonte de predicción *h* de 2 arranques y un histórico de *l* muestras

2.4 Modelos de aprendizaje automático

Una vez expuesta la formulación y técnicas en que nos vamos a basar para resolver nuestro problema, pasaremos a continuación a describir los diferentes algoritmos de aprendizaje automático que vamos a emplear en nuestro problema, justificando el porqué de su elección y describiendo sus principales bondades y características.

Como ya hemos comentado anteriormente, disponemos de las etiquetas de dos clases, fallo o éxito, en las que dividimos nuestro problema, por ello, analizaremos en primer lugar un paradigma de clasificación de dos clases y en segundo lugar uno relativo a la problemática de la detección de anomalías.

La primera aproximación consiste en emplear un clasificador cuya salida sea la probabilidad de pertenencia a una clase u otra, mientras que, en la segunda, considerábamos a priori conveniente e interesante tratar el problema como detección de anomalías, o lo que es lo mismo detección de *outliers*, donde la salida del sistema indica si una muestra de entrada se considera como una realización inusual o anómala del sistema físico, un *outlier* en términos estadísticos. La motivación principal que nos lleva a estudiar la segunda aproximación radica en que consideramos que el APU, como elemento físico, tiene un funcionamiento normal o esperado, esto es, muestras cercanas en el espacio de características y que siguen cierta estructura, mientras que los fallos o *shutdowns* corresponderían a muestras que difieren de cierta manera de la estructura de funcionamiento correcto, es por ello que presentamos los eventos de fallo como muestras anómalas fuera del rango normal de operación de la máquina.

2.4.1 Modelos de clasificación

Como ya hemos indicado en el apartado anterior, la primera aproximación que seguimos para resolver nuestro problema es el empleo de un clasificador de dos clases, fallo o éxito, en el que, dada una muestra de entrada, su salida sea la probabilidad de pertenencia a una clase u otra. Como clasificadores estudiaremos, por un lado, métodos basados en árboles de decisión tales como *Random Forests*, *Gradient Boosted Trees* y por otro lado las máquinas de vector soporte o SVM, cuyo objetivo es el de encontrar el hiperplano que maximice la separación de las clases.

2.4.1.1 Random forests

El algoritmo *Random Forest*, introducido en [12], consiste en una combinación de árboles de decisión. El algoritmo se basa en el método de *bagging* o combinación de clasificadores, donde se genera un gran número de árboles incorrelados que votan por la clase más popular, en función del vector de entrada. Además, puede ser empleado en problemas de clasificación y regresión, siendo nuestro caso el primero.

En cuanto a los parámetros más importantes del modelo, y los cuales buscaremos optimizar son:

- Profundidad: indica el número de particiones de cada árbol que se utilizan para realizar la predicción.
- Número de estimadores: cantidad de clasificadores empleados.
- Número de características: indica la cantidad de atributos partiendo de los originales, que se emplean en cada clasificador.
- Pesos de las clases: como ya indicamos anteriormente, podemos dar más peso a cierta clase que otra, esto es conveniente si nos encontramos con un gran desbalanceo entre ellas.

Cabe destacar que, si empleamos un gran número de árboles de decisión, la velocidad de predicción puede no ser buena para una aplicación en tiempo real, por lo que deberemos

tener esto en cuenta si implementamos el algoritmo en un sistema real de alertas. Por último, la principal motivación para emplear este algoritmo se debe a su capacidad de generalización y rendimiento en clasificación para *datasets* muy diversos, tal y como podemos ver en [12].

2.4.1.2 Gradient Boosted trees, AdaBoost

Los algoritmos que hacen referencia al término *boosting* se basan en construir una hipótesis fuerte a partir de utilizar hipótesis simples y débiles [13]. El algoritmo utilizado en este trabajo es *AdaBoost, AdaptativeBoosting*, que se basa en la creación y refuerzo de manera iterativa de clasificadores basados en su predecesor que se consideran clasificadores débiles, *weak learners,* y que guarda estrecha relación con *Random Forest*, ya que la decisión final se basa en la votación de los distintos clasificadores. Por un lado, hay que tener en cuenta que la optimización se realiza por descenso del gradiente de acuerdo con una función de pérdida determinada, por otro lado, es importante destacar que la técnica de *boosting* se centra en cada paso en las muestras difíciles de clasificar, lo que puede ser de gran ayuda si nos enfrentamos a clases desbalanceadas.

En cuanto a los parámetros más importantes del modelo, tenemos:

- *Función de pérdida*: es de tipo exponencial, la cual penaliza las predicciones incorrectas y tiene esta forma, $\mathcal{V}(f(x), y) = e^{-\beta y f(x)}$
- *Tasa de aprendizaje,* η: indica el ratio en el que se va descendiendo por el gradiente para optimizar la función de pérdida.
- *Profundidad, número de estimadores y características*: idénticos a la definición para *Random Forest*.

2.4.1.3 SVM

Los algoritmos anteriormente presentados se basaban en conjuntos de árboles de decisión para realizar la predicción. A continuación, analizaremos otro tipo de algoritmo, las máquinas de vector de soporte, SVM, *Support Vector Machines,* cuyo objetivo es el encontrar el hiperplano que maximice la separación entre las clases en el espacio de alta dimensión. Este algoritmo ha sido empleado en numerosas tareas de clasificación [14], dando excelentes resultados en tareas de reconocimiento de firma manuscrita y tareas de visión por ordenador.

En cuanto a la función de coste a minimizar por SVM, la cual es una ligera modificación de aquella propuesta en la regresión logística, tiene la siguiente forma:

$$\min_{\theta} \left[\sum_{i=1}^{m} y^{(i)} \left(-log h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \left(-\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)\right] + \frac{\lambda}{2}\sum_{j=1}^{n} \theta_j^2 \ ,$$

donde θ es el vector de parámetros a optimizar, *x* el vector de datos, *y* el vector de etiquetas y λ el parámetro de regularización que está relacionado inversamente con , $C = \frac{1}{\lambda}$, e indica la cantidad de margen en el que se mueve el hiperplano, un valor grande indica un margen pequeño lo que puede resultar en *overfitting*, mientras con un valor pequeño tendremos un mayor error de clasificación durante el entrenamiento.

Con ello, los parámetros más importantes de este modelo son:

- *El parámetro C*: controla el margen durante el entrenamiento.
- *El kernel*: se trata del tipo de frontera empleada en la clasificación, puede ser de tipo lineal, polinómica o radial, conocida esta última como *rbf, radial basis function* o *kernel* gaussiano.
- *Factor gamma:* se trata del coeficiente de los distintos *kernel.*
- *Pesos de las clases*: con ello podemos dar más peso a cierta clase que otra, esto es conveniente si nos encontramos con un gran desbalanceo entre ellas.

2.4.2 Modelos de detección de anomalías

En muchas aplicaciones es interesante decidir si una muestra pertenece a la misma distribución que las observaciones existentes o por el contrario es una realización no esperada y diferente estadísticamente [15]. Antes de nada, debemos distinguir entre los conceptos de:

- *Detección de outliers*: los datos de entrenamiento contienen muestras que difieren del comportamiento general, por lo que se buscan regiones que comprendan observaciones similares, mientras que se descartan las otras.
- *Detección de anomalías*: los datos de entrenamiento no contienen *outliers*, por lo tanto, estaremos interesados en decidir si una nueva observación es un *outlier* o no. Esta aproximación ha sido ampliamente usada en aplicaciones, por ejemplo, para detección de fraude o comportamientos inusuales del tráfico de una red de ordenadores.

Con ello, podemos trasladar el concepto de detección de anomalías para intentar resolver nuestro problema, ya que, podemos suponer que el APU, como sistema físico, tiene un comportamiento a priori esperado, esto es, los datos pertenecen a una distribución estadística concreta, sin embargo, la máquina puede experimentar fallos y comportamientos inesperados, que diferirían y no corresponderían con la distribución esperada o de funcionamiento correcto. Debido a ello, realizamos la analogía de que un fallo o evento fallido de la máquina correspondería con un *outlier* o muestra no esperada, por lo que si empleamos modelos que sean capaces de predecir si una muestra a la entrada del modelo es un *outlier*, serán capaces de indicarnos si hay que realizar alguna tarea de revisión o mantenimiento de la máquina. Concretamente, en nuestro problema tenemos arranques exitosos y arranques fallidos, los primeros suponemos que no están contaminados por *outliers* mientras que los segundos se comportan y se presentan a modo de *outliers.* Para este efecto, existen numerosos modelos que se han estudiado en el estado del arte, en nuestro trabajo emplearemos *Isolation forests* y *one-class SVM* [16].

2.4.2.1 Isolation Forests

El algoritmo de *Isolation forests*, o *iForests* [17], es similar a los algoritmos basados en conjuntos de árboles de decisión que hemos visto anteriormente como *Random Forest* o *AdaBoost*, en los que en cada árbol se creaba seleccionando ciertas características y finalmente se predecía una clase u otra en función de la clase más votada. Sin embargo, el paradigma cambia, ya que en el entrenamiento disponemos sólo de muestras de la clase de éxito y el objetivo es aislar explícitamente los *outliers* o anomalías. Para este efecto, el algoritmo parte de la idea de que un punto normal requiere más particiones, su camino en el

árbol es más largo en media, para ser aislado que un *outlier*, cuyo camino medio es más corto, tal y como podemos observar en la siguiente figura:

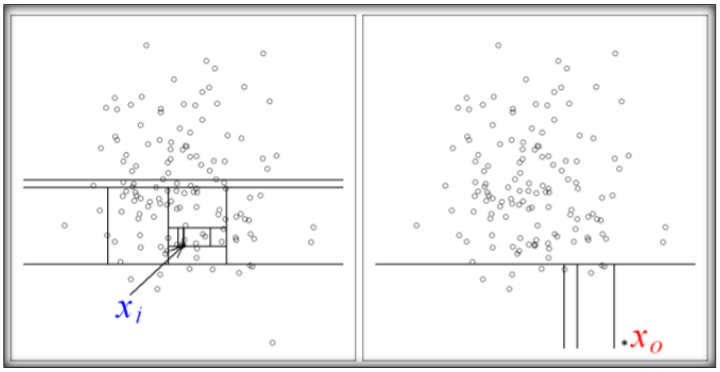

Figura 2-7: Ejemplo de aislamiento de una muestra normal y una anomalía con *iForests* [17]

Por ello, para clasificar entre una u otra se recurre a una puntuación que indica el grado de anomalía de la muestra, *anomaly score*, y tiene la siguiente forma:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

donde *h(x)* es la longitud del camino de la observación *x*, *E* el valor esperado, *c(n)* la media de la longitud de una búsqueda no satisfactoria en un árbol binario de búsqueda BST *(Binary Search Tree)* y *n* el número de muestras del *dataset*. Respecto a la puntuación, en [17] si $s \geq 0.6$, la muestra es considerada anomalía.

Por otro lado, los parámetros del modelo son los siguientes:

- *Número de estimadores.*
- *Número de muestras para entrenar cada estimador.*
- *Contaminación*: indica la proporción de *outliers* del *dataset*, se emplea durante el entrenamiento para definir un umbral de la función de decisión.
- *Número de características para entrenar cada estimador.*

Por último, cabe destacar que *iForests* es muy eficiente especialmente en grandes bases de datos, por lo que su capacidad para manejar grandes volúmenes de información es muy deseable en aplicaciones de la vida real, lo cual es interesante en nuestro caso.

2.4.2.2 One-class SVM

Por último, analizaremos otro algoritmo empleado en el paradigma de detección de anomalías, *one-class SVM*. Este algoritmo se basa en los mismos fundamentos matemáticos que introdujimos en la explicación de las SVM en el apartado de modelos de clasificación, sin embargo, este introduce ciertos cambios al ya mencionado.

El objetivo de este algoritmo es el de, partiendo de datos de entrenamiento de éxito exclusivamente, aprender una función de decisión para decidir si una muestra de entrada es un *outlier* o no [18]. Con ello, la función de decisión se traduce en una frontera, que engloba la mayoría de datos de entrenamiento, si la muestra de entrada está dentro de ella, entonces

no será clasificada como anomalía. Por último, es necesario especificar los siguientes parámetros del modelo:

- *Tipo de kernel: rbf,* polinómico o de tipo sigmoide.
- *Factor gamma:* coeficiente relativo al *kernel*.

2.5 Spark como framework para Big Data

En los últimos años, como ya adelantamos en la introducción de nuestro trabajo, se ha puesto gran énfasis y empeño en el desarrollo y mejoras de plataformas que integran una gran cantidad de volumen de datos, debido principalmente al gran potencial y posibilidades que éstos ofrecen. Por ello, Airbus apostó por la creación de una plataforma que permitiera la explotación de los datos procedentes de diversos sistemas de sus aviones, llamada *Skywise* [1], creada en colaboración con la empresa norteamericana *Palantir Technologies* [19]. En la sección de diseño describiremos con más detalle la plataforma *Skywise*, ya que previamente analizaremos su núcleo o base sobre la que se fundamenta, *Spark*™.

Spark™ [20] es un entorno o *framework* para Big Data, que se ha ido imponiendo en los últimos años a otros entornos como, por ejemplo, *Hadoop,* debido a las siguientes razones:

- Consigue un alto rendimiento, del orden de cien veces más rápido respecto a otros entornos.
- Integra diversos lenguajes ampliamente utilizados en el análisis de datos, tales como, *Java, Scala, Python, R* y *SQL*. Concretamente, emplearemos en nuestro trabajo *PySpark,* intérprete interactivo de Python en *Spark*™.
- Dispone módulos y librerías para manejar bases de datos *SQL,* procesamiento en tiempo real o *streaming*, aprendizaje automático (*MLlib*) y computación de grafos en paralelo (*GraphX).*
- Es compatible con numerosos entornos de gestión y tipos de bases de datos, *Cassandra, Kubernetes, Mesos, Hadoop,* entre otros, y puede correr aplicaciones en modo *standalone* o en la nube.

La manera en que los datos se integran en el entorno *Spark*™ son los llamados RDDs, *Resilient Distributed Datasets*, colección de elementos, tolerante a fallos, que se pueden operar en paralelo. Los tipos de *datasets* que *Spark*™ puede paralelizar son muy diversos, desde ficheros de texto a datos en formato HDFS, *Hadoop Distributed File System, Cassandra, HBase* o *Amazon S3,* empleado en el entorno propio de Amazon llamado AWS *o Amazon Web Services.* En cuanto a las operaciones que se pueden realizar con los RDDs son de dos tipos, transformaciones, que crean un *dataset* partiendo del original, y las acciones, que devuelven un valor después de haber computado la transformación. Esta manera de procesar la información es de gran relevancia, ya que el rendimiento y velocidad de las transformaciones de los datos depende de ella, debido a que *Spark*™ no evalúa cada acción en el momento, sino que crea un grafo llamado DaG, *Directed Acyclic Graph,* que recuerda las operaciones y sólo las ejecuta cuando el programa principal o *driver program* demanda un resultado, por ello se dice que *Spark*™ es un entorno de tipo de "*evaluación perezosa*" o *lazily evaluated.* Otra característica importante de los RDD es que la secuencia de operaciones efectuadas se guarda para poderlos recuperar de manera eficiente si alguna máquina deja de funcionar.

En cuanto a la arquitectura se refiere, ésta sigue un esquema maestro-esclavo, por ello, con el fin de entender cómo se ejecutan los procesos, se reflejan en la siguiente figura los elementos más importantes:

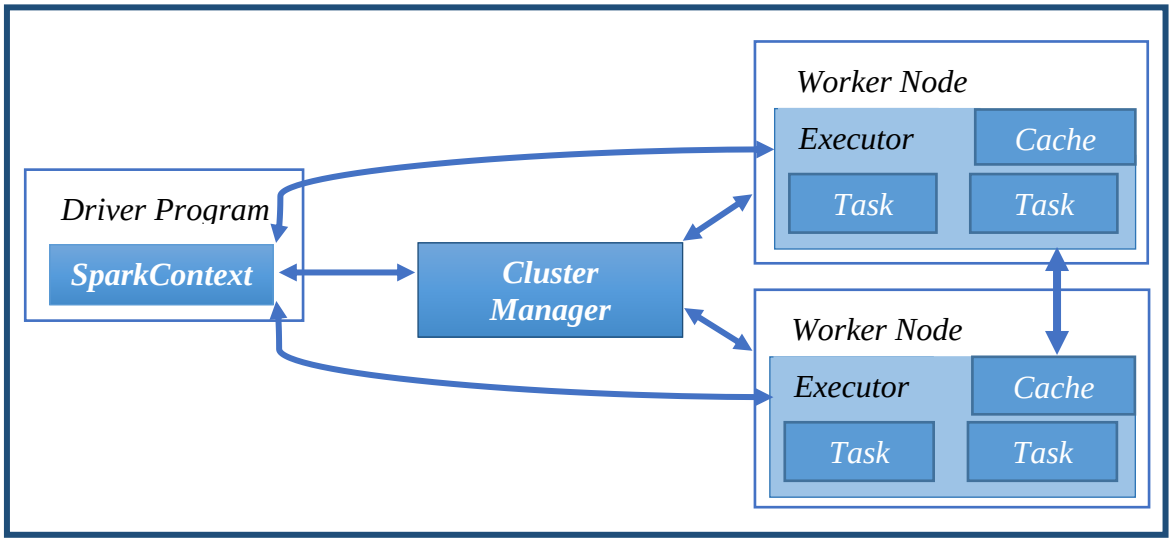

Figura 2-8: Arquitectura simplificada de *Spark*™, extraída de la documentación oficial [20]

Con ello, las características más importantes de los elementos de la arquitectura son las siguientes:

- *Driver program:* ejecuta el código principal de la aplicación, se encarga de negociar los recursos con el *cluster manager* y alberga el contexto o *SparkContext*. El contexto contiene toda la información necesaria para el funcionamiento de la aplicación, esto es, la programación de las tareas, metadatos y grafo para la ejecución. El grafo lo presentamos anteriormente como DaG y contiene todas las operaciones sobre los RDDs.
- *Cluster manager:* se encarga de adquirir los recursos necesarios y reservárselos a los nodos trabajadores o *workers nodes*.
- *Worker node:* realizan los trabajos demandados, devolviendo el resultado al *driver* o programa principal. Los nodos disponen de una memoria caché para comunicarse entre ellos si fuera necesario.

3 Diseño

3.1 Skywise

Anteriormente presentamos *Skywise* como la plataforma de análisis de datos de Airbus, basada en *Spark*™, la cual está gestionada en colaboración con la empresa norteamericana *Palantir Technologies*, con ello, en este apartado describiremos los elementos más importantes y las herramientas empleadas en el desarrollo de este trabajo. Concretamente, y para entender el volumen y la magnitud de datos que maneja la plataforma, el último avión de Airbus, el A350, dispone de 50.000 sensores a bordo, lo que supone 2.5 *terabytes* de datos diarios correspondiente a la operación de un único avión.

Figura 3-1: Logotipo de la plataforma *Skywise*, extraído de [1]

La razón de existencia de esta plataforma viene motivada principalmente por los siguientes puntos:

1. En general, el potencial que el análisis de datos puede ofrecer para entender la operación de los aviones, optimizar el mantenimiento, ayudar a la toma de decisiones, reducir los costes y facilitar la colaboración entre Airbus, aerolíneas y suministradores.
2. Concretamente, desde el punto de vista de las aerolíneas, éstas pueden mejorar la eficiencia de sus operaciones, reducir interrupciones operacionales y en general operar de una manera más eficiente, por ejemplo, aerolíneas como *EasyJet*, *Delta Airlines* o *Emirates* se han sumado ya a *Skywise*.
3. Además, los suministradores de los componentes de los distintos sistemas del avión pueden beneficiarse del análisis de datos que permite esta plataforma para mejorar sus productos.

Por último, destacar que la forma en la que se disponen los datos en *Skywise* es en la de un repositorio centralizado, que permite almacenar datos de cualquier tipo, ya sea de forma estructurada o en bruto, que se conoce como *data lake* [21]. De esta forma, podemos analizar de diversas formas los datos, visualizando su comportamiento en cuadros de mando o *dashboards*, realizando analítica en tiempo real y llevando a cabo a cabo tareas de aprendizaje automático para ayudar a la toma de decisiones.

3.1.1 Herramientas utilizadas

Una vez introducida la herramienta *Skywise*, así como sus principales propósitos y características, describiremos a continuación las herramientas integradas en la plataforma con las que hemos trabajado para llevar a cabo nuestras tareas. La primera de ellas es una herramienta cuyo objetivo es visualizar, y presentar a modo de cuadro de mando interactivo, los diferentes parámetros del avión, mientras que la finalidad de la segunda es la del tratamiento y codificación de transformaciones de los datos.

3.1.1.1 Herramienta de visualización de datos

En *Skywise* disponemos de una herramienta determinada de visualización de series temporales, la cual nos permite estudiar y analizar el comportamiento de los parámetros demandados. Con ello, esta herramienta se presenta a modo de *dashboard* interactivo, donde el usuario interactúa de forma dinámica con los datos y cuyas funcionalidades son parecidas a las de la librería de código abierto *d3.js* [22], escrita en *javascript*.

En cuanto a las funcionalidades de la herramienta se refiere, encontramos las siguientes:

- Nos permite añadir los parámetros disponibles a distintas gráficas, con ello es necesario especificar el nombre del parámetro deseado y el avión correspondiente.
- Una vez dispuestas las gráficas en las diferentes figuras que nosotros determinemos, podemos realizar distintas operaciones sobre ellas. En primer lugar, podemos navegar por el eje temporal para visualizar un periodo de tiempo concreto y hacer *zoom* en zonas de nuestro interés. En segundo lugar, podemos efectuar distintas operaciones sobre las series temporales, por ejemplo, obtener la derivada, realizar desplazamientos temporales, buscar regiones donde se den varias condiciones a la vez, útil para ver cuando se arrancan los motores en nuestro caso, en definitiva, nos proporciona transformaciones que pueden ser útiles para llevar a cabo nuestro análisis y visualización.

Por último, esta herramienta será de gran importancia en nuestro trabajo ya que gracias a ella podemos visualizar las señales involucradas en nuestro problema, comprendiendo mejor su comportamiento y entendiendo la naturaleza de ellas, de todas formas, analizaremos con más detalle el tipo de señales analizadas y la metodología que se ha seguido en la fase de desarrollo de este trabajo.

3.1.1.2 Herramienta de codificación, extracción y transformación de datos

Además de la herramienta de visualización integrada en *Skywise*, en este trabajo hemos empleado una herramienta que permite la codificación, extracción y transformación de los datos, la cual será de vital importancia en nuestro trabajo para generar los *datasets* en la forma apropiada para nuestro análisis. Esta herramienta contiene un intérprete de *Python* que se ejecuta sobre *Spark*™, *PySpark*, y permite, como ya hemos introducido, lo siguiente:

- Mediante una API propietaria, realizar la extracción y preprocesamiento de los datos, por lo que es necesario especificar los sensores, el rango temporal, los aviones, frecuencia de muestreo e interpolación deseados.
- Una vez obtenidos los datos, podemos, gracias a *PySpark* y ciertas librerías que describiremos más adelante, realizar transformaciones y análisis sobre esos datos, además de consultas de tipo *SQL* mediante *PySpark SQL*.

Con ello, esta herramienta será de gran utilidad en nuestro caso para generar los vectores de características en la forma adecuada y realizar análisis de diversa índole sobre los datos. Por

último, la manera en que se codifican las transformaciones y el detalle de cómo se han realizado se estudiarán en el desarrollo del trabajo.

3.2 Flota analizada

En este proyecto trabajaremos con la flota del A380 de varias aerolíneas[1], en concreto, y como ya hemos comentado anteriormente, el objetivo será el de diseñar un prototipo de sistema de aprendizaje automático que anticipe un evento de fallo del APU que afecta a la flota. Por otro lado, a continuación mostramos una ilustración del A380, el avión comercial de pasajeros más grande del mundo, con capacidad de hasta 615 pasajeros distribuidos en dos plantas y con un alcance máximo de 15.000 kilómetros, por todo ello, quisiéramos evidenciar y destacar una vez más la magnitud del impacto de los retrasos y fallos operacionales en este avión especialmente.

Figura 3-2: Vista de un A380, extraída de [23]

Para la realización del trabajo se han tenido en cuenta las siguientes cuestiones:

- El número de aviones involucrados en el análisis es de 27, los cuales están afectados por nuestro evento.
- Nuestro marco temporal de análisis irá de diciembre de 2016 hasta junio de 2018.
- El número y tipo de sensores del APU empleados, así como el porqué de su elección, se analizarán en el desarrollo del trabajo.

Por último, recalcar que se ha estudiado a nivel de sensores, señales, así como el funcionamiento mecánico básico, el modelo concreto de APU de la flota analizada, mediante reuniones técnicas con mi tutor empresarial, Alberto Martínez, ingenieros del departamento y manuales del suministrador [3]. Gracias a ello, hemos adquirido los conocimientos necesarios y la base para entender la raíz del problema y el funcionamiento elemental de la máquina.

3.2.1.1 BBDD de eventos de fallo

Como ya adelantamos en el estado del arte, disponemos de una base de datos que nos indica si se ha producido el evento de fallo durante el arranque o no. Esta información nos será de gran utilidad para conocer a priori qué arranques han sido satisfactorios o no, y con ello seguir una aproximación de aprendizaje supervisado, ya que disponemos de las etiquetas reales, fallo o éxito, para cada vector de características.

[1] Por cuestiones de confidencialidad no nos es posible concretar qué aerolíneas han sido empleadas en nuestro estudio.

En cuanto al etiquetado de los eventos, existe un grupo de trabajo dedicándose a este efecto, así como al desarrollo de algoritmos para buscarlos, ya que el evento está caracterizado por una serie de comportamientos conocidos, además, cuando los motores no arrancan puede ser debido a numerosos factores, de ahí que se haga necesario buscar el evento y confirmar que se trata de nuestro evento de fallo en concreto. Por otro lado, esta base de datos se actualiza según se van encontrando nuevos fallos, por ello, en una implementación futura, nuestro sistema de aprendizaje automático estaría realimentado por ella, siguiendo por ejemplo un esquema de aprendizaje por refuerzo, afinando los modelos con el paso del tiempo, sin embargo, para nuestro prototipo inicial, hemos tomado la base de datos en la forma en la que se encontraba a fecha de la realización de los experimentos. Por último, la base de datos contiene los registros de los distintos aviones que han tenido un evento de fallo y cuándo han ocurrido, siguiendo el formato que vemos en la tabla a continuación:

Aircraft	Unix timestamp	Label
A-BCDE	*1540891634000000000*	*1*

Tabla 3-1: Formato de la base de datos de eventos

3.3 Metodología, equipos y software empleados

Para concluir el apartado de diseño, describiremos a continuación el software que se ha empleado para la realización de este trabajo, así como la metodología seguida.

En primer lugar, para la extracción de los datos y etiquetado de los vectores de características, se ha utilizado la herramienta de codificación anteriormente descrita, beneficiándonos así de las bondades de un entorno de computación distribuida para la extracción y procesamiento de grandes volúmenes de información, para tal efecto existe una API determinada en la plataforma, escrita en *Python*. En segundo lugar, para la realización de los experimentos y entrenamiento de los modelos de aprendizaje automático, se ha empleado una estación de trabajo industrial disponible en el departamento, cuyas características más importantes son las siguientes:

- 32 GB de memoria RAM.
- 2 procesadores Intel® Xeon® de 2.90GHz, con un total de 32 núcleos de procesamiento disponibles, 16 por procesador.
- Tarjeta gráfica NVIDIA® Quadro® P4000.

Por otro lado, el software instalado en la estación de trabajo ha sido el siguiente:

- *Anaconda:* es un entorno de análisis de datos utilizado ampliamente en la industria, con numerosas librerías para el desarrollo, testeo y entrenamiento de modelos de aprendizaje automático [24], en la que podemos programar en *Python* y *R* para realizar nuestros análisis estadísticos y experimentos.
- *Jupyter Notebook:* corre bajo *Anaconda* a modo de aplicación web que permite crear y compartir documentos que contienen código, visualizaciones y texto [25], además es compatible con *Spark*™. Con ello, esta aplicación es muy conveniente para la presentación de resultados y depuración de código, el cual desarrollaremos en *Python.*

Figura 3-3: Logotipos de Anaconda y Jupyter Notebook [24] [25]

Por último, describiremos las librerías principales utilizadas en *Jupyter* para el desarrollo de nuestro trabajo:

- *Scikit-learn:* librería para la realización de tareas de aprendizaje máquina y análisis de datos en *Python* [26], contiene numerosas utilidades para ello, tales como, una gran variedad de algoritmos de clasificación, regresión, *clustering*, visualización y reducción de dimensionalidad, así como módulos para el preprocesamiento de datos y ajuste de parámetros de los modelos.
- *Matplotlib y seaborn:* compatibles con el formato de datos utilizado en *scikit-learn* que permiten generar gráficos y reportes a partir de ellos [27] [28].
- *Imbalanced-learn:* compatible con *scikit-learn*, además de *Keras* y *Tensorflow*, que contiene implementaciones de algoritmos que tratan el desbalanceo de clases [29].
- *Multicore t-SNE:* implementación, que permite varios núcleos de procesamiento, del algoritmo de visualización de datos *t-SNE* [30].

4 Desarrollo

4.1 Fases del desarrollo del trabajo

Una vez definidos el estado del arte y el diseño que vamos a seguir, describiremos a continuación el detalle de los diferentes puntos del desarrollo del trabajo, así como la metodología seguida. Por ello, de acuerdo con los objetivos que expusimos en el apartado de introducción, el flujo de trabajo podemos dividirlo en las siguientes fases:

1. *Estudio del estado del arte, APU y el evento de fallo*: el primer paso consistió en el estudio de la máquina mediante manuales propios del suministrador [3] y reuniones técnicas con el equipo del departamento, gracias a las cuales, se pudo entender el comportamiento físico fundamental de la máquina, así como sus características más importantes desde un punto de vista mecánico y de señal. Con ello, comprendimos porqué y cómo se producía el evento de fallo y qué señales, a priori, eran susceptibles de desencadenarlo, de esta forma, adquirimos la base a partir de la cual intentar abordar el problema a partir de algoritmos de aprendizaje automático basándonos en el estado del arte.
2. *Visualización de las señales involucradas en Skywise:* una vez estudiadas las características más relevantes del APU y la naturaleza del evento de fallo, procedimos al estudio y visualización de las señales que presumiblemente tenían que ver con éste. De esta forma, gracias a la herramienta de visualización disponible en la plataforma *Skywise*, pudimos analizar y estudiar el comportamiento de las señales durante el funcionamiento del APU y arranque de motores principales, siendo de gran ayuda para ver los distintos tipos de señales que intervienen en el sistema.
3. *Definición de los experimentos:* teniendo en cuenta las necesidades y particularidades del problema, definimos un marco experimental determinado, esto es, el formato de los *datasets* de entrada a los algoritmos, el tipo de algoritmos y cómo evaluar el rendimiento de este sistema de acuerdo con unos experimentos determinados.
4. *Codificación en Skywise:* con el estado del arte estudiado y comprendido, tanto el APU como los fundamentos matemáticos y procedimientos relativos al aprendizaje automático, así como la definición de los experimentos establecida, comenzamos a codificar un *pipeline* de extracción y procesamiento de los datos, cuya salida son los diferentes *datasets* que contienen los vectores de características con su etiqueta asociada, 0 si es un evento exitoso o 1 si es un fallo.
5. *Entrenamiento y evaluación de los modelos:* en esta fase desarrollamos los módulos necesarios para entrenar los distintos tipos de algoritmos y cómo evaluarlos teniendo en cuenta las necesidades particulares de nuestro problema.
6. *Presentación de los resultados:* con los resultados de los experimentos, la fase final consistió en presentarlos para analizar el rendimiento de los algoritmos. Además, como ya comentamos en el estado del arte, el objetivo principal es el de lograr el mejor rendimiento de acuerdo con unas métricas dadas.

4.2 Skywise: visualización y codificación

En el siguiente apartado analizaremos el trabajo desarrollado en la plataforma *Skywise*, la cual tiene dos partes, la primera consiste en el manejo de la herramienta de visualización mientras que la segunda nos permite codificar en *Python* las transformaciones deseadas sobre los datos.

4.2.1 Herramienta de visualización

Como ya introdujimos en el diseño de este trabajo, la plataforma *Skywise* dispone de una herramienta interactiva de visualización de series temporales, con ello, podemos analizar el comportamiento de los sensores que sean de nuestro interés. Para tal efecto, en primer lugar, disponemos de un buscador de señales en el que debemos indicar el nombre del sensor y el avión deseados, de esta manera, podemos incluir en uno o varios gráficos tantas señales como queramos. En segundo lugar, una vez dispuestas las series temporales en los gráficos determinados, podemos hacer operaciones sobre ellas y navegar en el eje temporal, algunos ejemplos de ellas utilizados en el trabajo son los siguientes:

- *Time series search event:* la herramienta dispone de una funcionalidad gracias a la cual ésta busca y resalta regiones temporales donde se den ciertas condiciones definidas a priori, con ello, podemos por ejemplo buscar los arranques de APU, los cuales están definidos por el comportamiento específico de ciertas señales, además, la herramienta contabiliza cuántos eventos ha encontrado, lo cual es muy útil para nuestros análisis.
- *Derivada de la señal:* el cálculo de la derivada de una serie temporal resulta interesante si no estamos interesados tanto en el valor absoluto de ella sino en la variación la señal a lo largo del tiempo, la cual puede indicarnos cierta degradación.
- *Multiplicación de señales:* interesante si nuestra intención es realizar máscaras con señales binarias de control sobre otras señales, por ejemplo, para restringir los valores de los sensores a una región de arranque de APU, la cual viene dada por ciertas señales.

4.2.1.1 Selección y estudio de características

Una vez explicada la herramienta de visualización, procederemos a continuación a describir el criterio tomado para la selección de las características. Como ya comentamos anteriormente, las características que se han tenido en cuenta, tanto para visualizar el comportamiento del APU como para la codificación de los experimentos, vienen dadas por la experiencia y el conocimiento a priori sobre el problema. Con ello, mediante reuniones técnicas con el equipo se definieron qué parámetros, fundamentando la decisión en el funcionamiento físico de la máquina, eran los más propensos a desencadenar el evento, además de las ventanas temporales en los que se tenían que tomar para el análisis.

Definidos ya los parámetros, el siguiente punto en nuestro trabajo fue visualizarlos en la herramienta disponible en *Skywise* y entender su comportamiento desde un punto de vista de señal, lo que fue de gran ayuda para el desarrollo de los algoritmos, ya que conocíamos, primero, de qué naturaleza eran las señales involucradas, y segundo, cómo se relacionaban entre ellas. Finalmente, se optó por tomar un *set* de 12 características, una parte de ellas eran señales binarias de control y otra eran señales procedentes de sensores que miden magnitudes continuas.

4.2.1.2 Distribución de eventos por avión

Como ya describimos en el apartado de diseño, para el análisis de nuestro problema tendremos en cuenta lo siguiente:

- La flota analizada se compone de 27 aviones, los cuales están afectados por el evento.
- El marco temporal de análisis abarca el período comprendido entre diciembre de 2016 hasta junio de 2018.

Teniendo en cuenta lo anterior, y para visualizar cómo afecta el fallo a cada avión, representamos en la siguiente figura la distribución del número de eventos fallidos de los 27 aviones involucrados:

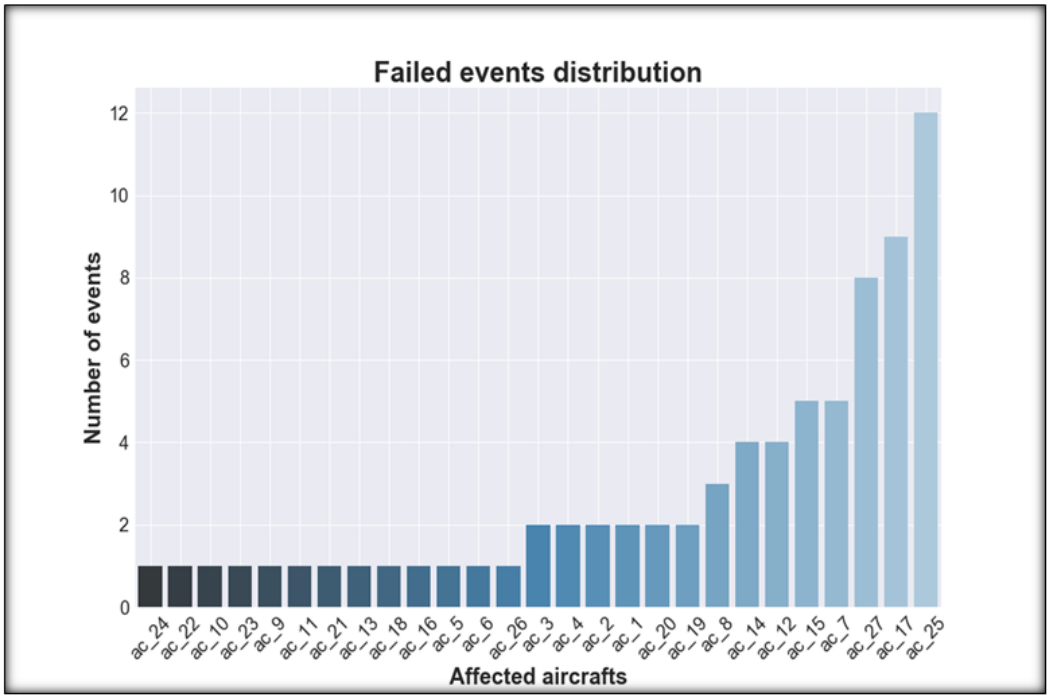

Figura 4-1: Distribución del número de eventos de fallo por avión

A la vista del resultado obtenido, el evento no se presenta de manera uniforme en la flota, esto es, hay aviones a los que les afecta más que al resto, siendo el más perjudicado el *ac_25* con un total de 12 fallos de los 78 totales. Por otro lado, en ese mismo período de análisis tenemos un total de 34545 eventos satisfactorios, lo que hace una proporción de 1:442, un evento de fallo por cada 442 exitosos, tal y como se refleja en la siguiente tabla:

Successful events	*Failed events*	*Fail:Success ratio*	*Failed events %*
34545	*78*	*1:442*	*0.22 %*

Tabla 4-1: Comparativa entre número de eventos exitosos y fallidos

Teniendo en cuenta la proporción de éxitos y fallos, nos encontramos ante un problema de desbalanceo de clases, en el que hay más representación de una clase que de otra, tal y como adelantamos en el apartado donde describimos el estado del arte en nuestro trabajo. Este fenómeno será determinante y tendrá un gran impacto en el rendimiento de nuestros algoritmos ya que disponemos de pocos datos de fallo, de todas maneras, dedicaremos un apartado en el desarrollo del trabajo a analizar y cuantificar este hecho, conocido como *curse of imbalanced datasets*.

4.2.2 Herramienta de codificación, extracción y procesamiento de los datos

En este apartado, nos centraremos en la descripción de la herramienta de codificación disponible en *Skywise*, la cual nos permite extraer la información y realizar diversas transformaciones sobre ella. La herramienta consiste en un entorno de programación en *PySpark*, el cual contiene las librerías necesarias para realizar tareas de extracción, procesamiento, visualización y codificación de las transformaciones de los datos. Con ello, partiremos de ella para generar los *datasets* que contienen los vectores de características con sus etiquetas tal y como definimos en el apartado *2.3.1* de este trabajo.

4.2.2.1 Generación de los vectores de características

La herramienta de codificación anteriormente mencionada nos permite, gracias a las bondades de un entorno de computación distribuida basado en *Spark*™, procesar un gran volumen de datos de un número considerable de aviones. Con ello, para generar los vectores de características, emplearemos un esquema de programación funcional.

La programación funcional es un paradigma de programación en el que se opta por el uso de funciones y composición de éstas para realizar las tareas deseadas, en nuestro caso, transformaciones sobre los datos, permitiendo tener un código más flexible, depurable y organizado. De esta forma, nuestro módulo de codificación estará compuesto por los siguientes bloques funcionales, donde la salida es la composición ordenada de todos ellos:

1. *Extracción y procesamiento de las series temporales:* en este bloque, mediante una API propietaria disponible en *Skywise*, declaramos la lista de aviones deseados, nombre de sensores, frecuencia de muestreo, rango temporal de análisis y tipo de interpolación, ya que no todos los sensores se muestrean con la misma frecuencia, de esta manera tenemos un *dataset* en el que cada fila corresponde a un instante de muestreo y cada columna un valor de un sensor en concreto. Una vez obtenidos los datos, procedemos en las siguientes funciones a aplicar las transformaciones necesarias para crear nuestros vectores de características, teniendo en cuenta, que, para optimizar la computación, indicaremos en *Spark*™ que deseamos realizar las operaciones en grupos o lotes del mismo avión, paralelizando de esta manera el proceso y sacando partido de la computación distribuida, siguiendo el esquema que describimos en el estado del arte.
2. *Filtrado temporal de las señales e identificación de arranques:* para esta fase se definieron en las reuniones técnicas las zonas de donde tomar los datos, las cuales están determinadas por ciertas señales, por ello, se define una ventana temporal entorno a ellas. Finalmente, a cada arranque de APU se le asigna un identificador único.
3. *Creación de vectores de características:* en esta función, se crean los vectores de características tal y como expusimos en el estado del arte del trabajo. Para ello, debemos definir a priori un horizonte de predicción *h,* y un *lag l,* de esta manera, por cada arranque se toman las *l* muestras del *h* arranque anterior, conformando así los vectores de características.
4. *Etiquetado de los vectores de características:* en esta fase final, partiendo de los vectores de características creados en la etapa anterior y de la base de datos de eventos, se añaden las etiquetas correspondientes a cada arranque, 0 si ha sido exitoso o 1 fallido. La manera en que se añaden estas etiquetas es mediante una sentencia en

PySpark SQL llamada *left inner join*, en la que gracias a ella, como sabemos cuándo empieza y termina cada arranque, además del instante temporal en el que ocurrió el fallo, podemos cruzar esa información y asignar una etiqueta de fallo a todos aquellos arranques en cuyo rango temporal de funcionamiento tenemos una etiqueta de fallo. Finalmente, si fijamos un horizonte de predicción *h* y un lag *l*, cada vector de características nos indica que, a partir del arranque actual, tomando *l* muestras hacia atrás, si dentro de *h* arranques se produce un evento fallido o no.

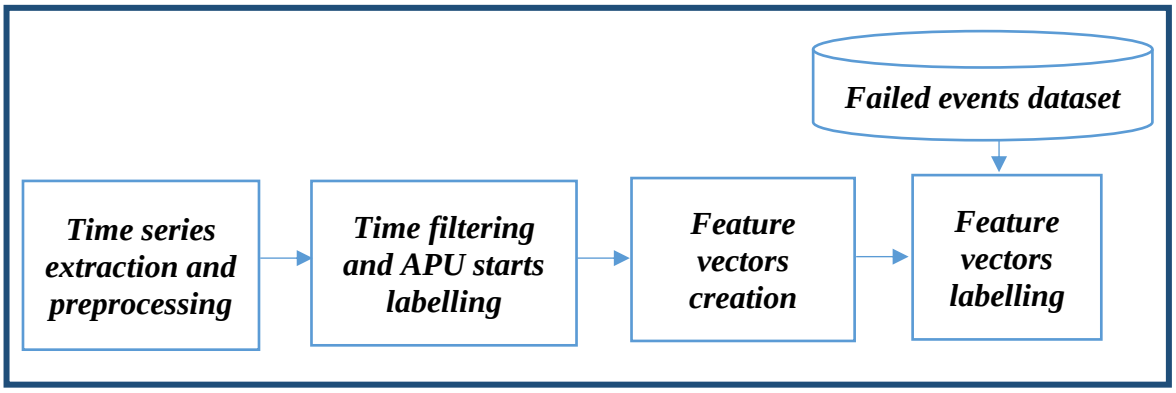

Figura 4-2: Diagrama de bloques de la generación de los vectores de características

4.3 Experimentos

Como ya expusimos en el apartado de diseño de esta memoria, el trabajo realizado se ha estructurado en dos partes, en la primera se ha realizado la visualización y caracterización de las señales involucradas así como la codificación de las transformaciones necesarias para la creación de los vectores de características, mientras que en la segunda, partiendo de esos vectores ya creados gracias a *Skywise*, se han realizado diversos experimentos, los cuales desarrollaremos en los siguientes apartados.

4.3.1 Protocolo experimental

Previamente al desarrollo de los distintos experimentos realizados, es necesario especificar el protocolo experimental que se ha seguido para llevarlos a cabo, el cual desarrollaremos con detalle a continuación.

4.3.1.1 Protocolo para la creación de los vectores de características

En primer lugar, en cuanto al módulo de generación de vectores de características se refiere, debemos definir qué horizonte de predicción *h* y lag *l* empleamos para su creación, tal y como explicamos en el apartado anterior. Para ello, la definición de estas dos variables depende del problema al que nos estemos enfrentando y las necesidades reales del sistema en cuestión, en consecuencia, mediante reuniones técnicas se definieron qué valores se tomarían para la realización de los experimentos, los cuales vemos reflejados en la siguiente tabla, donde el horizonte de predicción está expresado en número de arranques y el *lag* en número de muestras:

Horizon	Lag
1	*24,48,72*
5	
15	
30	

Tabla 4-2: Variación de horizontes y *lags* para la generación de los vectores de características

Teniendo en cuenta la tabla anterior, la justificación de la variación que se ha seguido para ambas variables es la siguiente:

- En cuanto al horizonte de predicción se refiere, este indica el número de arranques deseados para la anticipación del evento, ya sea fallo o éxito. En nuestro caso, acordamos variarlo de 1, que supondría el peor caso de anticipación, esto es, a un arranque sólo, a 30, el cual sería el horizonte de predicción óptimo para poder anticiparnos lo suficiente como para que los agentes correspondientes realizaran las tareas de mantenimiento. Por otro lado, los valores intermedios de 5 y 15 se tomaron para ver cómo evolucionaba el rendimiento en la clasificación de forma progresiva. Por último, debemos tener en cuenta que en media el APU se arranca dos veces al día, por lo que un horizonte de predicción igual a 30 equivaldría a unas dos semanas, tiempo suficiente para realizar las tareas de mantenimiento.
- En segundo lugar, la variación de los *lags* o muestras hacia atrás tomadas por cada arranque a partir del *h* arranque anterior, viene determinada por la ventana temporal de muestras que se ha definido a priori y que analizaremos en este apartado. Teniendo esto en cuenta, cada *lag* equivale el número de arranques hacia atrás de acuerdo a la siguiente fórmula, $\#APUstarts = \frac{lag}{24}$, por lo tanto, los *lags* 24, 48 y 72 tomados, equivalen a la información presente en 1,2 y 3 arranques respectivamente. El porqué de esta variación radica en el interés por ver cómo se comporta el modelo según el número o histórico de muestras de las que dispone.

Por otro lado, en relación con la ventana definida a partir de la cual se obtienen los datos, se acordó tomar la información presente en una ventana a partir de ciertas señales, los valores pertenecen a los 27 aviones afectados por el evento, en un rango temporal entre diciembre de 2016 y junio de 2018. En cuanto al número de características, se toman 12, potencialmente involucradas en nuestro evento de fallo. Para resumir todo lo anterior, definimos la siguiente tabla:

Analysed time range	Number of aircrafts	Number of features	Samples per APU start
Dec 16 – June 18	*27*	*12*	*24*

Tabla 4-3: Resumen de los valores tomados para la selección de muestras y creación de vectores de características.

4.3.1.2 Definición de ratios de oversampling y downsampling para SMOTE-EEN

En el estado del arte de esta memoria dedicamos un apartado a analizar el desbalanceo de clases, así como sus consecuencias, que pueden afectar de manera significativa al rendimiento de los algoritmos. Por ello, y para intentar paliar este fenómeno, introdujimos un algoritmo que realiza una primera fase de *oversampling* de la clase minoritaria y posterior *downsampling* de la mayoritaria, conocido como SMOTE-EEN.

En cuanto a la primera fase, es necesario especificar la proporción de *oversampling* deseada, la cual se expresa mediante un porcentaje respecto a la cantidad de muestras de la clase mayoritaria, por ejemplo, un valor de 0.1 indica que el número de muestras de la clase minoritaria representa el 10% del número total de la mayoritaria, además, tendremos en cuenta un vecindario de 5 muestras para la generación de los datos sintéticos. Por otro lado, en cuanto al *downsampling* de la clase mayoritaria se refiere, es necesario especificar el criterio para descartar o mantener una muestra, con ello, por cada muestra de la clase mayoritaria, se tiene en cuenta un vecindario de 5 muestras, si la proporción de vecinos es mayor para la clase mayoritaria la muestra permanece, de lo contrario se descarta. Cabe destacar que la fase de *downsampling* es idéntica a todas las proporciones empleadas en el *oversampling*, además, el algoritmo SMOTE-ENN lo aplicamos exclusivamente a los datos de entrenamiento. Por último, para ver cómo afecta al rendimiento de los modelos el algoritmo SMOTE-EEN y si gracias a él conseguimos mejores resultados en la clasificación, se varía la proporción de *oversampling* de la siguiente manera:

Oversampling ratio - OVRS	Meaning
0	*No oversampling performed*
0.1	*Minority class represents 10% of majority one*
0.5	*Minority class represents 50% of majority one*
1	*Minority and majority classes proportions are the same*

Tabla 4-4: Ratios de *oversmapling* empleados en los experimentos

4.3.1.3 Generación de los datos de entrenamiento, validación y prueba

Para concluir este apartado, definiremos la manera en la que se han tomado los datos de entrenamiento, validación y prueba para la realización de nuestros experimentos. Por ello, se han seguido las siguientes aproximaciones:

- *Conjunto de entrenamiento y test:* constituyen respectivamente una proporción de *70%* y *30%*, tomada al azar, de los vectores de características. Debido al desbalanceo de clases, y para asegurarnos de que disponemos de al menos un *30%* de las muestras de fallo para el conjunto de test, la división *70%* y *30%* se realiza por separado en función de la clase, posteriormente se unen los conjuntos creados a partir de cada una, de entrenamiento por un lado y de test por otro, con ello, para el entrenamiento tendremos 24181 muestras de éxito y 54 de fallo, mientras que en el *test*, 10363 y 24 respectivamente. Además, si procede, se aplica el algoritmo SMOTE-ENN al conjunto de entrenamiento, con una proporción de *oversampling* específica.
- *Conjunto de validación:* durante la fase de entrenamiento se opta por una validación cruzada del tipo *5-fold cross validation*, con ello, el conjunto de entrenamiento está compuesto de 5 particiones, 1 para la validación, que representa 1/5 o 20% del total, y 4 para el entrenamiento. De esta manera, se realizan 5 entrenamientos y gracias al conjunto de validación, se toma el modelo cuyo rendimiento sea superior en cuanto a una métrica específica.

4.3.2 Estudio del desbalanceo de clases mediante SMOTE-EEN

Como ya estudiamos en el apartado *4.2.1.2* de esta memoria, se da en nuestro problema un gran desbalanceo entre las clases de fallo y éxito, con una proporción de 1:442 respectivamente. Debido a este hecho, proponemos una aproximación mediante el algoritmo SMOTE-EEN para intentar paliar este problema, por lo que estudiaremos a continuación su efecto en el espacio de características y cómo podría ayudar en la tarea de clasificación.

Para la realización de este experimento, se aplica el algoritmo SMOTE-EEN a un conjunto de entrenamiento específico, con ratios de *oversampling* de *0, 0.1, 0.5* y *1*, con el objetivo de visualizar cómo el incremento de muestras minoritarias sintéticas afecta a la estructura del *dataset* y con ello, si pudiera derivar en una mejor clasificación. Para mostrar el efecto, se parte de unos vectores de características creados a partir de un horizonte de predicción *5* y un lag *72*, los cuales podemos adelantar, y como veremos en la sección de resultados, darán los mejores resultados, a los cuales se les aplica la técnica de reducción de dimensionalidad ya presentada en el estado del arte de este trabajo, t-SNE, algoritmo que transforma el espacio de características de alta dimensión a uno de 2 en nuestro caso, preservando lo máximo posible la estructura original de los datos. De esta forma, podemos ver en la siguiente figura el efecto del *oversampling* progresivo sobre este *dataset*, aplicando en primer lugar SMOTE-EEN y posteriormente t-SNE:

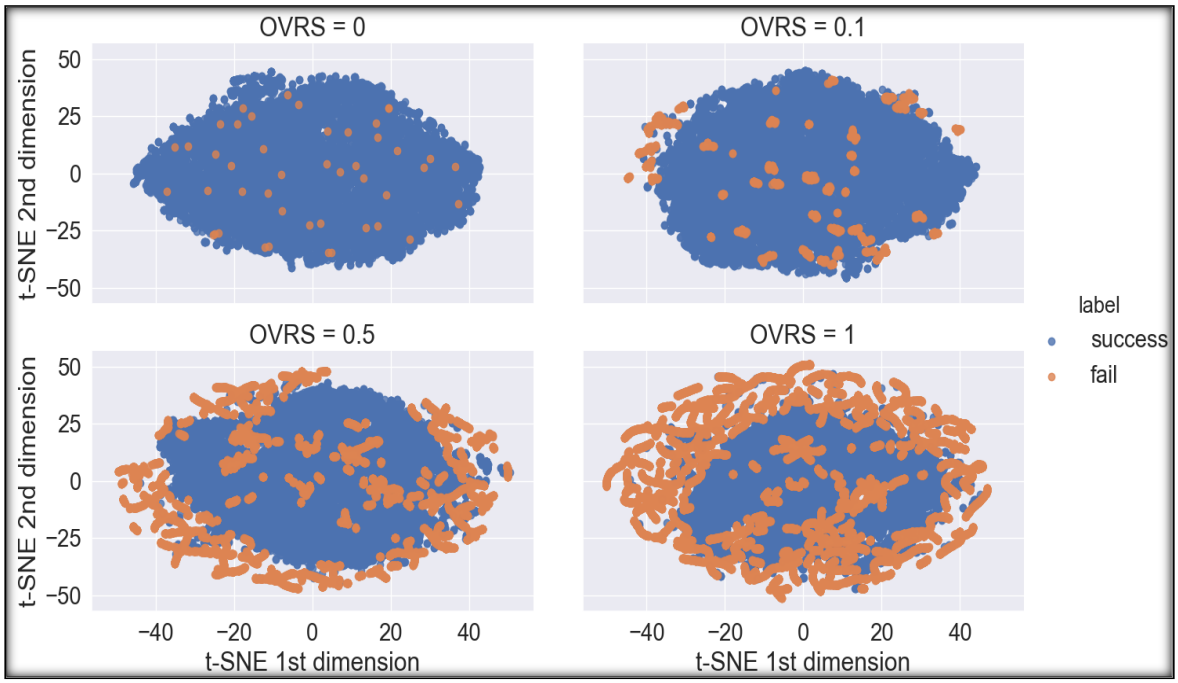

Figura 4-3: Comparativa del algoritmo t-SNE sobre un conjunto de entrenamiento, con distintos ratios de *oversampling* mediante la aplicación de SMOTE-EEN

A la vista de las gráficas obtenidas, podemos ver cómo progresivamente se va creando una estructura sintética de la clase minoritaria bien diferenciada respecto de la mayoritaria. Además, se pueden apreciar cómo las muestras de la clase minoritaria van creando dos estructuras, una entorno a las de la mayoritaria, conformándose como potenciales *outliers*, por ello creimos conveniente emplear algoritmos de detección de anomalías, y otra diferente situada en el centro de los puntos que representan éxito. Por otro lado, estas potenciales estructuras de los datos podrían indicar que en nuestro problema hay unos fallos distintos a

otros, y en consecuencia el rendimiento del algoritmo podría variar en función de ellos, de todas formas, analizaremos en profundidad esta cuestión en el apartado de resultados.

En conclusión, podemos afirmar que gracias al algoritmo SMOTE-ENN, los algoritmos de clasificación obtendrán previsiblemente mejores resultados ya que progresivamente se obtienen distintas estructuras presentes en los datos que mejorarían la separabilidad entre clases y con ello el rendimiento en la clasificación, en cualquier caso, abordaremos esta cuestión en la presentación de resultados de este trabajo.

4.3.3 Protocolo de entrenamiento y evaluación de los modelos

En los siguientes apartados presentaremos el protocolo que se ha seguido para el entrenamiento y evaluación de los distintos algoritmos empleados. En primer lugar, diferenciaremos entre las aproximaciones de la clasificación de dos clases y detección de anomalías, ya que cada una tiene diferentes particularidades, en segundo lugar, desglosaremos en cada una la descripción de los algoritmos empleados, centrándonos en los detalles relativos a los parámetros que se han probado para entrenarlos y con el objetivo de escoger el modelo que mejor se adecúe a nuestra tarea.

4.3.3.1 Clasificadores de dos clases

Como ya expusimos en el estado del arte de esta memoria, los algoritmos que se emplearán para la clasificación serán *Random Forest, AdaBoost* y *SVM.* Con ello, describiremos en este apartado el protocolo común que se ha seguido para el entrenamiento y la evaluación.

En primer lugar, el entrenamiento se realiza teniendo en cuenta las siguientes cuestiones:

- Como ya introdujimos en el estado del arte, para la búsqueda de los parámetros óptimos se emplea el algoritmo *random search*, por lo que debemos definir una distribución a priori de los parámetros, a partir de las cuales se realiza el muestreo por parte del algoritmo. Tal y como se aconsejaba en [8], se emplea 90 iteraciones.
- Previamente al entrenamiento de los modelos, se realiza una normalización de media y varianza de los datos de entrenamiento, transformación que se realizará de la misma forma a los datos de prueba.
- Se establece el número de particiones en las que el conjunto de entrenamiento se divide para tomar el conjunto de validación, con ello se realiza una validación cruzada del tipo *5-fold cross validation*, es decir, el modelo se entrena en 5 ocasiones, con un conjunto de entrenamiento y validación distinto, 4 para entrenamiento y 1 para la validación. Finalmente, la validación cruzada se combina con el ajuste de parámetros *random search*, esto es, por cada *set* de entrenamiento y validación distinto se realizan 90 entrenamientos con diferente conjunto de parámetros cada uno, con ello, en el proceso final habremos entrenado *90*5=450* modelos.
- Para la elección del mejor modelo, se ha de establecer una métrica a priori, calculada a partir del conjunto de validación, y dependiendo del tipo, se escoge el menor o mayor valor de ella. Para nuestro caso en concreto, hemos optado por optimizar la exhaustividad ponderada al peso de las clases, importante debido al desbalanceo existente en nuestro problema, conocida en la literatura también como *recall weighted.* Esta métrica viene definida en [26] como,

$$\frac{1}{\sum_{l\in L}|\hat{y}_l|}\sum_{l\in L}|\hat{y}_l|\,\phi(y_l,\hat{y}_l),$$

donde L es el conjunto de etiquetas, $\hat{y}$ la etiqueta real, y la etiqueta predicha, $|\hat{y}_l|$ el número de etiquetas reales relativas a la etiqueta l y $\phi(y_l, \hat{y}_l)$ la función que toma las etiquetas predichas y reales para calcular la exhaustividad. Finalmente, el porqué de la elección de esta métrica para tomar el mejor modelo viene dado debido a que en nuestra aplicación nos interesa recuperar todas las muestras, en mayor medida las de fallo, y no nos importa tanto no tener una precisión perfecta ya que el objetivo real del sistema es más bien detectar, al menor síntoma de fallo, el evento de fallo, y si tenemos falsos positivos, en una magnitud comedida, no resulta tan grave, ya que los agentes correspondientes, gracias a la salida de nuestro sistema, pueden monitorizar ciertas señales, comportamientos e información relativa a esa APU en concreto, las horas de funcionamiento por ejemplo, asegurándose de que es muy probable que se dé el fallo o no, partiendo de una verificación manual y conocimiento a priori del problema.

Por otro lado, en cuanto a la evaluación de los modelos se refiere, debemos tener en cuenta lo siguiente:

- Se parte del conjunto de prueba, que representa un *30%* de los datos, tal y como se explicó previamente.
- Para evaluar de forma gráfica el rendimiento, se hace uso de la curva ROC y la curva de precisión y exhaustividad de la clase minoritaria, obteniendo además las métricas AUC y AP respectivamente.
- Paralelamente, a partir de las matrices de confusión, se calcularán las matrices de métricas de ambas clases, las cuales integran la precisión, exhaustividad y el *f-score* de cada una.

Una vez explicados los protocolos para el entrenamiento y evaluación que se han seguido, desglosaremos a continuación los distintos algoritmos empleados, poniendo énfasis en los parámetros que se han tomado para el entrenamiento de ellos.

4.3.3.1.1 Random Forest

Para el entrenamiento de este modelo, se ha empleado la siguiente distribución de parámetros:

- *Profundidad*: se establece una distribución uniforme que va de 2 a 1024.
- *Número de estimadores*: idéntico a la distribución empleada para la profundidad.
- *Número de características*: se establece un vector de dos parámetros, el primero refleja la raíz cuadrada y el segundo el logaritmo en base 2, *['sqrt','log2']*.
- *Pesos de las clases*: se parte de un diccionario, estructura en *Python*, el cual posee tres pesos distintos para la clase minoritaria, mientras que el de la mayoritaria permanece igual, 0 indica la clase de éxito y 1 la de fallo, de esta manera, el diccionario queda compuesto así, *[{0:1,1:1}, {0:1,1:2}, {0:1,1:8}]*.
- *Número de componentes principales*: se establece una distribución uniforme que depende del *lag* y el número de características, 12 en nuestro caso, cuyos límites son, *[2, lag*12]*, siendo el último término la dimensión del *dataset* excluyendo la contribución de la etiqueta.

4.3.3.1.2 AdaBoost

En cuanto al modelo *AdaBoost,* la distribución de parámetros empleada es la siguiente:

- *Tasa de aprendizaje,* η: se establece un vector con los siguientes valores, *[0.1,0.25,0.5,0.6]*.
- *Número de estimadores, profundidad, número de características y componentes principales*: idénticas a la empleada para *Random Forest.*

4.3.3.1.3 SVM

Por último, para las máquinas de vector soporte, SVM, se emplea la siguiente distribución de parámetros:

- *El parámetro* C: distribución uniforme con valores entre 1 y 30.
- *El kernel*: se emplea un *rbf, radial basis function* o *kernel* gaussiano.
- *Factor gamma:* vector con valores *[0.1, 0.01, 0.001]*.
- *Pesos de las clases y número de componentes principales*: idénticas a la empleada para *Random Forest.*

4.3.3.2 Modelos de detección de anomalías

En este apartado, describiremos los detalles relativos al entrenamiento y evaluación de los dos modelos de detección de anomalías empleados, *iForests* y *one-class SVM*, particularizando cada caso.

4.3.3.2.1 iForests

Tanto los conjuntos de entrenamiento y prueba, así como el protocolo de entrenamiento son idénticos a los empleados para los algoritmos de clasificación de dos clases, sin embargo, para evaluar estos modelos debemos tener en cuenta ciertas particularidades:

- Como ya describimos en el estado del arte de esta memoria, dado un vector a la entrada del modelo, la salida consiste en el llamado *anomaly score*, mayor cuanto más 'anómala' se considere una muestra, sin embargo, debemos tener en cuenta que en la implementación de *scikit-learn* [26], se obtiene una puntuación que es inversa al *anomaly score*, por lo tanto, el sentido de interpretación es inverso al original. Además, debemos recordar que la puntuación obtenida no se trata de una probabilidad.
- Una vez calculadas las puntuaciones, para realizar la evaluación del modelo obtendremos las curvas ROC y de precisión y exhaustividad de la clase minoritaria, donde el umbral a variar, en lugar de ser una probabilidad, será un valor de *anomaly score*, a partir del cual considerar una muestra anómala o no.
- Además, se obtendrán las matrices de métricas para ambas clases, la cual indicará la precisión, exhaustividad y *f-score* de cada una.
- Por otro lado, para evaluar y estudiar la distribución de puntuaciones de ambas clases, se empleará el método *Kernel Density Estimation,* KDE [31], en lugar de la representación en un histograma, para obtener las respectivas curvas de densidades y analizar la distribución de las puntuaciones. Respecto a este último punto, si obtenemos varias modas para la distribución de puntuaciones de las muestras de

fallo, puede ser síntoma de que en nuestro problema hay fallos en concreto más detectables que otros.

Finalmente, la distribución de parámetros que hemos tenido en cuenta para el entrenamiento de este modelo ha sido:

- *Número de estimadores:* idéntico al empleado en *Random Forest* y *AdaBoost*.
- *Contaminación:* vector con valores, *[0,0.1, 0.25, 0.5]*.
- *Número de componentes principales:* idéntico al empleado en los clasificadores biclase.

4.3.3.2.2 One-class SVM

Por último, describiremos los protocolos de entrenamiento y evaluación seguidos para el algoritmo *one-class SVM*. A modo de recordatorio, el objetivo de este algoritmo era el de, partiendo únicamente de las muestras de éxito en el entrenamiento, aprender una frontera para decidir si una nueva muestra de entrada es un *outlier* o no. Con ello, nos encontramos ante un algoritmo de clasificación de tipo *hard*, es decir, no tiene en cuenta una estimación de probabilidad de pertenencia a una clase u otra, sino que toma la distancia a la frontera como una indicación del grado de confianza en la decisión, con ello, la salida es una decisión binaria rígida, *outlier* o no.

Teniendo lo anterior en cuenta, los aspectos más relevantes en cuanto al entrenamiento y evaluación de este tipo de modelos se resumen a continuación:

- Los datos de entrenamiento son idénticos a los empleados en los anteriores modelos, sin embargo, en este modelo en concreto no se tienen en cuenta los datos de fallo.
- La búsqueda de parámetros se realizará de la misma forma que en el resto de algoritmos.
- Para evaluar el rendimiento del clasificador, teniendo en cuenta que realiza una clasificación de tipo *hard*, se emplearán exclusivamente las matrices de métricas de ambas clases, indicando precisión, exhaustividad y *f-score.*

Por último, la distribución de parámetros que se ha seguido para entrenar el modelo ha sido:

- *El kernel*: se emplea un *rbf*, *radial basis function* o *kernel* gaussiano.
- *Factor gamma:* vector con valores *[0.1, 0.01, 0.001]*.
- *Número de componentes principales*: idéntica a la empleada para el resto de algoritmos.

4.4 Arquitectura del sistema propuesto

Para concluir el bloque de desarrollo, presentamos en este apartado el diagrama de nuestra propuesta para la arquitectura del sistema propuesto, el cual integra los bloques funcionales que hemos desarrollado anteriormente, especificando las dependencias entre ellos y variables más importantes.

Por último, debemos recordar que el objetivo de un futuro sistema final, si los resultados son concluyentes, es generar una alerta ante un potencial fallo en un futuro, con ello, la salida del sistema desarrollado en nuestro trabajo será un reporte de rendimiento de cierto algoritmo, el cual incluye las métricas y curvas de rendimiento ya mencionadas, por lo tanto, quedará a criterio del equipo técnico qué modelos elegir, si son convenientes en nuestro caso, y si es así, codificarlo e integrarlo finalmente en el sistema de alertas de Airbus.

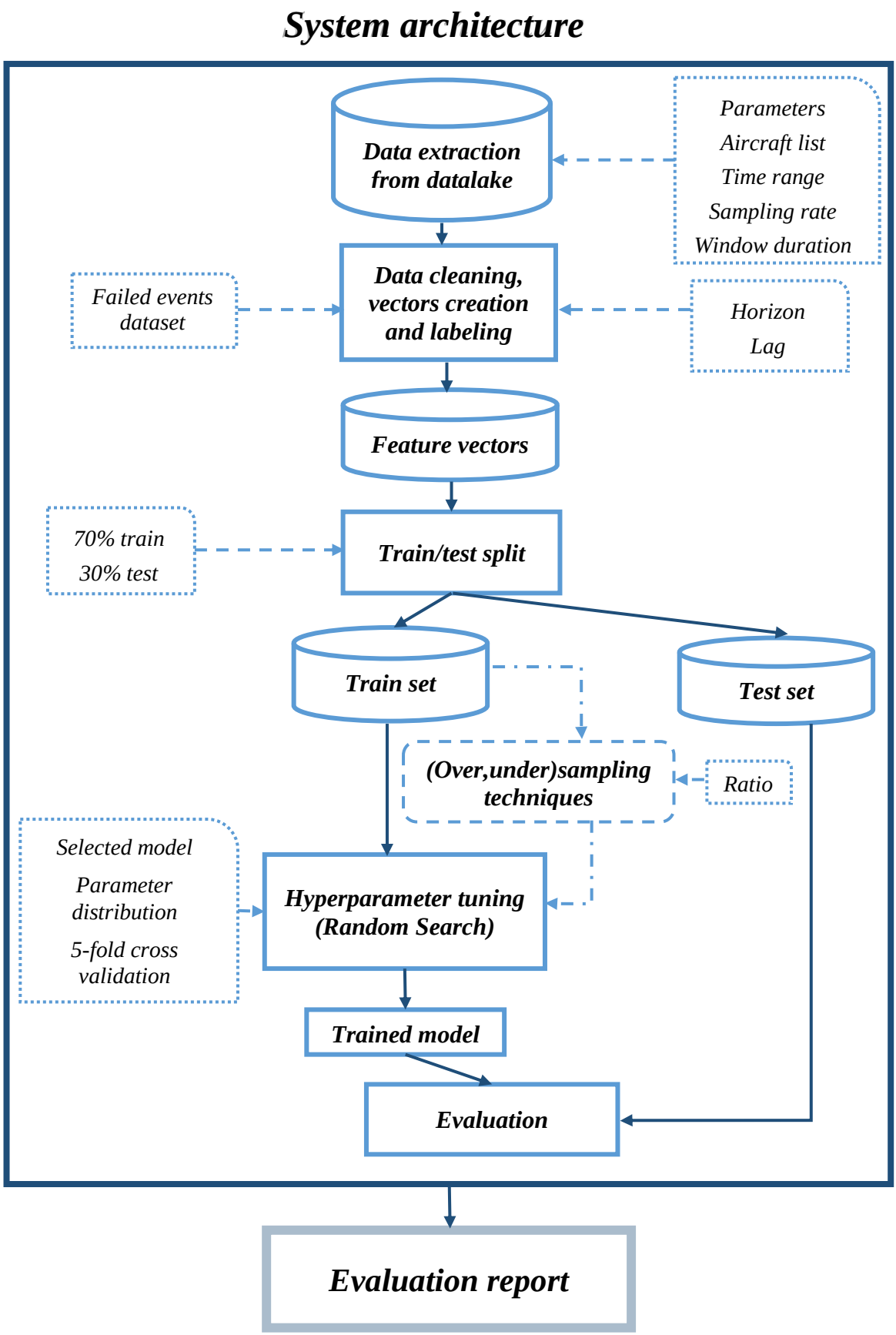

Figura 4-4: Diagrama de la arquitectura de nuestro sistema de aprendizaje automático

5 Integración, pruebas y resultados

5.1 Resultados de los modelos de clasificación

En este apartado analizaremos los resultados obtenidos para los algoritmos relativos a la clasificación de dos clases, *Random Forest*, *AdaBoost* y *SVM*.

Para la presentación de resultados, se mostrarán las métricas y gráficas ya mencionadas en el desarrollo de este trabajo. En primer lugar, por cada combinación de horizonte de predicción y *lag*, se obtendrán las curvas ROC y de precisión y exhaustividad con la variación del ratio de *oversampling* anteriormente desarrollado, presentando en este apartado las mejores, esto es, aquellos con una AP alta para la clase minoritaria y precisión y exhaustividad de la clase mayoritaria dentro de unos márgenes, ya que en esta última somos algo más permisivos con los falsos positivos, siendo el objetivo fundamental recuperar lo máximo posible el número de fallos. Gracias a ello, podremos ver y analizar si el empleo de la técnica SMOTE-EEN ayuda en la tarea de clasificación y mejora las métricas, tanto AUC, AP, como las métricas de precisión, exhaustividad y *f-score* extraídas a partir de la matriz de confusión. Finalmente, en el anexo de este trabajo se podrán encontrar todas las gráficas obtenidas para los algoritmos.

5.1.1 Random Forest

A continuación, mostramos las mejores gráficas obtenidas con *Random Forest*, donde la notación seguida en la leyenda ha sido la siguiente, {AUC ò AP; Ratio de *oversampling*}, el primer término indica la métrica correspondiente a la gráfica y el segundo el ratio de *oversampling* empleado, por otro lado, esta notación es la que se ha seguido en todas las gráficas.

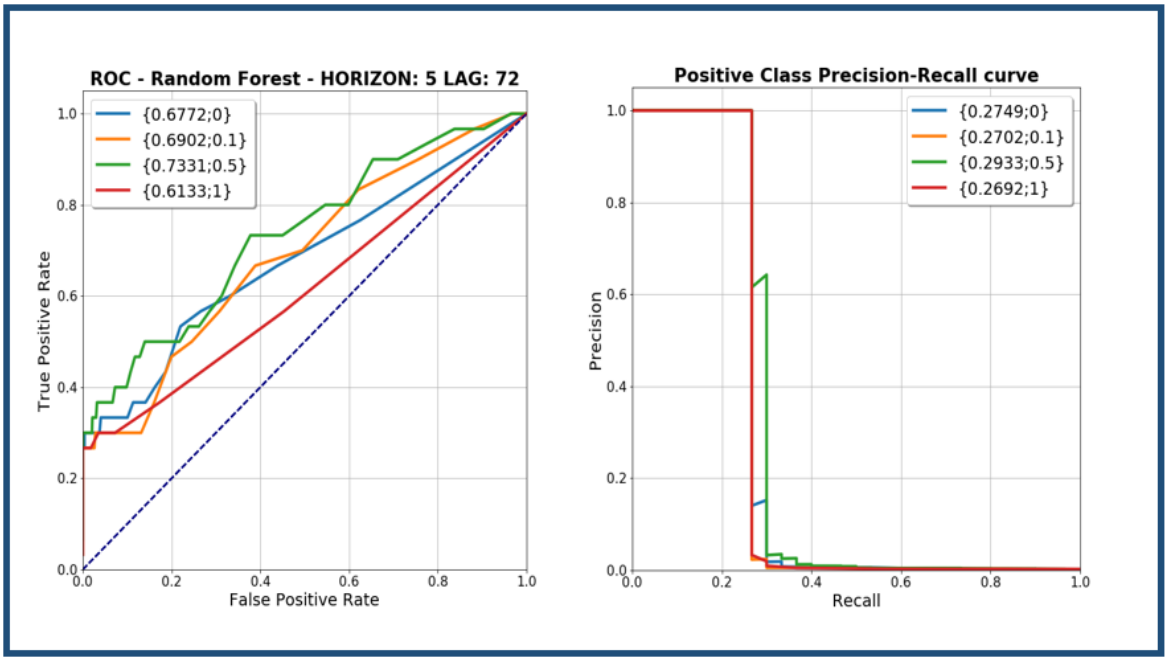

Figura 5-1: Curvas de los mejores resultados para la clasificación con *Random Forest*

A la vista de los resultados, las mejores métricas obtenidas han sido con un horizonte de predicción 5 y un *lag* 72, concretamente con un ratio de *oversampling* de 0.5, gracias al cual

conseguimos una AUC de 73.31% y AP para la clase minoritaria de 29.33%. Además, podemos observar que para ambas gráficas, el empleo del algoritmo SMOTE-EEN mejora el rendimiento general, sin embargo, si el ratio de *oversampling* es igual a 1, esto es, estamos equiparando la proporción de ambas clases, las métricas empeoran, el porqué de ello puede ser debido a que estamos creando un número de muestras sintéticas muy alto, lo que puede desvirtuar la estructura original del *dataset*, tal y como vimos en la figura 4-3 de este trabajo, que mostraba el efecto del algoritmo SMOTE-EEN en un espacio de dimensión reducida mediante t-SNE. Analizando el punto de partida de la curva ROC, es decir, el punto con tasa de falsos positivos igual a 0, vemos que las gráficas se mueven en un entorno de casi un 30% de verdaderos positivos, esto nos indicaría que somos capaces de anticiparnos 5 arranques de APU antes de la ocurrencia del fallo, casi el 30% de los casos sin ningún falso positivo, resultado que consideramos realmente positivo teniendo en cuenta el fuerte desbalanceo de clases con el que partíamos. Por otro lado, si nos fijamos en las curvas de la figura de la derecha, vemos que al llegar a cierto punto de *recall*, la precisión cae bruscamente, esto puede ser indicativo de que se dan varios tipos de fallos, en consecuencia, la capacidad de discriminación del clasificador podría verse afectada en función de ellos.

A continuación, presentamos las matrices de las mejores precisiones medias y AUC obtenidas por horizonte y *lag*:

Best Average precision per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	0.1520_{1}	$\mathbf{0.1807_{0.1}}$	0.0973_{1}
	5	0.1021_{0}	$0.2812_{0.1}$	$\mathbf{0.2933_{0.5}}$
	15	$\mathbf{0.1293_{0.1}}$	0.0468_{0}	0.0995_{0}
	30	$0.0563_{0.5}$	0.1170_{0}	$\mathbf{0.1720_{0.5}}$

Tabla 5-1: Mejor AP por Lag y Horizonte con RForest

Best Area Under Curve per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	$\mathbf{0.7219_{1}}$	$0.6818_{0.5}$	$0.6809_{0.1}$
	5	$\mathbf{0.7514_{0}}$	$0.7469_{0.1}$	$0.7331_{0.5}$
	15	$\mathbf{0.7172_{0.1}}$	0.5576_{0}	0.6696_{0}
	30	$0.6114_{0.1}$	$\mathbf{0.6872_{0.1}}$	0.5963_{0}

Tabla 5-2: Mejor AUC por Lag y Horizonte con RForest

En primer lugar, para la tabla de mejores AP, vemos que a medida que el horizonte de predicción aumenta, es necesario un mayor número de muestras, *lag*, para obtener el mejor resultado, a excepción del horizonte 15, cuyo rendimiento es mejor tomando un *lag* de 24, además, el mejor resultado se da para un horizonte 5, siendo menor en el resto. Por otro lado, vemos que las mejores métricas obtenidas han sido mediante el empleo del algoritmo SMOTE-EEN, con ratios de 0.5 y 0.1, por lo que su uso ayuda a la mejora de las métricas, además, como ya comentamos en la discusión anterior sobre las gráficas, si optamos por el empleo de un ratio de *oversampling* muy alto, corremos el riesgo de desvirtuar la estructura

original del *dataset* y con ello, obtener peores resultados. En segundo lugar, para la tabla de mejores AUC, éstas se obtienen generalmente, para todos los horizontes, con el menor *lag*, en las que hay casos donde SMOTE-EEN ayuda y otros no. Además, encontramos dos valores de AP, de los horizontes 5 y 30, con valor máximo en un *lag* de 72, por lo que sería interesante realizar otros experimentos aumentando el *lag* y ver cómo evoluciona, ya que la AP podría aumentar.

Para ver la variación de las distintas métricas de clasificación para ambas clases a medida que aumentamos el horizonte de predicción, presentamos en las siguientes tablas las mejores métricas obtenidas por horizonte, donde vemos que las relativas a la clase mayoritaria son casi perfectas, muy cercanas a 1, sin embargo, el *recall* de la clase minoritaria es bajo, siendo mayor en el horizonte de predicción igual a 5:

		Horizon 1 lag 48, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9979*	*1.00*	*0.9991*
	Fail	*1.00*	*0.18*	*0.31*

Tabla 5-3: Mejores resultados para un horizonte de predicción 1 con RForest

		Horizon 5 lag 72, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9986*	*1.00*	*0.9992*
	Fail	*1.00*	*0.27*	*0.42*

Tabla 5-4: Mejores resultados para un horizonte de predicción 5 con RForest

		Horizon 15 lag 24, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9986*	*0.9992*	*0.9988*
	Fail	*1.00*	*0.12*	*0.22*

Tabla 5-5: Mejores resultados para un horizonte de predicción 15 con RForest

		Horizon 30 lag 72, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9987*	*1.00*	*0.9993*
	Fail	*1.00*	*0.17*	*0.29*

Tabla 5-6: Mejores resultados para un horizonte de predicción 30 con RForest

Por último, para tener una representación visual de cómo la matriz de mejores AP de la clase minoritaria se comporta en función del horizonte de predicción, presentamos a continuación un mapa de calor donde las filas corresponden a los distintos horizontes de predicción empleados y las columnas los *lags*. A modo aclaratorio, la leyenda tiene su nivel máximo en 0.4, aun siendo el mayor valor de la AP de 1, éste ha sido fijado de esta manera con el objetivo de tener una mayor resolución y aumentar el rango dinámico del mapa de calor para ayudar a la visualización.

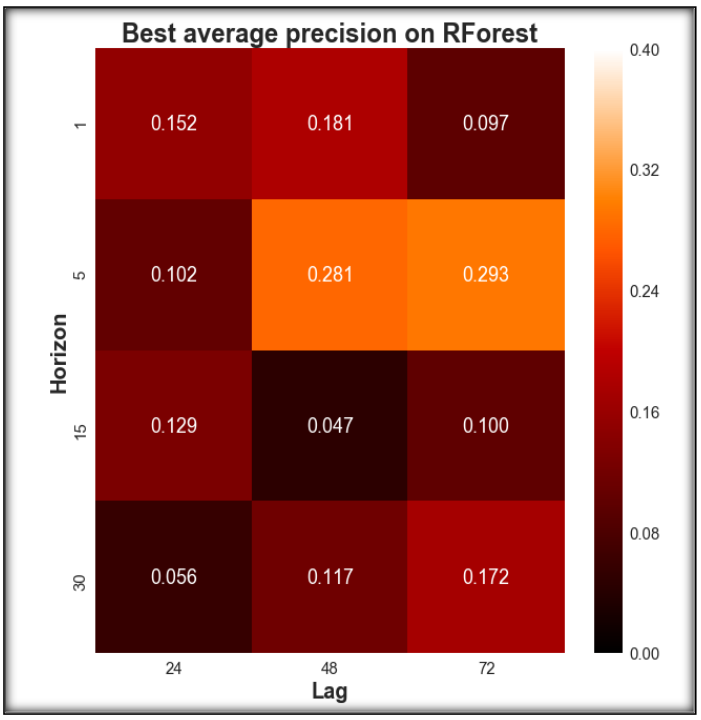

Figura 5-2: Mapa de calor de la matriz de mejores AP con *Random Forest*

5.1.2 AdaBoost

Para el algoritmo *AdaBoost*, las gráficas de rendimiento obtenidas han sido las siguientes:

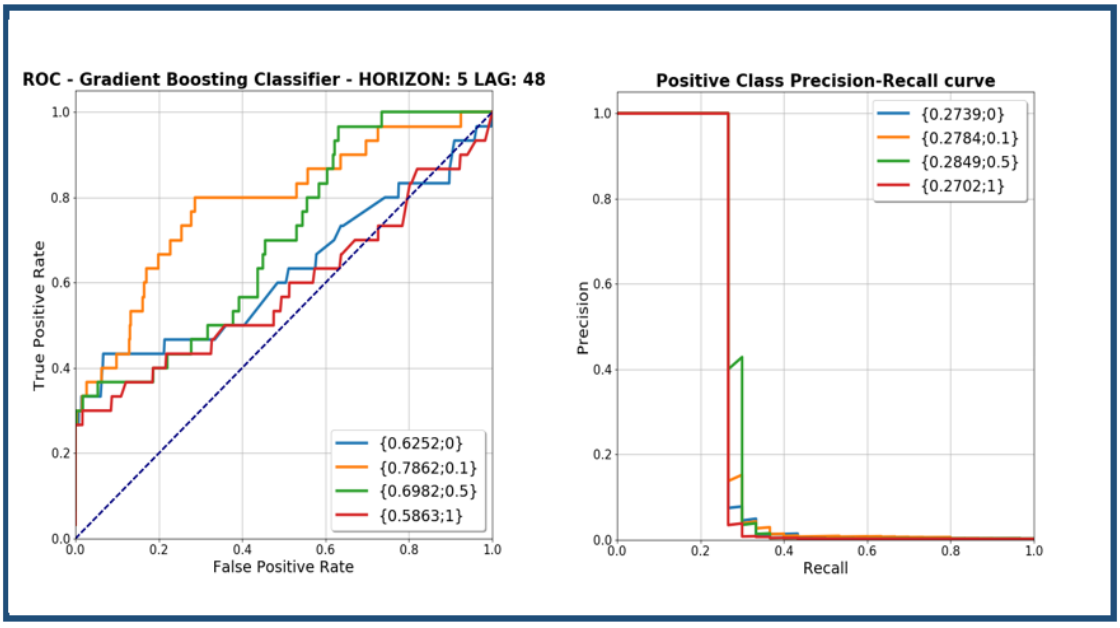

Figura 5-3: Curvas de los mejores resultados para la clasificación con *AdaBoost*

En cuanto al comportamiento de ellas se refiere, es similar a aquel obtenido para *Random Forest*, destacando que si empleamos un ratio de *oversampling* de 0.1, el rendimiento de la curva ROC correspondiente es significativamente mayor que el resto, esto puede ser debido a la penalización por una mala clasificación que realiza el algoritmo durante su entrenamiento, obteniendo así una gran capacidad de discriminación entre verdaderos y falsos positivos. Por último, el mejor resultado obtenido es para un ratio de *oversampling* de 0.5, con una AP de 28.49% y AUC igual a 69.82%, esta elección pone de manifiesto el objetivo de nuestro sistema real, ya que, a pesar de tener una curva con AUC igual a 78.62% y una AP algo menor de la escogida, se opta por sacrificar la precisión, dentro de unos márgenes, a favor de una mayor AP.

Por otro lado, presentamos a continuación las tablas con las mejores AP y AUC para cada *lag* y horizonte de predicción, donde vemos que en general el empleo de SMOTE-EEN ayuda a mejorar las métricas, en este caso, con ratios más altos que con *Random Forest*:

Best Average precision per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	$0.1637_{0.1}$	$\mathbf{0.1818_{1}}$	$0.1086_{0.5}$
	5	0.0977_{0}	$\mathbf{0.2849_{0.5}}$	$0.2733_{0.5}$
	15	$\mathbf{0.1321_{0}}$	0.0441_{1}	0.0919_{1}
	30	$0.0579_{0.1}$	$0.1215_{0.1}$	$\mathbf{0.1712_{1}}$

Tabla 5-7: Mejor AP por Lag y Horizonte con AdaBoost

Best Area Under Curve per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	$\mathbf{0.7073_{0.1}}$	$0.6164_{0.5}$	$0.6896_{0.1}$
	5	$0.5758_{0.1}$	$\mathbf{0.7862_{0.1}}$	$0.6873_{0.1}$
	15	$\mathbf{0.6492_{0.1}}$	0.5533_{1}	0.5759_{1}
	30	$0.5704_{0.1}$	$0.5331_{0.5}$	$\mathbf{0.6421_{1}}$

Tabla 5-8: Mejor AUC por Lag y Horizonte con AdaBoost

A continuación, mostramos las tablas de las mejores métricas obtenidas por horizonte de predicción, donde el mejor se ha obtenido con un horizonte igual a 5:

		Horizon 1 lag 48, OVRS=1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9979*	*1.00*	*0.9991*
	Fail	*1.00*	*0.18*	*0.30*

Tabla 5-9: Mejores resultados para un horizonte de predicción 1 con AdaBoost

		Horizon 5 lag 48, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9986*	*1.00*	*0.9992*
	Fail	*1.00*	*0.27*	*0.42*

Tabla 5-10: Mejores resultados para un horizonte de predicción 5 con AdaBoost

		Horizon 15 lag 24, OVRS=0		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9982*	*1*	*0.9991*
	Fail	*1*	*0.12*	*0.21*

Tabla 5-11: Mejores resultados para un horizonte de predicción 15 con AdaBoost

		Horizon 30 lag 72, OVRS=1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9983*	*1.00*	*0.9991*
	Fail	*1.00*	*0.20*	*0.3*

Tabla 5-12: Mejores resultados para un horizonte de predicción 30 con AdaBoost

Por último, presentamos el mapa de calor obtenido a partir de la matriz de las mejores AP:

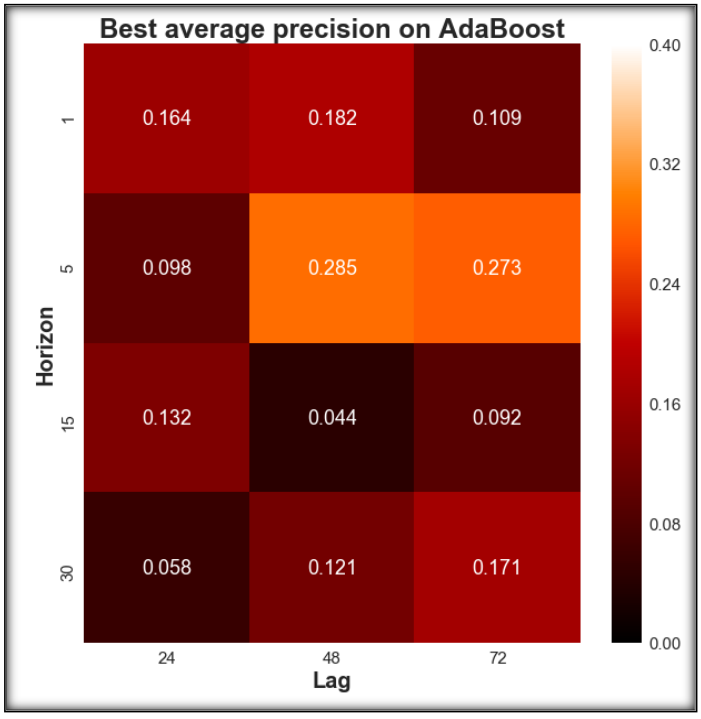

Figura 5-4: Mapa de calor de la matriz de mejores AP con *AdaBoost*

5.1.3 SVM

En cuanto a las mejores gráficas obtenidas con el algoritmo SVM, presentamos las siguientes:

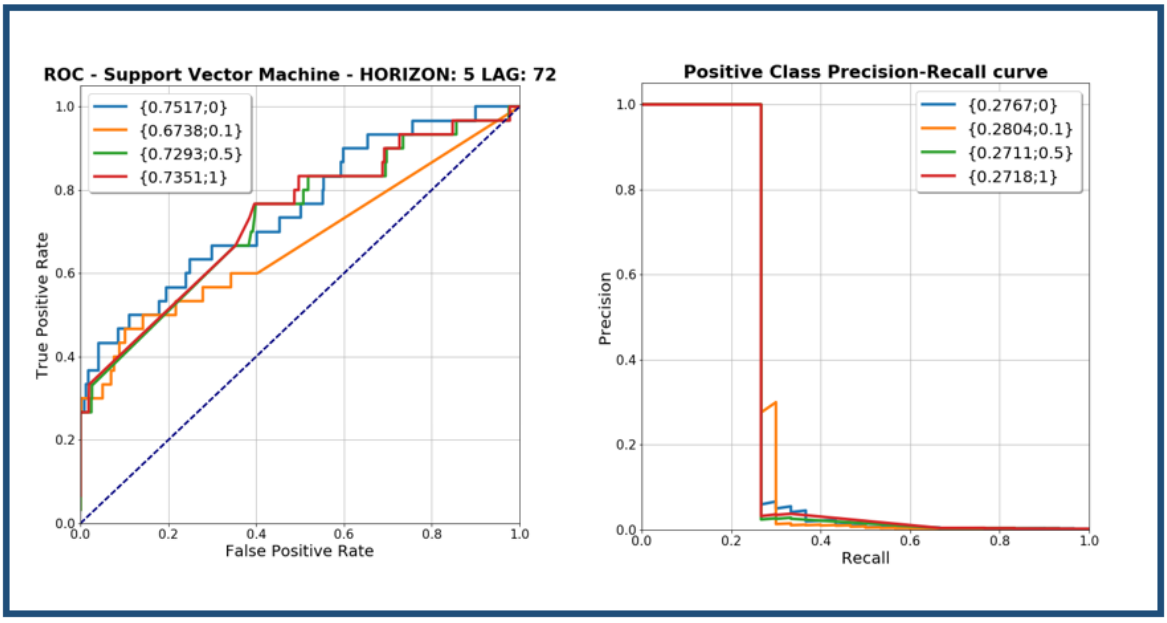

Figura 5-5: Curvas de los mejores resultados para la clasificación con SVM

Donde se vuelve a obtener los mejores resultados para un horizonte de predicción igual a 5, con curvas similares a los dos algoritmos anteriores, en este caso, el mejor ratio de *oversampling* ha sido 0.1, sin embargo, en cuanto a la mejor AUC se refiere, ésta se consigue sin aplicar SMOTE-EEN. Por otro lado, el comportamiento de la curva de precisión y exhaustividad es similar a las anteriores, donde a partir de cierto valor de *recall* se produce una caída abrupta.

Una vez mostradas las gráficas, procedemos a continuación a analizar las tablas que condensan las mejores AP y AUC:

Best Average precision per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	0.1566_{0}	$\mathbf{0.1847_{0.1}}$	0.0977_{0}
	5	0.1049_{0}	0.2758_{1}	$\mathbf{0.2804_{0.1}}$
	15	$\mathbf{0.1349_{0.1}}$	0.0450_{0}	$0.1348_{0.5}$
	30	$0.0558_{0.5}$	$0.1236_{0.1}$	$\mathbf{0.2392_{0.1}}$

Tabla 5-13: Mejor AP por Lag y Horizonte con SVM

Best Area Under Curve per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	$\mathbf{0.7365_{0.1}}$	$0.7336_{0.5}$	0.6330_{0}
	5	0.7202_{0}	0.7050_{0}	$\mathbf{0.7517_{0}}$
	15	$\mathbf{0.7095_{0.5}}$	0.5938_{0}	$0.6799_{0.1}$
	30	0.5467_{0}	0.6815_{1}	$\mathbf{0.7134_{0}}$

Tabla 5-14: Mejor AUC por Lag y Horizonte con SVM

El comportamiento de ambas es similar a los resultados anteriores, donde el algoritmo SMOTE-EEN ayuda, en general, a obtener mejores resultados. En este algoritmo particularmente se obtienen resultados ligeramente superiores respecto a los anteriores para un horizonte de predicción igual a 30, donde llegamos a tener una AP de 23,92%, por lo que el empleo de máquinas de vector soporte en horizontes de predicción mayores puede ser adecuado. Además, tal y como ocurría en *Random Forest*, encontramos dos valores de AP, de los horizontes 5 y 30, con valor máximo en un *lag* de 72, por lo que sería interesante aumentar el *lag* y ver si conseguimos mejores AP.

Por otro lado, en cuanto a las mejores métricas obtenidas por horizonte de predicción, tenemos las siguientes, además del mapa de calor con las mejores AP:

		Horizon 1 lag 48, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9979*	*1.00*	*0.9991*
	Fail	*1.00*	*0.18*	*0.30*

Tabla 5-15: Mejores resultados para un horizonte de predicción 1 con SVM

		Horizon 5 lag 72, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9982*	*1.00*	*0.9991*
	Fail	*1.00*	*0.27*	*0.42*

Tabla 5-16: Mejores resultados para un horizonte de predicción 5 con SVM

		Horizon 15 lag 24, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9982*	*1*	*0.9991*
	Fail	*1*	*0.12*	*0.21*

Tabla 5-17: Resultados para un horizonte de predicción 15 con SVM

		Horizon 30 lag 72, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success**	*0.9984*	*1.00*	*0.9992*
	Fail	*1.00*	*0.22*	*0.36*

Tabla 5-18: Resultados para un horizonte de predicción 30 con SVM

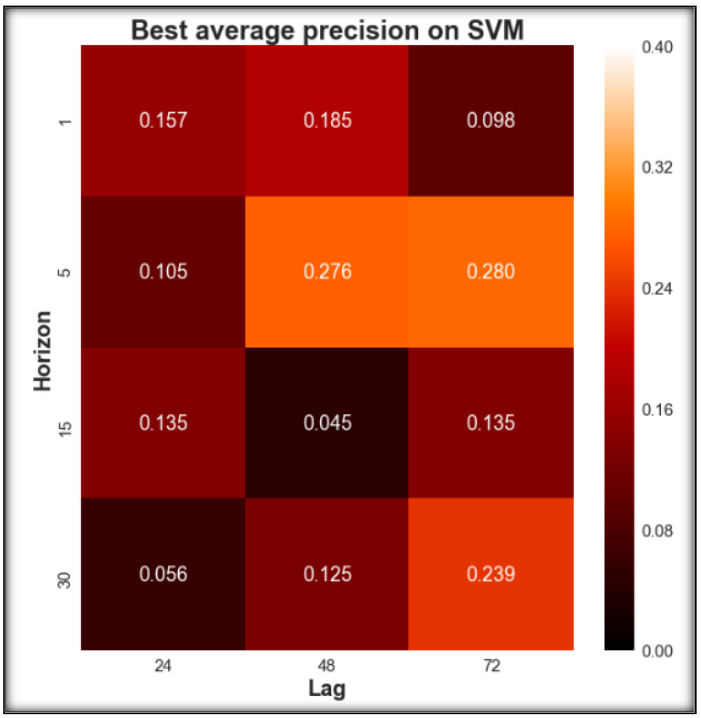

Figura 5-6: Mapa de calor de la matriz de mejores AP con *SVM*

5.2 Resultados de los modelos de detección de anomalías

En este apartado, igual que hemos hecho para los algoritmos de clasificación de dos clases, analizaremos el desempeño de los algoritmos de detección de anomalías, *iForests* y *one-class SVM*, justificando si son adecuados para el problema que estamos abordando o no. Además, debemos recordar las particularidades respecto a la evaluación de estos modelos, ya que difiere en cierta medida de la abordada para los modelos de clasificación de dos clases.

5.2.1 Isolation Forests

Para la evaluación de los *iForests*, debemos recordar que la salida del sistema de aprendizaje automático es una puntuación o *score* de cuán anómalo se considera un vector de entrada. Por ello, presentamos a continuación las curvas ROC y de precisión y exhaustividad obtenidas de la mejor combinación de horizonte y *lag*, además, mostramos las distribuciones de scores obtenidas con un ratio de *oversampling* de 0 y 1, con el fin de analizar cómo SMOTE-EEN afecta a la distribución de puntuaciones:

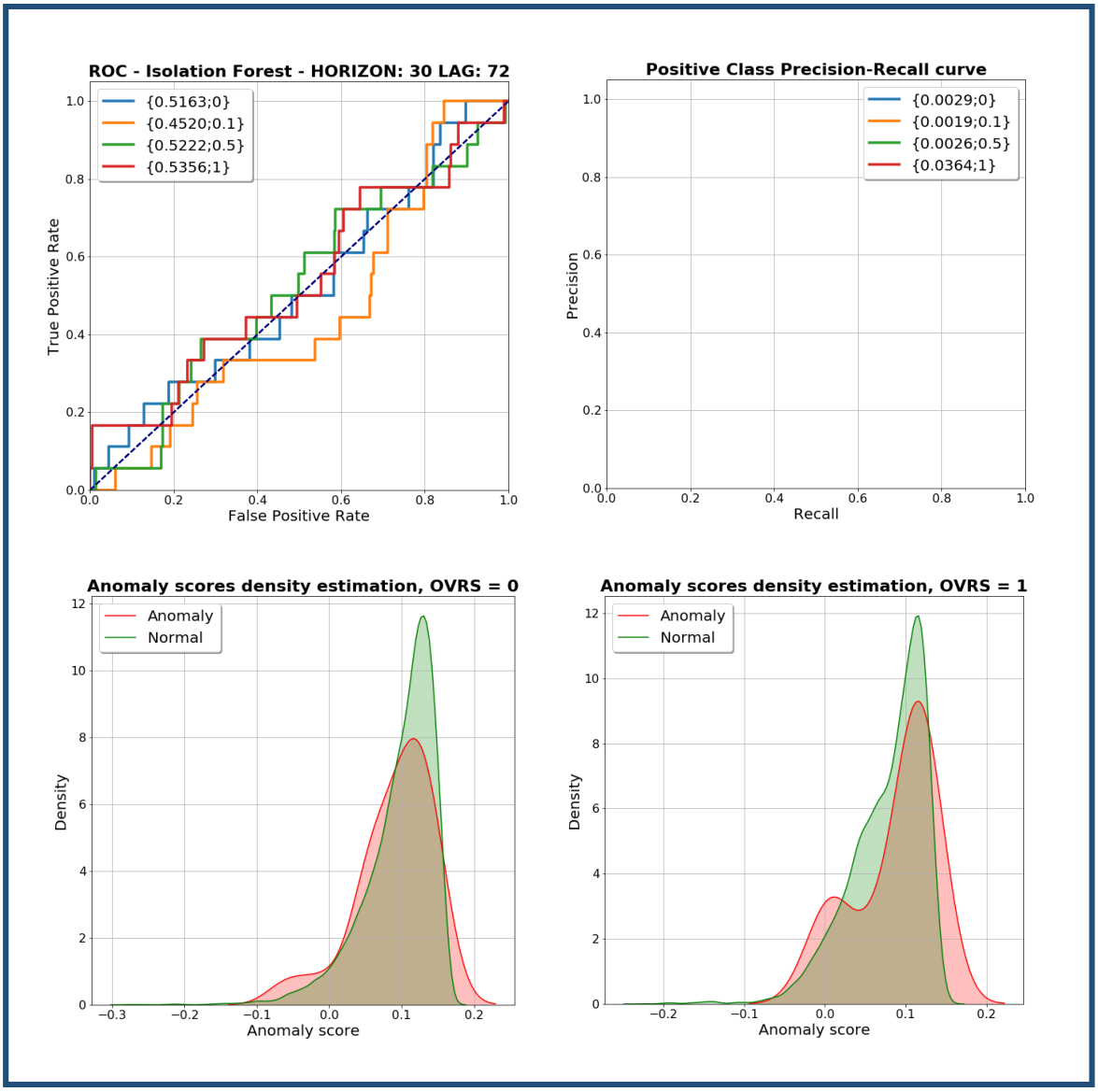

Figura 5-7: Curvas de los mejores resultados para la detección de anomalías con *iForests*

A la vista de los resultados obtenidos, podemos afirmar que el rendimiento del algoritmo no es apropiado, ya que se comporta básicamente como un clasificador azaroso, que decide indistintamente entre ambas clases, anomalía y realización normal, el rendimiento se puede deducir que no será adecuado debido a que la distribución de puntuaciones de anomalías y normales están muy juntas entre sí, lo que dificulta la discriminación para un valor de umbral dado. Sin embargo, si nos fijamos en la distribución de puntuaciones, las cuales se han obtenido mediante el método *Kernel Density Estimation*, KDE [30] y están centradas en 0,

vemos dos modas bien diferenciadas para la distribución de puntuaciones de fallo, las cuales se acentúan más aún conforme el ratio de *oversampling* va aumentando, respecto a las realizaciones normales del APU, sus puntuaciones siguen prácticamente una distribución monomodal. Por otro lado, este comportamiento es general para todos los casos analizados, por lo que podría ser indicativo de que en nuestro problema podríamos tener varios tipos de fallos, además, debido a que una de las dos modas está más escorada hacia la izquierda, un clasificador clasificaría mejor un tipo de fallos que otros, cuestión que sucedía cuando analizábamos las curvas de precisión y exhaustividad en los algoritmos de clasificación de dos clases, donde había una caída abrupta de precisión, y también en el apartado de análisis mediante t-SNE con distintos ratios de *oversampling*, donde veíamos que se iban configurando progresivamente distintas estructuras de la clase minoritaria bastante diferenciadas. De todas maneras, esta cuestión debemos estudiarla más en profundidad y ver si nuestras hipótesis tienen relación entre sí, aparte de ello, el uso de *iForests* puede ser interesante para diagnosticar un sistema y ver si se dan distintos modos de fallo. Teniendo en cuenta todo lo anterior, presentamos los resultados de mejor AP y AUC:

Best Average precision per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	$0.0019_{0.5}$	$\mathbf{0.0026_{0.1}}$	$0.0021_{0.1}$
	5	$0.0017_{0.5}$	$\mathbf{0.0024_{0}}$	$0.0022_{0.5}$
	15	$0.0019_{0.5}$	$0.0024_{0.5}$	$\mathbf{0.0028_{1}}$
	30	0.0030_{1}	0.0089_{1}	$\mathbf{0.0364_{1}}$

Tabla 5-19: Mejor AP por Lag y Horizonte con *iForests*

Best Area Under Curve per Lag and Horizon

		Lag		
		24	48	72
Horizon	1	0.4137_{0}	$\mathbf{0.5051_{0.1}}$	0.4536_{1}
	5	$0.3407_{0.5}$	$\mathbf{0.4312_{0.1}}$	0.4007_{1}
	15	$0.3718_{0.5}$	$0.4501_{0.5}$	$\mathbf{0.5131_{1}}$
	30	0.4815_{0}	$\mathbf{0.5784_{1}}$	0.5356_{1}

Tabla 5-20: Mejor AUC por Lag y Horizonte con *iForests*

Por último, presentamos las tablas con las métricas para cada horizonte de predicción y un mapa de calor con las mejores AP, donde se da un buen rendimiento en la clase mayoritaria y uno deficiente en la minoritaria:

		Horizon 1 lag 48, OVRS=0.1		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*0.9999*	*0.9975*	*0.9987*
	Fail/Anomaly	*0.00*	*0.00*	*0.00*

Tabla 5-21: Mejores resultados para un horizonte de predicción 1 con *iForests*

		Horizon 5 lag 48, OVRS=0		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*0.9977*	*0.9996*	*0.9986*
	Fail/Anomaly	*0.00*	*0.00*	*0.00*

Tabla 5-22: Mejores resultados para un horizonte de predicción 5 con *iForests*

		Horizon 15 lag 72, OVRS=1		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*0.9979*	*0.9991*	*0.9986*
	Fail/Anomaly	*0.00*	*0.00*	*0.00*

Tabla 5-23: Mejores resultados para un horizonte de predicción 15 con *iForests*

		Horizon 30 lag 48, OVRS=1		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*0.9979*	*0.9993*	*0.9986*
	Fail/Anomaly	*0.00*	*0.00*	*0.00*

Tabla 5-24: Mejores resultados para un horizonte de predicción 30 con *iForests*

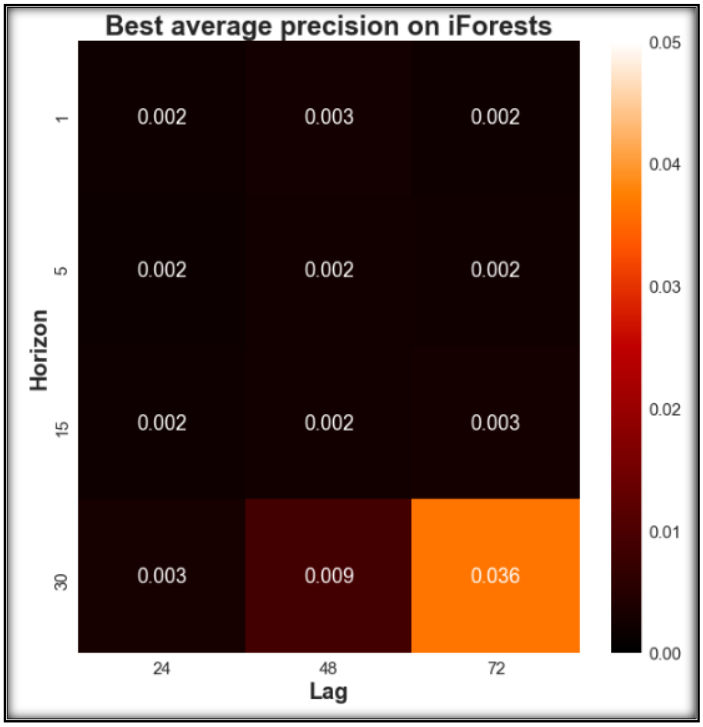

Figura 5-8: Mapa de calor de la matriz de mejores AP con *iForests*

5.2.2 One-class SVM

Por último, analizaremos el algoritmo de detección de anomalías *one-class SVM*, donde las mejores métricas obtenidas por horizonte de predicción han sido las siguientes:

		Horizon 5 lag 24, OVRS=1		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*1.00*	*0.86*	*0.92*
	Fail/Anomaly	*0.01*	*0.44*	*0.01*

Tabla 5-25: Mejores resultados para one-class SVM

		Horizon 1 lag 48, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*1.00*	*0.86*	*0.92*
	Fail/Anomaly	*0.01*	*0.35*	*0.01*

Tabla 5-26: Mejores resultados para un horizonte de predicción 1 con *one-class SVM*

		Horizon 15 lag 24, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*1.00*	*0.72*	*0.83*
	Fail/Anomaly	*0.00*	*0.42*	*0.01*

Tabla 5-27: Mejores resultados para un horizonte de predicción 15 con *one-class SVM*

		Horizon 30 lag 72, OVRS=0.5		
		Precision	**Recall**	**F_1-score**
Class	**Success/Normal**	*1.00*	*0.89*	*0.94*
	Fail/Anomaly	*0.00*	*0.22*	*0.01*

Tabla 5-28: Mejores resultados para un horizonte de predicción 30 con *one-class SVM*

A la vista de los resultados obtenidos, el rendimiento general con este algoritmo es mayor que para *iForests*, donde tenemos un mayor *recall* para la clase minoritaria pero una precisión deficiente para ella, con lo que el empleo de este algoritmo en nuestro problema no es adecuado, además, debemos recordar que al tratarse de un algoritmo cuya decisión es de tipo *hard*, éste no es tan flexible en la clasificación.

5.3 Análisis global de resultados y elección de mejores modelos

En este último apartado, analizaremos de manera global el rendimiento y los resultados de los algoritmos empleados, con el fin de mostrar aquellos con mejores resultados. Con ello, se presenta al equipo técnico, aparte de la totalidad de esta memoria, el rendimiento de los mejores algoritmos desglosados por horizonte de predicción, esto es, en función de cuánto nos queramos anticipar a un fallo, los resultados van a estar determinados por una serie de

métricas y curvas, por lo tanto, dependiendo de las necesidades del problema y el compromiso que se quiera tener en cuanto a la anticipación del fallo se refiere, se procede de la siguiente forma:

- En primer lugar, debemos tener en cuenta las necesidades del problema y si los resultados obtenidos son concluyentes o no. Por ello, en cuanto a la clase minoritaria se refiere, se busca un *recall* idealmente elevado, donde la precisión puede variar entre unos límites fijados a priori, por otro lado, para la clase mayoritaria, se buscará tener una precisión y exhaustividad adecuados, donde podrán encontrarse dentro de unos límites al igual que la precisión para la clase minoritaria.
- En función de las necesidades de anticipación, debido a que presentamos las mejores gráficas y métricas por horizonte de predicción, el trabajo del equipo técnico será elegir un comportamiento determinado, teniendo en cuenta las diferencias y similitudes entre cómo se presentan los resultados dependiendo del modelo de aprendizaje automático.
- Para los algoritmos cuyo rendimiento se exprese de manera gráfica, todos los de este trabajo a excepción de *one-class SVM*, partiendo de la curva de precisión y exhaustividad, se debe elegir un punto de ella, determinado por un umbral en la decisión, que marcará la precisión y exhaustividad para la clase minoritaria directamente y de la mayoritaria indirectamente.

Teniendo en cuenta todo lo anterior, presentamos a continuación las mejores curvas de precisión y exhaustividad de la clase minoritaria desglosadas por horizonte de predicción:

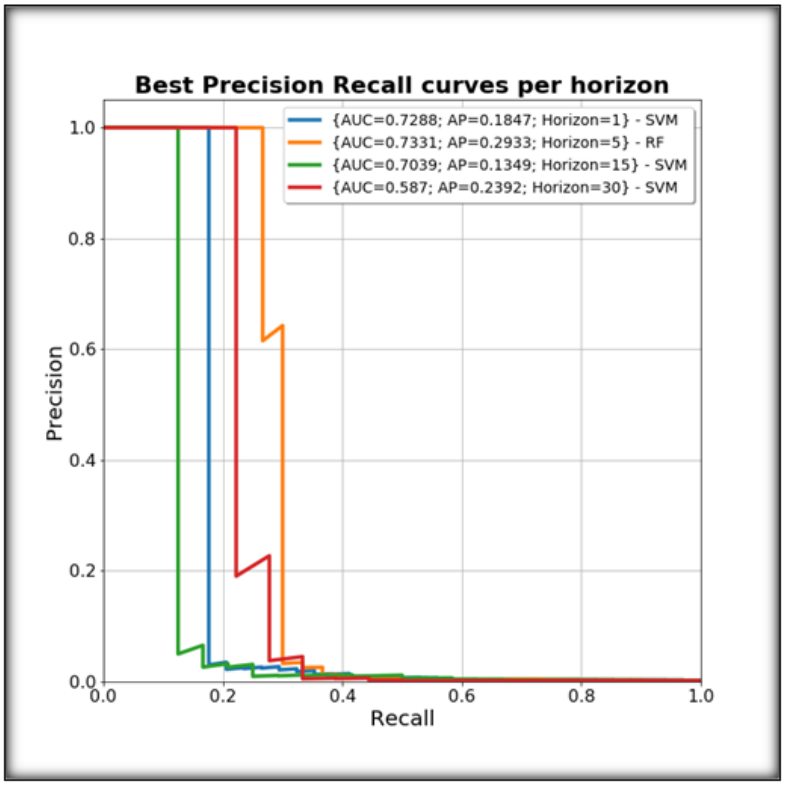

Figura 5-9: Mejores curvas de la clase minoritaria por horizonte de predicción

A la vista de los resultados obtenidos, el algoritmo SVM obtiene los mejores resultados para tres cuartas partes de los horizontes de predicción analizados, siendo *Random Forest* el que reporta el mejor, previsiblemente, y tal como enunciamos en el estado del arte, debido a su buen rendimiento en *datasets* de diversa índole. Además, según va aumentando el horizonte de predicción, la capacidad de discriminación entre falsos positivos y verdaderos positivos, reflejado en la AUC, disminuye, pasando de un máximo de 73.31% con un horizonte 5 a 58.70% con un horizonte 30. En cuanto a la precisión media, AP, alcanza su máximo para un horizonte 5, siendo la menor para el 15, por otra parte, en cuanto al comportamiento de estas curvas se refiere, es similar para todos los horizontes, en las que en un determinado punto se produce una caída brusca de la precisión, por ello, es posible que nos encontremos ante fallos en los que el desempeño del algoritmo es adecuado mientras en otros no. En relación con los puntos previos a la presentación de la gráfica, podemos observar que, si avanzamos en la gráfica en busca de un mayor *recall*, llegamos a un punto en el que la precisión cae abruptamente, viéndose bastante afectada, en dos etapas, donde se produce un pequeño aumento de *recall*.

En conclusión, a pesar de no obtener un *recall* demasiado alto en general, podemos destacar, teniendo en cuenta el desbalanceo de clases tan acusado, que se consigue una precisión de 100% con un *recall* cercano al 30%, esto es, para el mejor caso con 5 arranques de APU de anticipación, podríamos predecir sin falsos positivos un tercio de los fallos, lo cual consideramos bastante prometedor.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Con la realización de este trabajo hemos empleado técnicas de aprendizaje automático para la resolución de un problema real al que se enfrenta el departamento, las cuales no habían sido exploradas previamente. De esta manera, se han presentado los resultados obtenidos con las diferentes aproximaciones seguidas, así como los fundamentos matemáticos en las que se basan, por ello, este trabajo se propone a modo de guía o marco de actuación en caso de que se quiera abordar otro problema similar de anticipación de fallo, ya que hemos analizado uno específico que se da en el APU, sin embargo, los razonamientos, fundamentos matemáticos y herramientas se pueden extender a otros problemas similares, fallos por ejemplo en una válvula o cualquier sistema que sea objeto de experimentar un fallo en un futuro.

En cuanto a los conocimientos técnicos adquiridos se refiere, hemos aprendido a entender y tratar el funcionamiento y señales involucradas en un sistema aeronáutico real, el APU, siendo estos conocimientos necesarios a priori, con el fin de abordar una aproximación basada en el desarrollo de algoritmos de aprendizaje automático, en relación con esto último, se han propuesto algoritmos empleados en el estado del arte, siguiendo dos aproximaciones, la primera mediante una clasificación en dos clases y la otra mediante la detección de anomalías, siendo el rendimiento de esta última deficiente. Además, hemos aprendido a trabajar con *Skywise* y las distintas herramientas de las que se compone, siendo éste un entorno de computación distribuida que maneja una cantidad considerable de datos procedentes de distintos aviones.

Por otro lado, en lo que a los resultados y experimentos se refiere, podemos concluir que si nos enfrentamos ante un problema de desbalanceo de clases, las técnicas de *oversampling* y *downsampling* combinadas pueden ayudar en la clasificación, en nuestro caso SMOTE-EEN, gracias a la cual hemos conseguido mejorar el rendimiento general de los algoritmos. En relación con los resultados obtenidos, hemos presentado el mejor rendimiento para los horizontes de predicción analizados, los cuales el equipo analizará y estudiará su viabilidad, en cualquier caso, y teniendo en cuenta de que se trata de una primera aproximación, los resultados obtenidos nos animan a seguir trabajando en esta tarea, ya que los consideramos realmente positivos a pesar de ser un primer prototipo, debido a que en el mejor caso conseguimos clasificar casi el 30% de los fallos sin ningún falso positivo. Por último, nos gustaría remarcar que el uso del algoritmo t-SNE para la visualización de *datasets* puede ayudar a entender la forma y estructura en la que se presentan nuestros datos y en consecuencia, elegir técnicas y algoritmos que mejor se adecúen a ellos.

Finalmente, nos gustaría destacar la colaboración que ha habido entre el mundo empresarial y universitario, la cual consideramos realmente enriquecedora en nuestro caso concreto. Tal y como expusimos en la introducción de este trabajo, hemos contado con la colaboración, en primer lugar, desde el punto de vista de la empresa, de Alberto Martínez Sancho, conocedor del mundo aeronáutico, dedicado a tareas relacionadas con mantenimiento predictivo y *health monitoring*, y en segundo lugar de Doroteo Torre Toledano, investigador especializado en la aplicación de técnicas de *machine learning* para audio y voz, en consecuencia, consideramos la confluencia de estos dos puntos de vista y perspectivas realmente positiva.

6.2 Trabajo futuro

En cuanto al trabajo futuro se refiere, proponemos los siguientes puntos o tareas:

- Profundizar en el estudio de los resultados obtenidos, concretamente en la caída tan abrupta, a partir de cierto valor de *recall*, de la precisión, que podría ser indicativo que nuestros algoritmos clasifican mejor unos fallos que otros.
- Presentar los resultados al equipo de científicos de datos de Toulouse, con el fin de compartir valoraciones y proponer nuevas aproximaciones.
- Introducir nuevos ejemplos de fallo a medida que se vaya actualizando la BBDD de eventos.
- Explorar técnicas basadas en aprendizaje profundo, con redes neuronales de tipo LSTM o CNN, las cuales han dado excelentes resultados en los últimos años, y para nuestro caso, en el tratamiento y predicción en series temporales.
- Estudiar la viabilidad de la aplicación de técnicas de reconocimiento de patrones de señales de audio, por ejemplo, de grabaciones acústicas recogidas en la cavidad donde se aloja el APU, las cuales puedan ayudar a predecir y anticipar fallos.

Referencias

[1] *Información de la plataforma Skywise de la página web oficial de Airbus: https://www.airbus.com/aircraft/support-services/skywise.html, fuente consultada el 26/10/2018*

[2] *Alestra, S., Brand, C., Burnaev, E., Erofeev, P., Papanov, A., Bordry, C., & Silveira-Freixo, C. (2014, July). Rare event anticipation and degradation trending for aircraft predictive maintenance. In 11th World Congress on Computational Mechanics, WCCM (pp. 6571-6582).*

[3] *Documentación interna de la empresa: Aircraft Maintenance Manual (A380 AMM), PW980 Training Manual.*

[4] *Heise, P., Gaillardet, I., Rahman, H., & Mannur, V. (2015). Avionics Full Duplex Ethernet and the Time Sensitive Networking Standard.*

[5] *Smith, L. I. (2002). A tutorial on principal components analysis.*

[6] *Abdi, H., & Williams, L. J. (2010). Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4), 433-459.*

[7] *Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.*

[8] *Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.*

[9] *Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.*

[10] *Wang, J., Xu, M., Wang, H., & Zhang, J. (2006). Classification of imbalanced data by using the SMOTE algorithm and locally linear embedding. In Signal Processing, 2006 8th International Conference on (Vol. 3). IEEE.*

[11] *More, A. (2016). Survey of resampling techniques for improving classification performance in unbalanced datasets. arXiv preprint arXiv:1608.06048.*

[12] *Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.*

[13] *Freund, Y. Schapire, R., & Abe, N. (1999). A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence, 14(771-780), 1612.*

[14] *LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S. ... & Simard, P. (1995, October). Comparison of learning algorithms for handwritten digit recognition. In International conference on artificial neural networks (Vol. 60, pp. 53-60).*

[15] *Documentación sobre algoritmos de detección de anomalías disponibles en la librería scikit-learn: http://scikit-learn.org/stable/modules/outlier_detection.html , fuente consultada el 06/09/2018*

[16] *Representación del rendimiento de distintos algoritmos de detección de anomalías en scikit-learn:: http://scikit-learn.org/stable/auto_examples/plot_anomaly_comparison.html#sphx-glr-auto-examples-plot-anomaly-comparison-py , fuente consultada el 10/09/2018*

[17] *Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE.*

[18] *Li, K. L., Huang, H. K., Tian, S. F., & Xu, W. (2003, November). Improving one-class SVM for anomaly detection. In Machine Learning and Cybernetics, 2003 International Conference on(Vol. 5, pp. 3077-3081). IEEE.*

[19] *Información de Palantir y Skywise: http://www.palantir.com/solutions/skywise/ , fuente consultada el 06/10/2018*

[20] *Información y documentación de Spark: https://spark.apache.org/, fuente consultada el 03/09/2018*

[21] *Información sobre datalakes de Amazon: https://aws.amazon.com/es/big-data/datalakes-and-analytics/what-is-a-data-lake/ , fuente consultada el 03/11/2018*

[22] *Librería en javascript para la manipulación y visualización de datos: https://d3js.org/,fuente consultada el 12/10/2018*

[23] *Información oficial del A380: https://www.airbus.com/aircraft/passenger-aircraft/a380-family.html, fuente consultada el 07/11/2018*

[24] *Plataforma Anaconda: https://www.anaconda.com/, fuente consultada el 28/10/2018*

[25] *Información sobre la herramienta Jupyter: http://jupyter.org/, fuente consultada el 01/11/2018*

[26] *Documentación de la librería scikit-learn: http://scikit-learn.org/stable/, fuente consultada el 04/09/2018*

[27] Documentación de la librería matplotlib: https://matplotlib.org/, fuente consultada el 15/09/2018

[28] Documentación de la librería seaborn: https://seaborn.pydata.org/, fuente consultada el 15/09/2018

[29] Repositorio github de la librería imbalanced-learn: https://github.com/scikit-learn-contrib/imbalanced-learn, fuente consultada el 02/09/2018

[30] Repositorio github de la librería con la implementación en varios núcleos de t-SNE: https://github.com/DmitryUlyanov/Multicore-TSNE, fuente consultada el 13/10/2018

[31] Kim, J., & Scott, C. D. (2012). Robust kernel density estimation. Journal of Machine Learning Research, 13(Sep), 2529-2565.

Glosario

IoT	Internet of Things
API	Application Programming Interface
APU	Auxiliary Power Unit
ECB	Electronic Control Box
QoS	Quality of Service
ROC	Receiving Operating Characteristic
AUC	Area Under Curve
AP	Average Precision
RDD	Resilient Distributed Dataset
HDFS	Hadoop Distributed File System
AWS	Amazon Web Services
DaG	Directed Acyclic Graph
SQL	Structured Query Language
BST	Binary Search Tree
SMOTE	Synthetic Minority Over-sampling Technique
ENN	Edited Nearest Neighbour
SVM	Support Vector Machine
KDE	Kernel Density Estimator
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network

Anexos

A Mejores resultados de clasificación con Random Forest

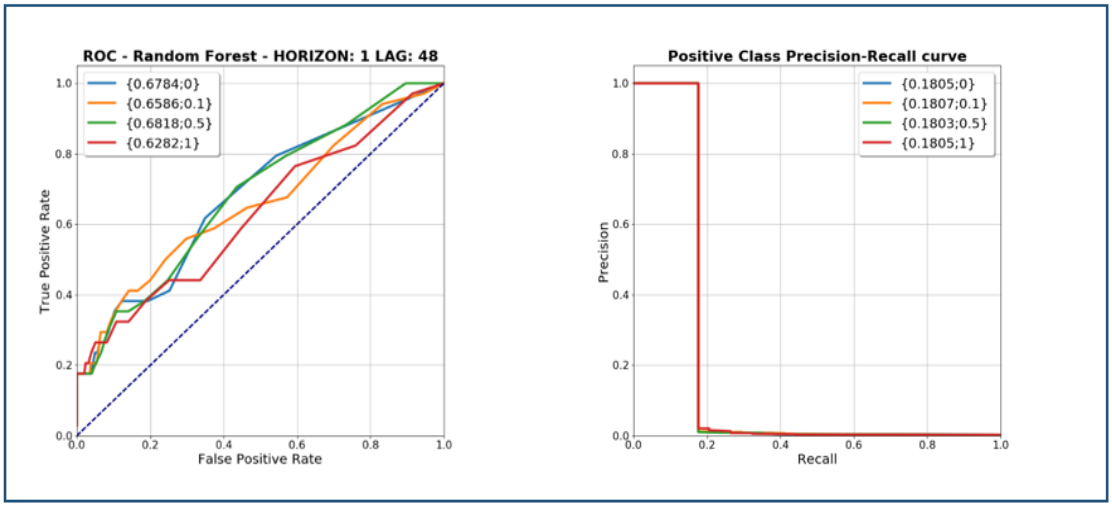

Figura 0-1: Mejores curvas para Random Forest con un horizonte de predicción igual a 1

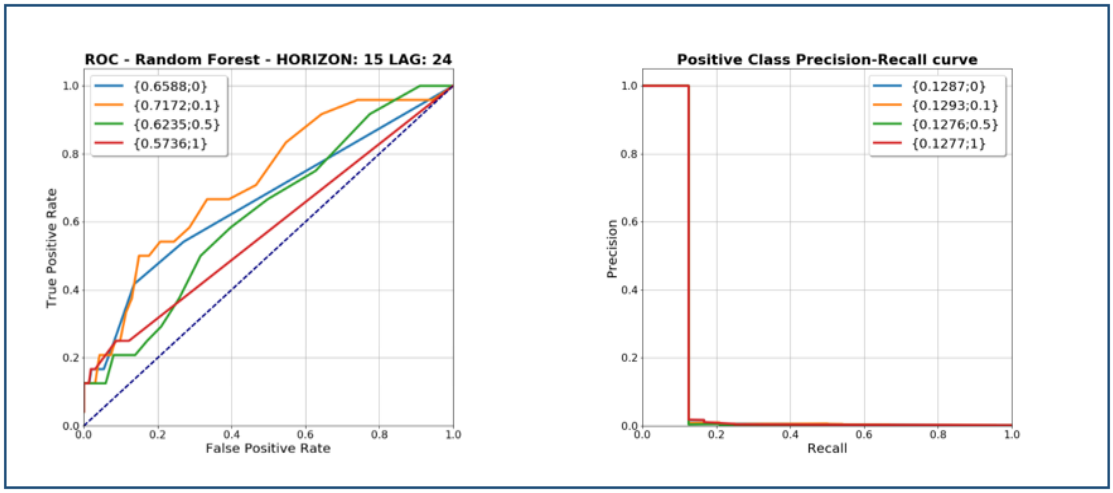

Figura 0-2: Mejores curvas para Random Forest con un horizonte de predicción igual a 15

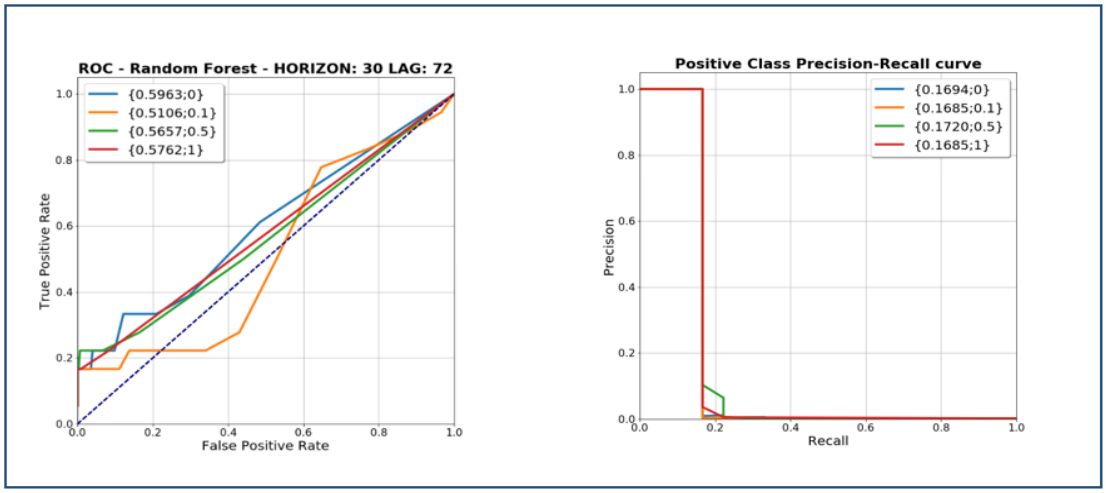

Figura 0-3: Mejores curvas para Random Forest con un horizonte de predicción igual a 30

B Mejores resultados de clasificación con AdaBoost

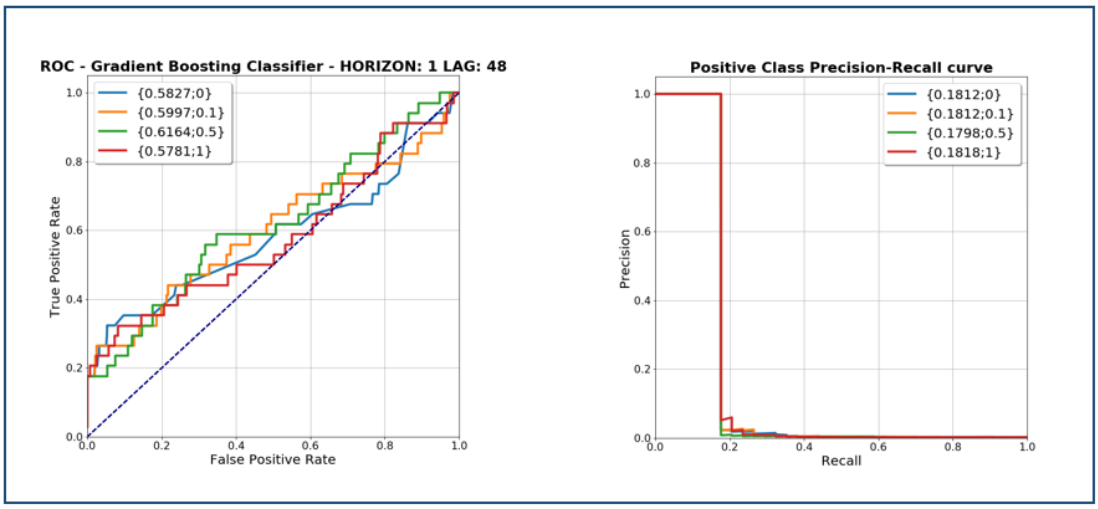

Figura 0-4: Mejores curvas para AdaBoost con un horizonte de predicción igual a 1

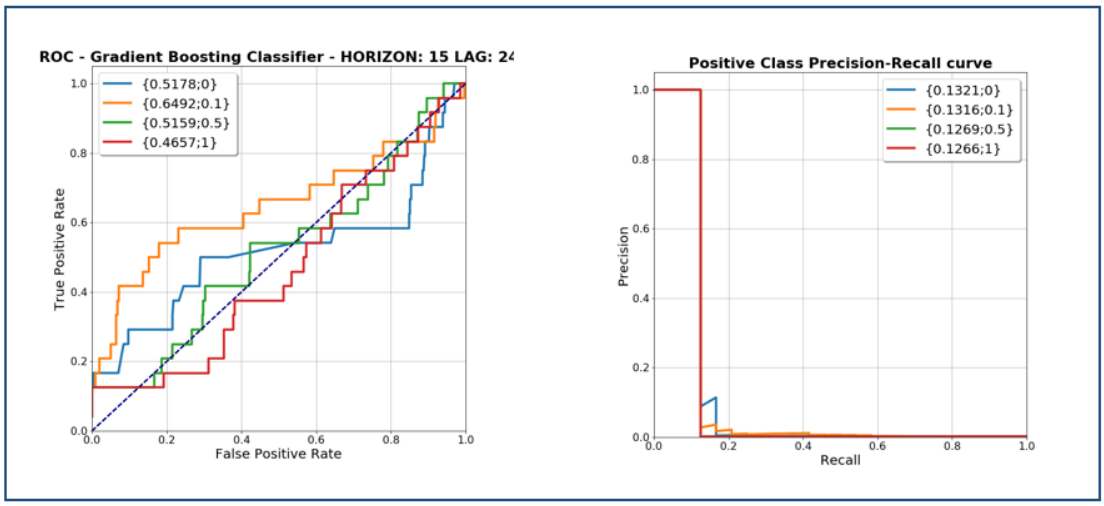

Figura 0-5: Mejores curvas para AdaBoost con un horizonte de predicción igual a 15

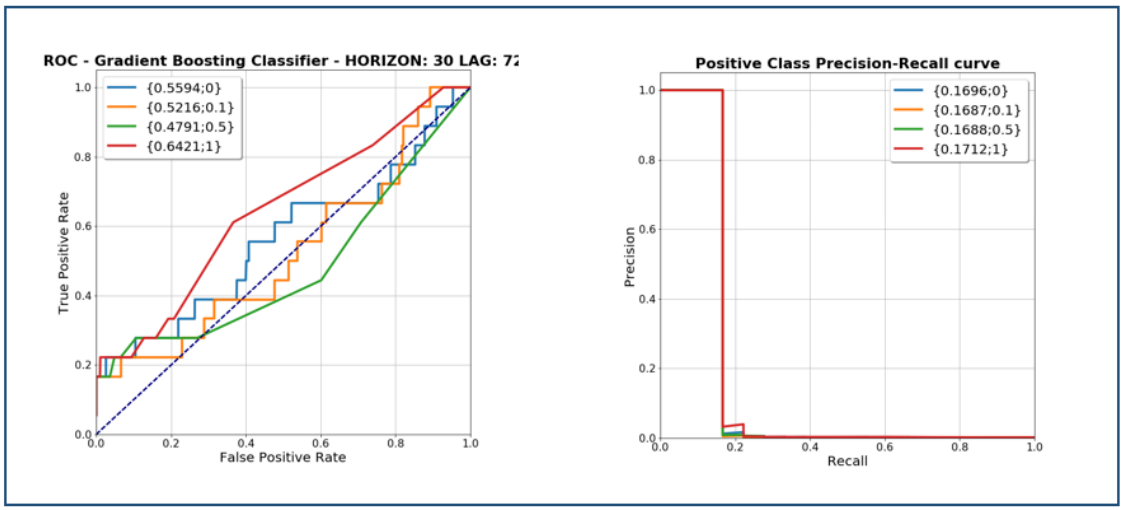

Figura 0-6: Mejores curvas para AdaBoost con un horizonte de predicción igual a 30

C Mejores resultados de clasificación con SVM

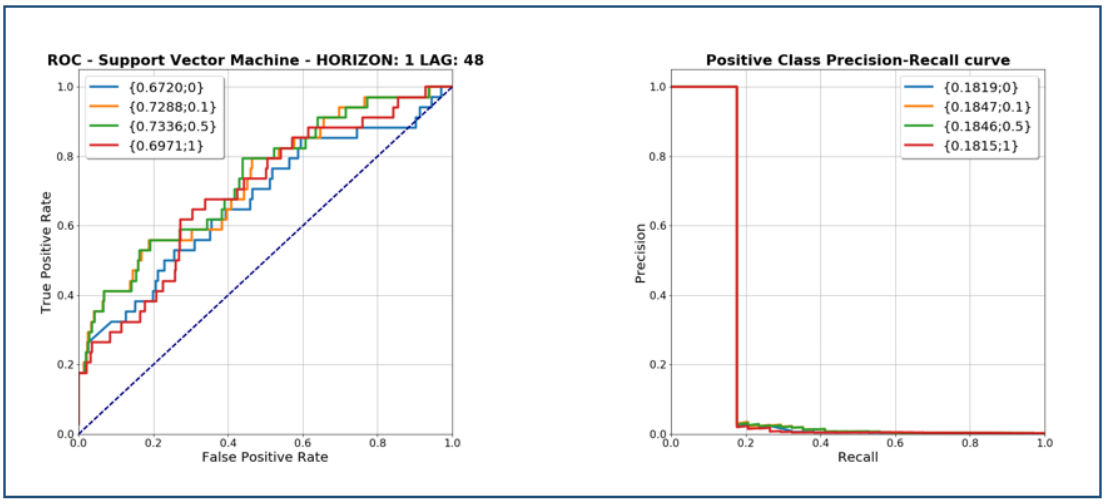

Figura 0-7: Mejores curvas para SVM con un horizonte de predicción igual a 1

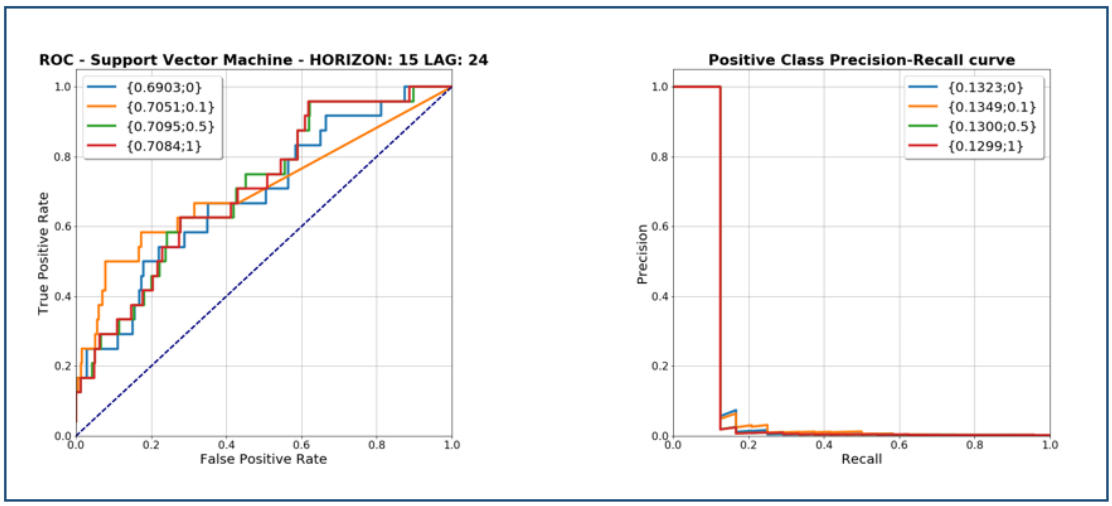

Figura 0-8: Mejores curvas para SVM con un horizonte de predicción igual a 15

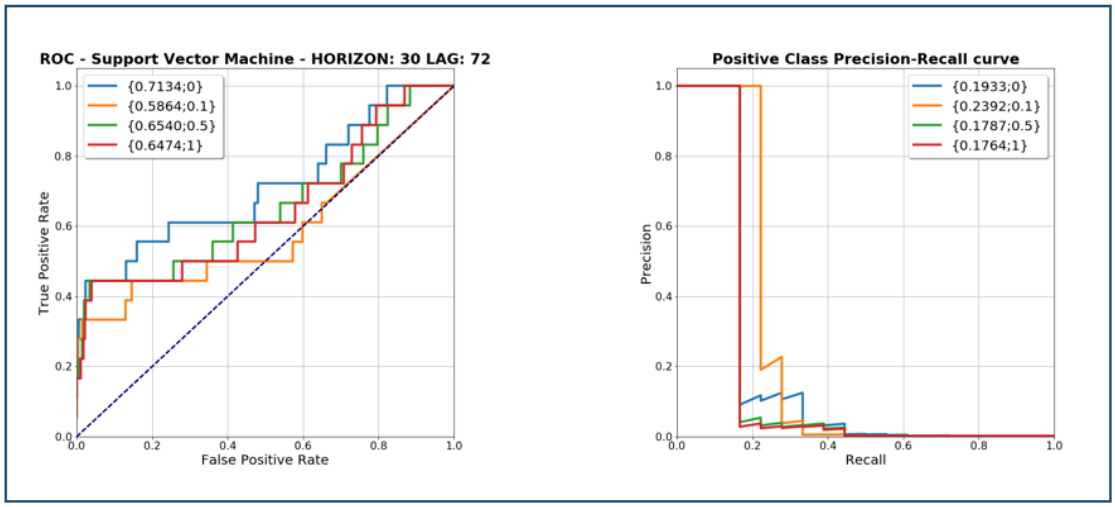

Figura 0-9: Mejores curvas para SVM con un horizonte de predicción igual a 30

D Mejores resultados de clasificación con iForests

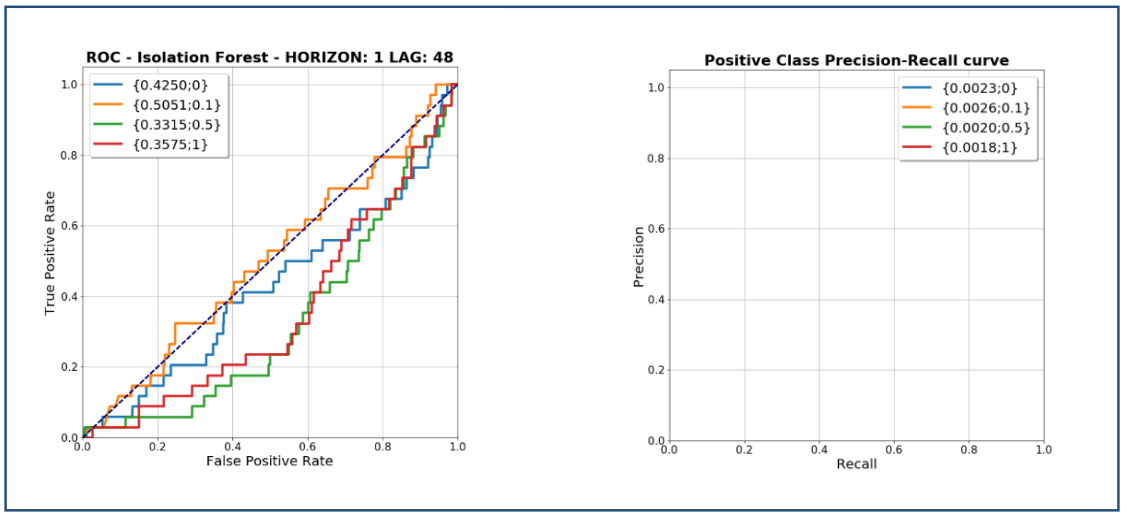

Figura 0-10: Mejores curvas para iForests con un horizonte de predicción igual a 1

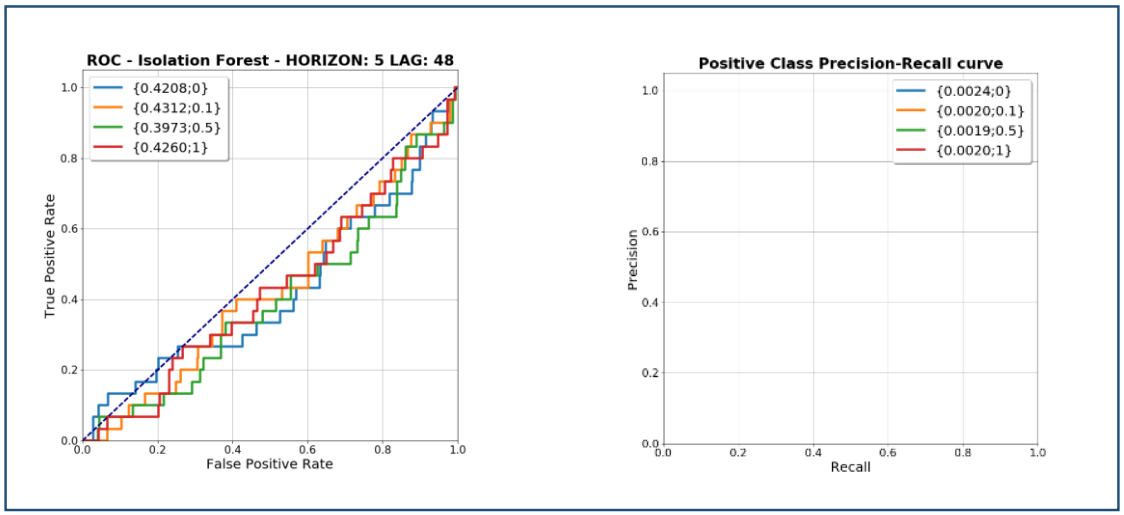

Figura 0-11: Mejores curvas para iForests con un horizonte de predicción igual a 5

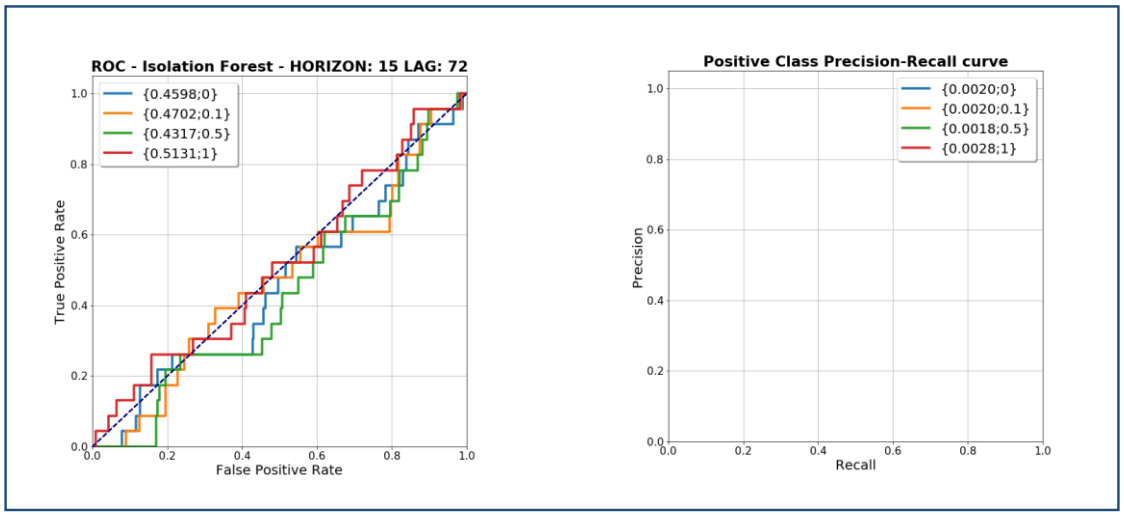

Figura 0-12: Mejores curvas para iForests con un horizonte de predicción igual a 15