SAIN4

Sistemas Avanzados de eficiencia productiva para la Industria 4.0

PROGRAMA: PROYECTOS DE I+D EN COLABORACIÓN

ACTUACIÓN: IMDECA-Proyectos de I+D en colaboración

Entregable 3.1 (E3.1- Publicable)

Informe de Resultados del Motor de Prognosis para la Eficiencia Productiva

Perteneciente al paquete de trabajo: PT3

Participante responsable: ITI

Mes estimado de entrega: Mes 30

RESUMEN

SAIN4 es un proyecto financiado con el Instituto Valenciano de Competitividad Empresarial (IVACE) y la Unión Europea a través del Fondo Europeo de Desarrollo Regional (FEDER).

El presente documento tiene el objetivo detallar las tareas de diseño y construcción de un Motor de Prognosis que permita a las empresas Valencianas de los sectores Madera-Mueble y Metalmecánico, en términos generales, la optimización de tres factores con repercusión en el OEE del proceso: la Eficiencia en los equipos industriales, su Disponibilidad gracias a los procesos de mantenimiento y la estimación de la Calidad en la producción.

ABSTRACT

SAIN4 is a project funded by the Valencian Institute for Business Competitiveness (IVACE) and the European Union through the European Regional Development Fund (FEDER).

This document aims to detail the tasks of design and construction of a Prognosis Engine that will allow Valencian companies in the Wood-Furniture and Metal-Mechanical sectors, in general terms, to optimize three factors with an impact on the OEE of the process: Efficiency in Industrial equipment, its availability thanks to the processes of maintenance and the estimation of the Quality in the production.

Tabla de Contenidos

1 Introducción

1.1 Objetivos del Paquete de Trabajo 3

Este paquete tiene el objetivo de diseñar y desarrollar un Motor de Prognosis que sea capaz de modelar y recomendar mejoras de la eficiencia del sistema productivo de la Industria 4.0. Para ello, el sistema deberá contemplar:

- Tecnologías en el dominio de *Big Data Analytics* para proveer de un **sistema de procesamiento paralelo eficiente para de grandes volúmenes de datos** provenientes del sistema de Captura de Datos visto en el PT2.
- Técnicas estadísticas de **procesamiento y análisis de datos** para la modelización del sistema inteligente que permita la *medición, predicción* y *detección* de: (1) anomalías de funcionamiento, (2) optimización de parámetros relacionados con la calidad de producción.

Basándonos en medidas de precisión tomadas en tiempo real mediante distintos sistemas de información, como por ejemplo los sensores instalados en máquina, se desarrollarán algoritmos de modelado estadístico y aprendizaje automático para inferir y modelar los parámetros de configuración óptimos para un determinado par producto/máquina. El **objetivo** es ofrecer información al usuario que le ayude a maximizar la calidad final del producto, minimizar los riesgos de averías y los recursos consumidos.

1.2 Objetivo del Presente Documento

El objetivo del entregable E3.1 es detallar las tareas de diseño y construcción de un Motor de Prognosis que permita a las empresas Valencianas de los sectores Madera-Mueble y Metalmecánico, en términos generales, la optimización de tres factores con repercusión en el OEE del proceso, es decir la Eficiencia en los equipos industriales, su Disponibilidad gracias a los procesos de mantenimiento y la estimación de la Calidad en la producción.

En concreto, recoge las actividades realizadas durante la ejecución del paquete de trabajo y que incluye:

- **Estudio de las técnicas y tecnologías de Deep Learning:** realización de un estado del arte de técnicas de Deep Learning con el objetivo de identificar qué topologías de redes neuronales profundas son de utilidad para el Motor de Prognosis.
- **Diseño del Motor de Prognosis:** a partir de la aplicación de la metodología para la selección de las variables explicativas del proceso productivo, se realizó un trabajo de identificación de los objetivos, estructuración y especificación de las técnicas estadísticas que formarán parte del ecosistema de modelos estadísticos que contiene el Motor de Prognosis.
- **Construcción y validación del Motor de Prognosis:** componente integrador de los modelos estadísticos que debe ser desplegado, integrado y validado para comprobar el correcto funcionamiento del Sistema de Gestión Avanzada (SGA).

2 Estudio de técnicas y tecnologías para Deep Learning

2.1 Introducción

Desde tiempos ancestrales ha existido el deseo de crear máquinas que piensen. De hecho, cuando se concibieron las primeras computadoras programables las personas ya se preguntaban si dichas computadoras podrían llegar a ser inteligentes. En la actualidad, la inteligencia artificial es un campo muy activo con infinidad de aplicaciones prácticas y temas de investigación abiertos que están siendo desarrollados. Hoy en día se puede observar como los programas basados en Machine Learning o Inteligencia Artificial son capaces de automatizar labores rutinarias, interpretar voz e imágenes, realizar diagnósticos, etc. En los primeros tiempos de la Inteligencia Artificial, estas técnicas eran capaces de resolver, de manera eficiente, problemas de formalización complicada, desde un punto de vista intelectual.

Conceptualmente el objetivo buscado consiste en que las computadoras aprendan a partir de la experiencia y entiendan el mundo en términos de conceptos jerárquicos, donde cada concepto tiene relaciones con conceptos cada vez más simples. El que las computadoras sean capaces de aprender a partir de la experiencia evita la necesidad de introducir el conocimiento a través de complicados formalismos que requieren del conocimiento de expertos. La jerarquía de los diferentes conceptos permite que las computadoras aprendan los conceptos cada vez más complicados de manera incremental a partir de la construcción de los más simples. Esta idea que va desde lo más simple a lo más complejo, nos lleva a grafos conceptuales con gran profundidad, pues suelen disponer de un gran número de niveles. Es esta la razón de ser del concepto acuñado como **Deep Learning**.

Algunos proyectos sobre Inteligencia Artificial introducen el conocimiento mediante el uso de lenguajes formales, de manera que una computadora puede razonar de manera automática a partir de las evidencias observadas en un momento dado y las reglas introducidas en las bases de conocimiento mediante estos lenguajes. La complejidad a la hora de introducir el conocimiento requerido para poder realizar deducciones nos lleva a la necesidad de crear sistemas que adquieran dicho conocimiento de forma automática a través de los datos observados. Esta capacidad de aprendizaje automático es la que se conoce con el término de **Machine Learning**. La introducción de este nuevo enfoque permitió que las computadoras resolvieran problemas que suponían un conocimiento previo o tomaran decisiones que eran bastante subjetivas. Los modelos aprendidos de esta forma toman como entrada una serie de evidencias que se conocen como **características** y ofrecen un resultado a partir de dichas evidencias observadas. El resultado puede ser una predicción, una clasificación o una clusterización, dependiendo de la naturaleza del resultado para el que el modelo de machine learning fue pensado originalmente.

El rendimiento de estos algoritmos de machine learning depende enormemente del tipo de representación de los datos que se ofrecen. Esta dependencia con la representación de los datos es un problema bastante habitual en este campo. Así, una operación de búsqueda en una colección de datos puede ver reducido su coste de forma exponencial si dichos datos se representan o indexan con una estructura adecuada.

Muchas tareas de inteligencia artificial pueden resolverse mediante la obtención de un conjunto correcto de características. Sin embargo, en muchas ocasiones, es muy difícil conocer cuáles son las características que deberían de extraerse. Sin embargo, esta tarea suele ser complicada. Una posible solución a este problema consiste en usar machine learning para descubrir no solo el mapa desde la representación a la salida final sino la propia representación. Esta aproximación se conoce como **aprendizaje de la representación**. El aprendizaje de la representación suele resultar en una mejora clara del rendimiento de los algoritmos de inteligencia artificial que pueden aprender y adaptarse a nuevas tareas con una intervención humana mínima.

Un típico algoritmo para el aprendizaje de la representación es el **autoencoder**. Un autoencoder consiste en la combinación de una función de codificación (encoder), que convierte los datos de entrada en una representación diferente, y una función de decodificación (decoder) que convierte la nueva representación al formato original. Los autoencoders se entrenan para preservar la mayor cantidad de información posible ante una entrada ofrecida al codificador y pasada posteriormente por el decodificador, la ventaja de la aplicación de autoencoders es que se entrenan para ofrecer nuevas representaciones de los datos con propiedades interesantes. Diferentes tipos de autoencoders tienen diferentes tipos de propiedades.

Cuando se diseñan las características o los algoritmos para el aprendizaje de las características, el objetivo habitualmente consiste en separar los factores de variación que explican a los datos observados. Habitualmente estos factores no son cantidades observadas directamente sino que se extrapolan a través de otras medidas observadas, pues los primeros afectan a las mediciones observadas. Podría pensarse en ellos como conceptos de alto nivel de abstracción que nos permiten explicar la variabilidad observada en nuestros datos.

Deep learning resuelve este problema central, relativo al aprendizaje de la representación, introduciendo representaciones que se expresan en términos de otras representaciones más simples. Un ejemplo común de un modelo de Deep learning es el **perceptrón multicapa (MLP)**. Un MLP es justamente una función matemática que proyecta una serie de entradas en una serie de salidas. La función de un MLP se forma a través de la composición de muchas funciones simples. De esta manera se podría pensar que en la aplicación de una función matemática ofrece una nueva representación de la entrada.

La idea relacionada con el aprendizaje de la correcta representación de los datos ofrece una de las perspectivas sobre Deep learning. Otra perspectiva del Deep learning consiste en considerar que cada capa de profundidad permita a la computadora aprender un programa con múltiples pasos. De manera, que cada capa de la representación puede ser visto como el

estado de la memoria de la computadora después de haber ejecutado otro conjunto de instrucciones en paralelo. Las redes con una gran profundidad pueden ejecutar más instrucciones en secuencia. Las instrucciones secuenciales nos ofrecen un gran potencial porque las más tardías pueden tener referencias a los resultados obtenidos en las instrucciones más tempranas. De esta manera, no toda la información presente en una capa de activación codifica necesariamente los factores de variación que explican la entrada. La representación también guarda información sobre el estado cuya función podría ser similar a la de un contador o puntero en una programa de ordenador tradicional.

Otra aproximación, utilizada por los modelos probabilísticos profundos, visualiza la profundidad de un modelo no como la profundidad de un grafo computacional sino como la profundidad de un grafo que describe como se relacionan los diferentes conceptos unos con otros. Esto es debido a que el entendimiento por parte de los sistemas de los conceptos simples puede ser refinado ofreciendo información relativa a conceptos más complejos.

Debido a que no siempre está claro cuál de los dos puntos de vista, la profundidad de un grafo computacional, o la profundidad de un modelo de grafo probabilístico, es más relevante, no hay un único valor correcto para la profundidad de una arquitectura, al igual que no hay un valor objetivo para la longitud de un programa de ordenador. Ni tampoco hay un consenso a la hora de definir la profundidad a partir de la cual un modelo se puede calificar como “Deep” profundo. Sin embargo, Deep learning puede ser visto como el estudio de modelos que envuelven una gran cantidad de funciones o conceptos aprendidos que tradicionalmente hace el machine learning.

Una primera clasificación de los modelos se dividen desde el punto de vista de la necesidad de etiqueta (**supervisión**) o no (**sin supervisión**) que el método de Machine Learning aplicado requiere.

La tabla que se muestra a continuación divide las redes bajo esta perspectiva.

Tipo	Neural networks
Con Supervisión	RNN CNN
Sin Supervisión	AUTOENCODERS GAN ART SOM

2.2 Topologías de redes neuronales

2.2.1 Redes sin supervisión

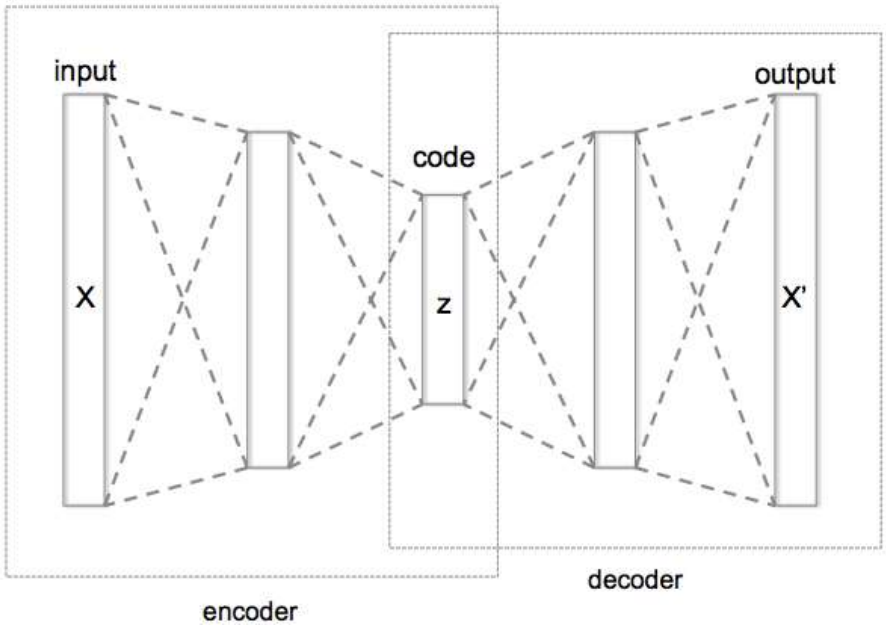
2.2.1.1 Autoencoder

Un *autoencoder* es una red neuronal artificial utilizada para el aprendizaje no supervisado de la representación de características. Así, uso de los *autoencoders* consiste en el aprendizaje de una codificación de un conjunto de datos, siendo el objetivo final más común la obtención de

una reducción de la dimensionalidad del espacio original. Actualmente, el concepto de autoencoder está también muy vinculado con el aprendizaje de modelos generativos de datos para simulación.

Arquitectónicamente, la forma más simple de autoencoder consiste en una red no recurrente muy similar al perceptrón multicapa (MLP), que tiene una capa de entrada, una capa de salida y una o más capas ocultas conectando a ambas. La capa de salida contiene el mismo número de nodos que la capa de entrada. El propósito de este tipo de redes no es otro que el reconstruir su propia entrada y no el de predecir determinados valores de salida relacionados con los primeros que suele ser lo habitual en las redes neuronales. Por lo tanto, se puede ver a los autoencoders como modelos de aprendizaje no supervisado, pues las propias entradas son los valores de salida deseados.

La siguiente figura muestra la estructura general de un autoencoder. Conforme se puede apreciar en la figura, éste se compone de una capa de entrada (input) y una de salida (output), ambas de la misma dimensión, y una serie de capas ocultas que confluyen a una central (code), que serán las variables latentes de la entrada. Los autoencoder se componen de dos fases: una primera fase conocida como *encoder* (va desde *input* a *code*) y una segunda etapa conocida como *decoder* (va desde *code* a *output*). Tal y como puede observarse code tiene un menor número de neuronas y sería la representación utilizada de una entrada si nuestro objetivo final es la reducción de la dimensionalidad del espacio original.



Estructura esquemática de un autoencoder con 3 capas ocultas completamente conectadas

2.2.1.2 Generative adversarial networks (GAN)

Las redes generativas adversarias son una clase de algoritmos de inteligencia artificial no supervisados, implementadas por un sistema con dos redes neuronales que se contestan una a otra en un juego consistente en buscar el equilibrio entre ambas. Su uso principal es para la simulación de muestras sintéticas.

Así, una de las redes genera candidatos y la otra los evalúa. El modelo generador aprende el espacio latente de una distribución de datos particular, para la que se quiere crear una simulación, mientras que la red discriminativa se encarga de discernir entre las instancias que proceden de la distribución de datos real y los candidatos producidos por el generador. El objetivo de la red generadora consiste en incrementar la tasa de error de la red discriminativa, pues conforme mayor sea este error más parecidas serán las muestras reales a las generadas sintéticamente.

2.2.1.3 ART

La teoría de resonancia adaptativa (ART) es una teoría desarrollada por Stephen Grossberg y Gail Carpenter sobre aspectos relacionados con el modo en el que el cerebro procesa la información. En ella se describen un número de modelos de redes neuronales que hacen uso de métodos de aprendizaje supervisado y no supervisado, enfocados a problemas de reconocimiento de patrones y predicción.

La primera intuición detrás de los modelos ART es que la identificación y el reconocimiento del objeto generalmente ocurren como resultado de la interacción entre las expectativas del observador (top-down) y la información procedente de un sistema de sensores (bottom-up). El modelo postula que las expectativas (top-down) toman la forma de un prototipo en memoria que es comparado con las características de un objeto detectado por los sensores. Esta comparación ofrece una medida de la pertenencia a una categoría. De esta manera, conforme la diferencia entre lo esperado y lo medido no exceden un umbral, conocido éste como *parámetro de vigilancia*, el objeto medido mediante el sistema de sensores será considerado como perteneciente a la clase esperada. Este sistema ofrece una solución al problema de plasticidad y estabilidad, un ejemplo del cual sería la adquisición de nuevo conocimiento sin quebrantar el ya adquirido previamente (aprendizaje incremental).

Los sistemas ART básicos son modelos de aprendizaje no supervisado. A grandes rasgos, estos sistemas disponen de un campo de comparación y otro de reconocimiento compuesto por neuronas, un parámetro de vigilancia (umbral de reconocimiento), y un módulo de reinicialización. El campo de comparación recoge un vector de características y lo transfiere a su mejor patrón en el campo de reconocimiento. El mejor patrón viene representado por la neurona cuyo conjunto de pesos se ajustan mejor al vector de entrada. Cada neurona del campo de reconocimiento obtiene como salida una señal negativa a cada una de las otras neuronas del campo de reconocimiento inhibiendo así su salida si la entrada está muy distante de lo aprendido por dicha neurona de salida. De esta forma el campo de reconocimiento exhibe la conocida como inhibición lateral, que permite que cada neurona en dicho campo represente a una categoría a la cual pertenezca un vector de entrada.

Después de que un vector de entrada se clasifique, el módulo de reinicio compara la fuerza del patrón de reconocimiento con el parámetro de vigilancia. Si el parámetro de vigilancia es superado (el vector de entrada está dentro del rango de vectores de entrada previamente observados), el entrenamiento da comienzo: Los pesos de la neurona ganadora son reajustados hacia las características del vector de entrada. De otra forma, si el nivel del patrón está por debajo del parámetro de vigilancia (el patrón del vector de entrada está fuera del rango de valores normales para esa neurona) la neurona ganadora de la capa de reconocimiento es inhibida y se lleva a cabo un procedimiento de búsqueda. Durante este procedimiento de búsqueda, las neuronas de reconocimiento se deshabilitan una a una por la función de reinicio, hasta que el parámetro de vigilancia es superado por un patrón de

reconocimiento. En el caso en el que ninguna neurona de reconocimiento supere el umbral del parámetro de vigilancia, se compromete una neurona y sus pesos se reajustan hacia el patrón del vector de entrada. Tal y como puede observarse, por el procedimiento de aprendizaje, el parámetro de vigilancia tiene una influencia considerable sobre el sistema: valores altos del parámetro de vigilancia tenderán a producir categorías de gránulo fino, mientras que valores bajos del parámetro de vigilancia resultarán en categorías más generales.

2.2.1.4 SOM

Un mapa autoorganizativo (SOM) es un tipo de red neuronal artificial (ANN) que se entrena mediante el uso de técnicas no supervisadas y que produce una representación discreta de las características de entrada, es un mapa de muy pocas dimensiones (normalmente dos), es por ello que puede considerarse como un método de reducción de la dimensionalidad. Los mapas autoorganizativos difieren de las redes neuronales artificiales, en el hecho de que aplican técnicas de aprendizaje competitivo en lugar de las típicas técnicas de reducción del error, y además usan funciones de vecindad para preservar las propiedades topológicas del espacio de entrada. Esto hace a las SOM muy útiles a la hora de visualizar en muy pocas dimensiones datos con una elevada dimensionalidad.

2.2.2 Redes Supervisadas

2.2.2.1 Convolucionales

Las redes convolucionales, también conocidas como redes neuronales convolucionales (CNN), son un tipo especial de redes neuronales para procesar datos que tienen una topología en rejilla (grid). Ejemplos de este tipo son las series temporales, que pueden ser vistas como un grid 1D donde los ejemplos se muestrean a intervalos regulares, o las imágenes que pueden ser vistas como grid 2D de píxeles. El nombre de *red neuronal convolucional* indica que utilizan una operación matemática llamada convolución. Las redes convolucionales son redes neuronales donde se usa la operación matemática de la convolución en lugar de la multiplicación general de matrices en al menos una de las capas.

En su forma general una convolución consiste en el producto entre dos funciones. En las redes convolucionales la primera función sería la entrada y la segunda el Kernel, conociéndose la salida de la convolución como el mapa de características. Además, en las aplicaciones de Machine Learning, la entrada es frecuentemente un vector multidimensional que referiremos como tensor.

La convolución maneja tres ideas importantes que pueden ayudarnos a mejorar los sistemas de Machine Learning: **interacciones dispersas**, **compartición de parámetros** y **representaciones equivariantes**. Además, la convolución nos ofrece un medio para trabajar con entradas de tamaño variable.

Las redes convolucionales, tienen **interacciones dispersas**. Esto se consigue haciendo un kernel más pequeño que la entrada. Por ejemplo, cuando se procesa una imagen, la imagen de entrada podría tener miles de millones de píxeles, pero nosotros podemos detectar pequeñas

características significativas, como fronteras que ocupan muy pocos píxeles. Esto supone una reducción en el número de parámetros a almacenar, reduciendo así la necesidad de recursos de memoria y mejorando la eficiencia estadística, reduciendo también el número de operaciones a realizar para obtener la salida final. Esto permite a las redes describir eficientemente complicadas interacciones entre muchas variables, construyendo dichas interacciones a partir de bloques simples que describen sólo interacciones dispersas.

La **compartición de parámetros** se refiere al hecho de usar el mismo parámetro para más de una función en el modelo. En una red neuronal tradicional, cada elemento de la matriz de pesos se utiliza exactamente una única vez cuando se calcula la salida de una capa. Un sinónimo del concepto de compartición de parámetros, es el de pesos ligados, pues el valor de un peso aplicado a una entrada está ligado al valor de un peso aplicado en cualquier lugar de la red. En una red neuronal convolucional, cada miembro del kernel se usa en cada una de las posiciones de la entrada. La compartición de parámetros implementada mediante la operación de convolución significa que en lugar de aprender un conjunto diferente de parámetros para cada una de las posiciones, aprendemos únicamente un conjunto para todas. Con esto conseguimos una reducción en el número de parámetros a estimar que redunda en una reducción considerable en las necesidades de almacenamiento del modelo por la reducción de parámetros que la compartición de parámetros supone, mejorando además indirectamente las propiedades estadísticas.

En el caso de la convolución, la forma particular de los parámetros compartidos causa que la capa tenga la propiedad equivariante a la traslación. Esto quiere decir que si la entrada cambia entonces la salida también cambia. Matemáticamente, una función f(x) es equivariante a una función g si f(g(x))=g(f(x)). En el caso de la convolución, si g es una función que traslada la entrada, entonces la función de convolución es equivalente a g. Se ha de tener en cuenta que la convolución no es equivariante a otras transformaciones como los cambios de escala o las rotaciones.

2.2.2.2 Redes recurrentes y recursivas

(fuente: www.deeplearningbook.org)

Las redes neuronales recurrentes (RNN) son una familia de redes neuronales para el proceso de datos secuenciales. Si se considera la forma clásica de un sistema dinámico:

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

La ecuación de arriba es recurrente porque la definición de **S** en el instante **t** es la misma que en el instante **t-1**. Para un número finito de pasos τ se descompone aplicando la misma definición τ-1 veces. Por ejemplo si τ=3, obtenemos:

$$s^{(3)} = f(s^2; \theta) = f(f(s^1; \theta); \theta)$$

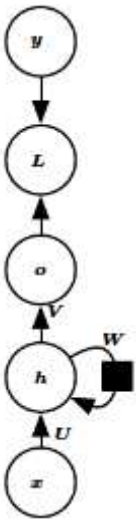
Por otra parte, es posible representar un sistema dinámico que incluye variables exógenas ($x^{(t)}$) como:

$$s^{(t)} = f(s^{(t-1)}; x^{(t)}; \theta)$$

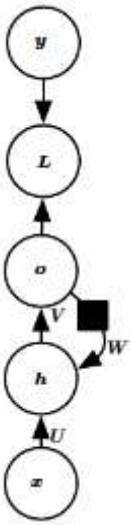
dicha función nos muestra como los parámetros o variables de entrada son: la propia señal en el instante anterior y variables medidas en este instante actual.

Algunos ejemplos de patrones de diseño de redes neuronales recurrentes son:

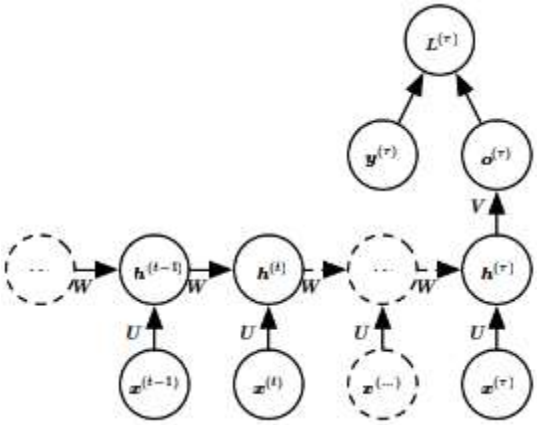
- Redes recurrentes que producen una salida en cada paso temporal y tienen conexiones recurrentes entre las capas ocultas.



- Redes recurrentes que producen una salida en cada paso temporal y tienen conexiones recurrentes sólo desde la salida a las capas ocultas.

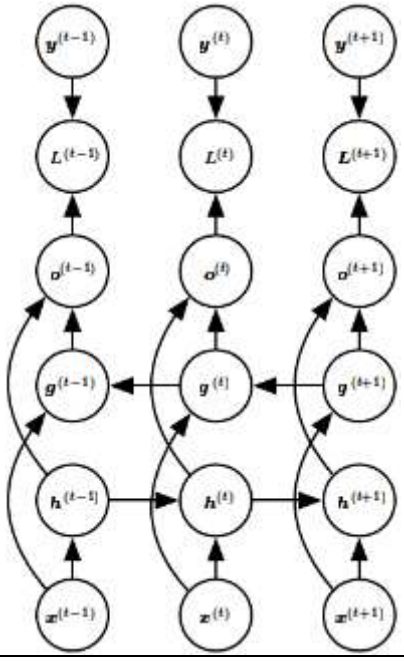


- Redes recurrentes con conexiones recurrentes entre capas ocultas, que leen una secuencia entera y producen una única salida.

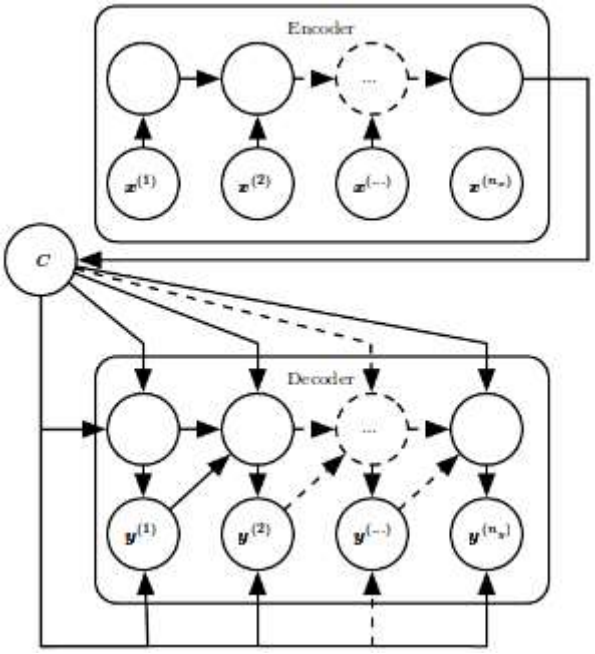


Hasta este momento, todas las redes neuronales consideradas disponen de una estructura causal, lo que significa que el estado en el instante t sólo captura información desde el pasado, $x^{(1)}, \ldots, x^{(t-1)}$, y la entrada presente $x^{(t)}$. Algunos de los modelos también permitían añadir información de los valores pasado de y. Sin embargo en muchas ocasiones se desea una salida $y^{(t)}$ que dependa de la secuencia completa de la entrada. Por ejemplo, en reconocimiento del habla, la interpretación del sonido como un fonema puede depender también de los nuevos fonemas debido a la coarticulación y de manera potencial también puede depender de las dependencias lingüísticas entre palabras cercanas: en una situación de ambigüedad acústica de la palabra, será bueno poder observar el contexto futuro al igual que el pasado para poder realizar una buena desambiguación.

Tal y como su nombre sugiere, las redes neuronales recurrentes bidireccionales combinan dos redes neuronales recurrentes una que se mueve hacia delante desde el principio de la secuencia hasta el estado actual, representada con $h^{(t)}$, y otra que se mueve hacia atrás desde el final de la secuencia hasta el instante actual, representada con $g^{(t)}$. Esto permite que las neuronas de salida de la red calculen una representación que depende de ambos, que desde un punto de vista temporal representaría al pasado y al futuro.



Hasta ahora hemos visto como una RNN puede mapear una secuencia de entrada a un vector de tamaño fijo. Además, hemos visto RNNs que mapeaban vectores de tamaño fijo a una secuencia. También se ha visto como una RNN puede mapear una secuencia de entrada a una secuencia de salida de la misma longitud. Ahora vamos a presentar una RNN, denominada codificador-decodificador, que puede ser entrenada para ofrecer un mapeo de una secuencia de entrada a una secuencia de salida que no tienen porque tener la misma longitud. Resolviendo así los casos en los que las secuencias de entrada y salida en el entrenamiento no tienen la misma longitud. Este tipo de redes se usan en aplicaciones de traducción automática y reconocimiento del habla, donde las secuencias de entrada y salida no tienen la misma longitud.



En este tipo de redes a la entrada **X** se le denomina contexto y lo que se pretende es aprender una representación del contexto **C** que será un vector o conjunto de vectores que resumen la secuencia de entrada. Dicho vector C será la entrada al decodificador que producirá la secuencia de salida. Cabe destacar que para la arquitectura presentada aquí, el tamaño de las secuencias de entrada y salida pueden no ser iguales.

El cálculo de una red profunda recurrente se puede descomponer en tres bloques de parámetros y transformaciones asociadas:

- Desde las entradas hacia los estados ocultos.
- Desde los estados ocultos previos a los siguientes estados ocultos.
- Desde los estados ocultos a la salida.

Cada uno de estos bloques tiene asociada una única matriz de pesos. De esta manera la salida de cada nivel se corresponde con una transformación de su propia entrada y pueden ir introduciéndose niveles (Deep), donde cada nuevo nivel introducido va a ir sintetizando la

información precedente. Así las capas externas sintetizarán la información más general y conforme se avance en los niveles más profundos se irá sintetizando información que irá más focalizada al detalle.

En las redes neuronales recurrentes, el aprendizaje de dependencias a largo plazo es un reto matemático de gran interés. El problema básico consiste en que los gradientes propagados a lo largo de muchas etapas tienden o bien a volverse insignificantes, paralizando el proceso de aprendizaje, o en otras ocasiones a explotar, provocando graves problemas durante el proceso de optimización. A continuación se muestra una descripción detallada del problema.

El cálculo de las redes recurrentes conlleva la composición de la misma función múltiples veces, una para cada paso dado. Estas composiciones pueden llevar a comportamientos no lineales. En particular la función de composición utilizada por las redes neuronales se puede representar mediante el cálculo matricial como:

$$h^{(t)} = W^T h^{(t-1)}$$

Donde: $h^{(t)}$ es el valor de la capa oculta en el instante t, W es una matriz de pesos.

De manera simplificada bajo la perspectiva de una red neuronal recurrente simple quedaría:

$$h^{(t)} = (W^t)^T h^{(0)}$$

Conforme se puede observar W^t irá haciendo cada vez más pequeños los pesos menores que 0 y más grandes los pesos mayores que 1. Este problema conlleva la dificultad de aprender relaciones entre secuencias temporales actuales con secuencias pasadas varios instantes temporales atrás. Para solucionar este problema se hace uso de las redes **LSTM** que han demostrado un funcionamiento adecuado a la hora de modelar este tipo de relación con instantes pasados.

2.3 Tecnologías para Deep Learning

2.3.1 Tensor Flow

TensorFlow (Martín Abadi et al.) es por una parte una interfaz para poder expresar algoritmos de Machine Learning, y por otra una implementación para ejecutarlos. Se trata de una librería de código abierto liberada en Noviembre de 2015 bajo la licencia de Apache 2.0 y disponible en www.tensorflow.org. Los cómputos expresados haciendo uso de TensorFlow pueden ejecutarse con pocos o incluso ningún cambio en una gran variedad de sistemas heterogéneos. El sistema es flexible y puede ser usado para expresar una gran variedad de algoritmos, incluyendo el entrenamiento e inferencia de algoritmos para modelos de redes neuronales profundas.

TensorFlow permite a los clientes expresar fácilmente varios tipos de paralelismo a través de la replicación y la ejecución paralela de un modelo representado como un grafo de flujo de

datos, con muchos dispositivos diferentes colaborando en la actualización de una serie de parámetros compartidos u otros estados.

Los componentes principales en un sistema TensorFlow son el *cliente*, que utiliza una sesión para comunicarse con el *máster*, y uno o más procesos *worker*, donde cada proceso worker es reponsable de arbitrar el acceso a uno o más dispositivos de cómputo (cores CPU, tarjetas GPU) y también de ejecutar los nodos de grafo sobre estos dispositivos tal y como se lo indica el máster. Tenemos además dos implementaciones diferentes de la interfaz de TensorFlow, una interfaz local y otra distribuida. La implementación local se utiliza cuando el cliente, el máster y el worker corren juntos sobre una única máquina en el contexto de un único proceso de sistema operativo (posiblemente atacando varios dispositivos si por ejemplo se dispone de varias tarjetas GPU). La implementación distribuida comparte la mayor parte del código con la implementación local, pero lo extiende para dar soporte a un entorno donde el cliente, el máster y los workers pueden ser procesos diferentes sobre máquinas diferentes.

Los dispositivos son el corazón principal de TensorFlow. Cada worker es responsable de uno o más dispositivos, y cada dispositivo está identificado mediante un tipo y un nombre. Cada objeto dispositivo es responsable de gestionar el alojamiento y desalojamiento de la memoria del dispositivo y de organizar la ejecución de cualquier kernel que sea demandado por los niveles superiores en la implementación de TensorFlow.

Un tensor en TensorFlow es un vector multidimensional tipado. El almacenamiento del tipo apropiado se gestiona por un asignador que es específico del dispositivo donde reside el tensor.

En el siguiente código se puede apreciar el ejemplo de modelo de regresión multidimensional planteado con tensorFlow.

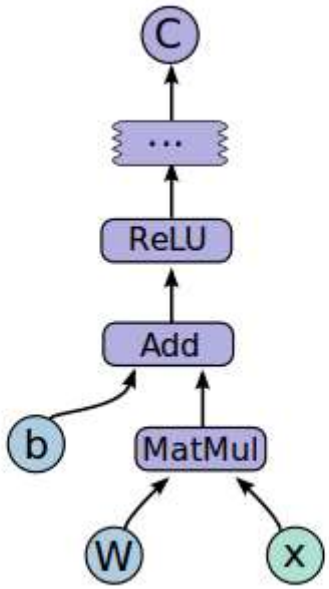
```
Import tensorflow as tf


# Vector 100-d, inicializado a ceros.
b=tf.Variable(tf.zeros([100])


# 784x100 matrix w/rand vals
W=tf.Variable(tf.random_uniform([784,100],-1,1))
# Espacio reservado para la entrada
x=tf.placeholder(name="x")
# Relu (Wx+b)
relu = tf.nn.relu(tf.matmul(W,x)+b)
# Coste calculado como una función de Relu.
C = […]
```

```
s=tf.Session()
for step in xrange(0,10):
    # Creación de un vector 100-d de entrada.
    input = … construye 100-d vector de entrad
    # Recuperación del coste, a partir de la entrada x
    result = s.run(C, feed_dict={x: input})
    print step, result
```

La computación en TensorFlow se describe por medio de un grafo dirigido, compuesto por un conjunto de nodos. El grafo representa un flujo de datos de la computación, con ciertas extensiones para permitir algunos tipos de nodos para mantener y actualizar estados persistentes y para controlar la ramificación y los bucles. La siguiente figura muestra el grafo de computación del programa anterior realizado en TensorFlow.



Grafo computacional correspondiente al código anterior

En un grafo de TensorFlow, cada nodo tiene cero o más entradas y cero o más salidas, y representa la instanciación de una operación. Los valores que fluyen desde las salidas a las entradas de los nodos son tensores. En determinadas ocasiones pueden aparecer aristas especiales en el grafo, conocidas como *dependencias de control*. En dichas aristas los datos no fluyen y se encargan de indicar que el código del nodo fuente debe de acabar su ejecución antes de que el nodo destino comience la ejecución de su código.

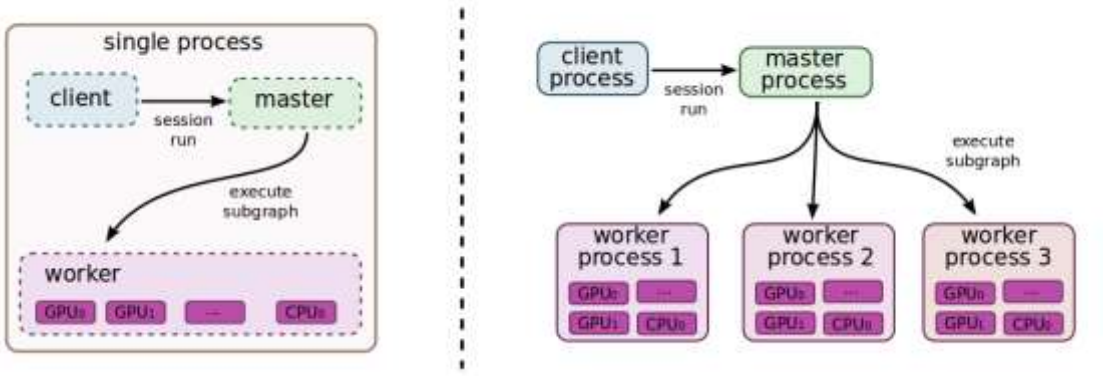
Cualquier operación tiene un nombre y representa una abstracción de cómputo (ej: multiplicación de matrices o suma de matrices). Las operaciones pueden tener atributos y todos los atributos deben proveerse en tiempo de construcción del grafo para poder instanciar

un nodo con objeto de realizar la operación. La siguiente tabla muestra un resumen del tipo de operaciones que se pueden construir con la librería de TensorFlow.

Categoría	Ejemplos
Operaciones matemáticas	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal,…
Operaciones con Array	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, …
Operaciones con Matrices	MatMul, Matrix Inverse, Matrix Determinant, …
Operaciones de Estado	Variable, Assign, AssignAdd, …
Bloques de construcción de redes neuronales	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, …
Operaciones de control	Save, Restore
Colas y operaciones de sincronización	Enqueue, Dequeue, Mutex Acquire, MutexRelease, …
Operaciones de control de Flujo	Merge, Switch, Enter, Leave, NextIteration

Los programas cliente interactúan con el sistema TensorFlow creando una *Session*. Para poder crear un grafo de computación, la interfaz de *Session* soporta un método Extend que aumenta la gestión normal del grafo con aristas y nodos adicionales, pues cuando se crea la *Session* el grafo está vacío originalmente. Otra operación inicial soportada por la interfaz es *Run*, ésta operación es la encargada de coger los nombres de las salidas que se necesitan calcular, junto a los conjuntos de tensores, para alimentar el grafo y realizar el cómputo en el orden especificado por la estructura del grafo.

Los componentes principales en un sistema TensorFlow son los clientes, que utilizan la interfaz de *Session* para comunicarse con el *master*, y uno o más *procesos workers*. Cada *proceso worker* es responsable, por un lado, de arbitrar el acceso a uno o más dispositivos de computación (CPUs, GPUs), y por otro lado, de ejecutar los nodos del grafo sobre dichos dispositivos tal y como viene indicado por el *master*. Existen dos implementaciones, local y distribuida, de la interfaz de TensorFlow. La implementación local se utiliza cuando el cliente, master y el worker se ejecutan sobre una única máquina en el contexto de un único proceso sobre un sistema operativo (pudiendo usar varios dispositivos si por ejemplo la máquina local dispone de GPUs o varias CPUs). La implementación distribuida comparte la mayor parte del código de la implementación local, pero la extiende para poder soportar entornos donde el cliente, el master y los workers puedan concurrir en diferentes procesos sobre diferentes máquinas. La siguiente figura nos muestra un esquema de estos dos modos de funcionamiento.



Estructura de máquina local y sistema distribuido para la librería TensorFlow.

En el modo de computación distribuida, una de las responsabilidades principales de la implementación de TensorFlow consiste en el mapeo de los cálculos sobre el conjunto de dispositivos disponibles. Para ello el sistema dispone de una función que estima el coste de ejecución del modelo mediante estimaciones de los tamaños de los tensores de entradas y salidas para cada uno de los nodos del grafo, junto a estimaciones de los tiempos requeridos para el cálculo en cada uno de los nodos cuando se presenta un determinado tensor de entrada. Este modelo de coste es estimado estadísticamente, bien mediante la introducción de heurísticos asociados con los diferentes tipos de operación, o bien mediante mediciones basadas en la actual localización basándose en los cálculos realizados previamente.

El algoritmo de emplazamiento corre inicialmente una ejecución simulada del grafo. A partir de esta simulación se obtiene el conjunto de dispositivos sobre los que se ejecutarán las diferentes partes del proceso basándose en una serie de heurísticos voraces. Estos heurísticos toman en consideración los tiempos de ejecución de las operaciones en cada uno de los tipos de dispositivos disponibles, y también consideran los costes de cualquier comunicación que deba de ser introducida para poder transmitir las entradas a los nodos desde otros dispositivos. Al final de la aplicación del algoritmo de emplazamiento se obtendrá el lugar donde se han de ubicar las diferentes partes durante el cómputo del modelo además de los dispositivos sobre los que se ejecutarán.

La tolerancia a fallos se implementa de tal forma que cuando un fallo es detectado, entonces la ejecución del grafo entero se aborta y se comienza de nuevo. Sobre este tema cabe comentar que los nodos que hacen referencia a variables que se refieren a tensores persisten durante la ejecución del grafo. Esto posibilita su recuperación consistente durante los diferentes reinicios del modelo.

2.3.1.1 Ejemplo de implementación con Tensor Flow

A continuación se muestra un ejemplo que hace uso de Skitlearn y tensorflow para obtener un modelo clasificador de dígitos mediante el uso de redes neuronales.

```
# coding: utf-8
```

```
# # TENSORFLOW
# Script que implementa un modelo de redes neuronales en Tensorflow para el reconocimiento de dígitos manuscritos (0-9).
# Se hace uso de las funciones que provee la librería scikit-learn para la descarga del dataset y la división del mismo en los conjuntos de entrenamiento y prueba.
# El clasificador implementado bajo Tensorflow es DNNClassifier (1)
# https://www.tensorflow.org/api_docs/python/tf/contrib/learn/DNNClassifier

# Importación de las librerías necesarias.
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from tensorflow.contrib.learn.python import SKCompat
import tensorflow as tf

# Se cargan las imágnes.
digits = load_digits()

# Se comprueban las dimensiones.
print(digits.data.shape)

# Se obtienen los datos y el objetivo de forma separada
data = digits.data
target = digits.target.reshape(-1, 1)

# Se comprueban las dimensiones de cada fuente de datos.
print("data", data.shape)
print("target", target.shape)

# Se dividen los datos en entrenamiento y test. 60% de los datos para entrenamiento y 40% para test.
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size = 0.4, random_state = 123456)

# Se obtienen las columnas de características.
feature_columns = [tf.contrib.layers.real_valued_column("", dimension = data.shape[1])]

# Se define el modelo a entrenar. En este caso se trata de una rede neuronal para clasificación con una capa oculta formada por 100 neuronas.
Model = SKCompat(tf.contrib.learn.DNNClassifier(feature_columns = feature_columns, hidden_units = (100, ),
n_classes = len(set(digits.target)), model_dir = "models"))

# Se entrena el modelo con 100 iteraciones en la estimación de los diferentes parámetros que conforman las conexiones de la red.
Model.fit(X_train, Y_train, steps = 100)

# Se obtienen las tasas de acierto para el corpus de entrenamiento y test.
accuracy_train = Model.score(x = X_train, y = Y_train)["accuracy"]
accuracy_test = Model.score(x = X_test, y = Y_test)["accuracy"]

# Se muestran las tasas por pantalla.
print("accuracy_train", accuracy_train)
print("accuracy_test", accuracy_test)
```

2.3.2 Keras

Keras[7] es un API de alto nivel para la implementación de redes neuronales escrita en Python y capaz de integrarse, en una capa inferior con TensorFlow[8], CNTK[9], o Theano[10]. Fue desarrollada con el objetivo de posibilitar la experimentación rápida. Los principios que guían a Keras son:

- **Facilidad de uso:** Keras es un API diseñado para seres humanos, y se centra en las experiencias del usuario. Sigue buenas prácticas para reducir la carga cognitiva ofreciendo APIs consistentes y simples.
- **Modularidad:** Las diferentes partes que conforman las redes neuronales se introducen como módulos como si se tratara de un puzle que va incluyendo piezas conforme a las necesidades.
- **Fácil extensibilidad:** La introducción de nuevos módulos es simple, lo que abre a Keras dentro del campo de la investigación.
- **Trabajo con Python:** Los modelos son descritos en código Python, que es compacto y fácil de depurar además de permitir la extensibilidad de forma sencilla.

2.3.2.1 Ejemplo de implementación con Keras.

A continuación se muestra un ejemplo de uso de Keras:

```
# coding: utf-8
# Keras
# Modelo Secuencial que permite añadir las diferentes partes en forma de pila.
from keras.models import Sequential
model = Sequential()

# Se añaden caps a la red:
from keras.layers import Dense
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

# Configuración del proceso de aprendizaje
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# Si se desea es posible especificar más el optimizador
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.SGD(lr=0.01,
momentum=0.9, nesterov=True))

# Entrenamiento del modelo en lotes.
model.fit(x_train, y_train, epochs=5, batch_size=32)

# Alimentación manual de lotes.
model.train_on_batch(x_batch, y_batch)

# Evaluación del modelo
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

# Predicciones
classes = model.predict(x_test, batch_size=128)
```

3 Diseño del Motor de Prognosis

El objetivo que persigue el Motor de Prognosis es el de orquestar un ecosistema de modelos estadísticos que analicen el funcionamiento en tiempo real de un proceso industrial con el fin de apoyar en el proceso de mejora de indicadores OEE como el de *Disponibilidad*, *Rendimiento* y *Calidad*.

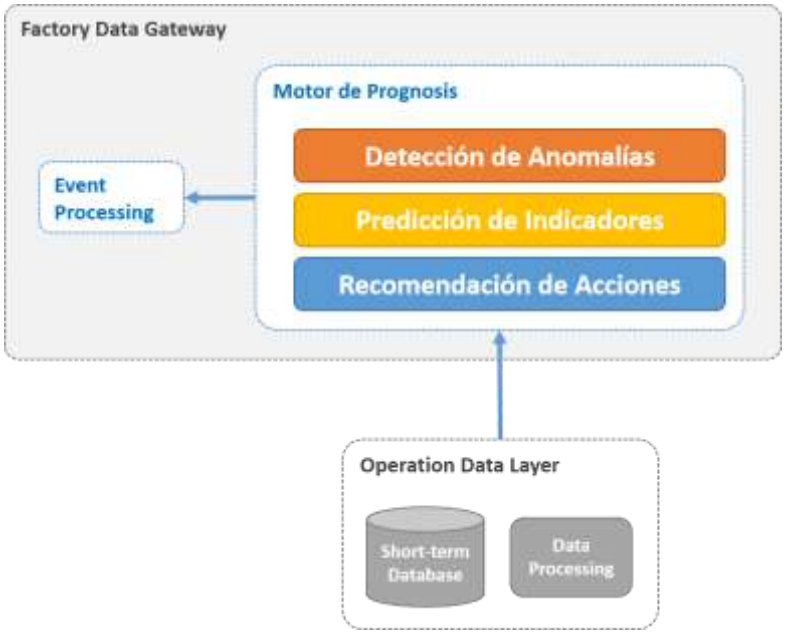
Alcance del motor de prognosis

Para ello, el Motor de Prognosis se entiende como un ecosistema de modelos estadísticos que permiten dar soporte al usuario en tres áreas descritas en la figura anterior:

- **Detección de anomalías:** permitirá modelar el comportamiento normal de un conjunto de variables, señales e indicadores para determinar cuándo obtiene un valor anómalo que deba ser informado al usuario. El enfoque multivariante permite analizar anomalías desde el punto de vista de rotura de estructura de correlación o bien de valores distantes. Estas anomalías deben ser validadas por el usuario experto para correlacionarlo con paradas no planificadas que finalmente tengan impacto en el rendimiento, calidad o disponibilidad.
- **Predicción de indicadores:** pronosticar los valores futuros de los indicadores de OEE podremos conocer mejor el funcionamiento y resultado de las acciones correctivas o la implementación de estrategias de mejora del proceso. La predicción de los indicadores no sólo tendrá en cuenta el valor de esta variable, sino el conjunto total de variables que intervienen en el proceso, correlacionando variables de sensores o señales. Junto a valores del pasado de los indicadores como el turno anterior, el día anterior, una semana o un mes, es posible contrastar tendencias.
- **Recomendación de acciones:** el resultado de este modelo permitirá recomendar acciones correctivas que tuvieron un impacto positivo en alguno de los indicadores de OEE, correlacionándolo con las anomalías detectadas en el proceso. Para ello, el SGA deberá de disponer la forma de etiquetar las acciones correctivas implementadas antes de comenzar un turno de trabajo o para atender las anomalías detectadas.

La integración del Motor de prognosis en el SGA deberá realizarse tal y como se especifica en el diagrama. Por una parte, la Operation Data Layer es la encargada de almacenar la historia reciente de funcionamiento del proceso y de proveer de los mecanismos de transformación que permita servir a los modelos estadísticos de la información necesaria para realizar su

función en tiempo suficiente. El resultado será enviado a un procesador de eventos encargado de generar una salida en forma de alerta o de valor contínuo, almacenado finalmente en la base de datos que consumirá el SGA.



Arquitectura del motor de prognosis

3.1 Detección de anomalías de funcionamiento

Los sistemas de detección de anomalías tienen como objetivo final implementar un sistema que ofrezca alertas cuando se observen patrones distintos a los aprendidos como habituales. Estas alertas permitirán prever la aparición de problemas y poder actuar a tiempo evitando así roturas futuras.

Para entender bien el funcionamiento de estos sistemas se han de introducir previamente unos conceptos estadísticos formales y el concepto, bien conocido en la industria, de gráficos de control.

En términos generales, un gráfico de control se basa en técnicas de inferencia estadística y contraste de hipótesis que confirman o refutan una hipótesis inicial (H0), que siempre supone que todo está bajo control y funcionando correctamente, por lo que no hay motivo para realizar ninguna acción correctiva, la idea de la inferencia estadística consiste en medir un estadístico conocido, para el que se conoce su distribución de probabilidad, y en base a dicha distribución refutar dicha hipótesis nula cuando existe una probabilidad baja del cumplimiento de dicha hipótesis dado el valor del estadístico observado.

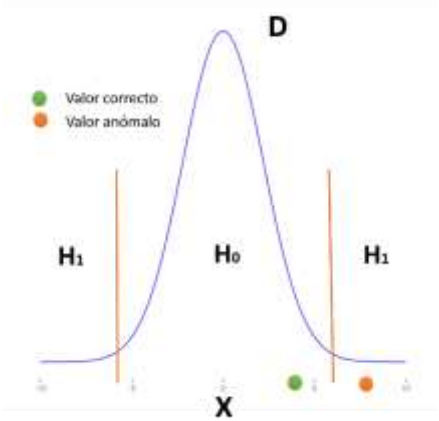
Para ello, se toma como referencia la distribución de probabilidad (D) del estadístico X y se le pregunta sobre la probabilidad que hay, asumiendo que el sensor o sensores están funcionando de manera correcta, conforme a las especificaciones, se obtenga el valor de estadístico medido. Si el modelo bajo la hipótesis nula que lo crea tiene ofrece poca probabilidad a este valor del estadístico, se supondrá falsa la asunción inicial y por lo tanto se considerará que el sensor está dando valores anómalos y ofrecerá una alerta al sistema.

Formalmente, sean las hipótesis:

- **H0:** hipótesis de partida que considera que el sensor o conjunto de sensores están funcionando bien.
- **H1:¬H0** hipótesis alternativa que considera que el sensor o conjunto de sensores no funciona conforme a lo esperado.

Sea X nuestro valor estadístico obtenido a partir de las mediciones obtenidas de los sensores durante un periodo de tiempo y sea D la distribución de probabilidad seguida por dicho estadístico cuando el sensor está bajo control y funcionando correctamente.

Dado el valor del estadístico **X** obtenido en un momento dado se le pregunta al modelo **D** sobre la probabilidad de obtener un valor del estadístico como el obtenido asumiendo que **H0** es cierta. Si dicha probabilidad es pequeña entonces ganará la hipótesis alternativa **H1** que supone que nuestro sensor o conjunto de sensores no están ofreciendo las mediciones esperadas y por lo tanto que algo anómalo está ocurriendo.



Gráficamente se puede observar como el conocimiento del modelo **D** nos permite establecer de manera objetiva una serie de límites sobre la distribución D, a partir de los valores de probabilidad aceptados como bajos por el propio usuario, y a través de estos valores de probabilidad se pueden establecer las fronteras entre la aceptación de la hipótesis H0 y el rechazo de ésta.

A partir de aquí, para disponer de un sistema de alertas basadas en un control estadístico es necesario establecer el estadístico o estadísticos X a medir (medias, desviaciones, rangos, distancias, etc.) y la distribución de probabilidad D que tienen.

El modelo que define la distribución D se aprende, durante la fase de entrenamiento, a partir de los valores históricos del sensor o sensores implicados o de algún tipo de proyección de éstos, considerando dichos valores como representativos del funcionamiento correcto (H0) que se desea modelar.

Los límites de esta distribución D los establece el usuario de forma objetiva, durante la fase de producción, asumiendo un riesgo bajo de primera especie. Dicho riesgo representa la probabilidad de equivocarse al aceptar la hipótesis alternativa dado que lo que realmente está ocurriendo en el sistema, es la hipótesis nula.

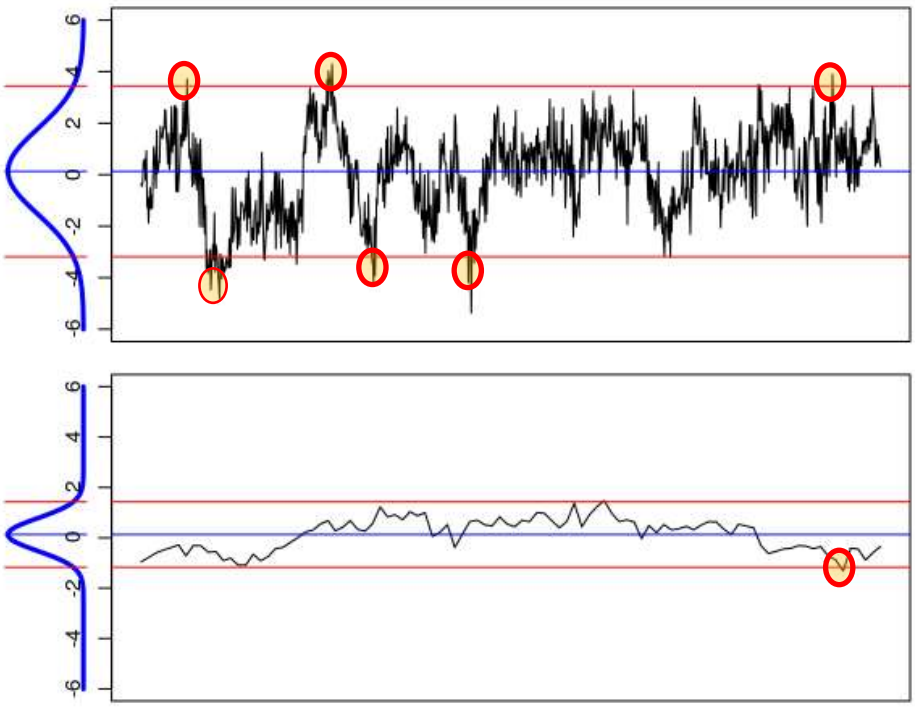
Durante la fase de producción, cada cierto periodo de tiempo previamente establecido por el modelo entrenado, se recogen los datos obtenidos y se extrae el estadístico para el que fue pensado el modelo. El valor que ofrezca dicho estadístico se contrasta con los límites de control establecidos. Si dicho estadístico se sitúa en la cola límite superior, si sólo estamos interesados en la cola superior o también la inferior, si además estamos interesados en la cola inferior, entonces el sistema ofrecerá una alerta indicando la situación anómala.

Durante el periodo de aprendizaje se usan datos históricos que representan lo que se considera con un comportamiento normal. El propio proceso de aprendizaje suele incluir determinados procesos de limpieza para eliminar de los patrones de aprendizaje ciertas anomalías y conseguir así que el modelo estimado los considere como patrones habituales.

Los modelos han de ser reentrenados cada vez que las condiciones normales del proceso productivo cambien. Existen diversas formas de evidenciar cambios de comportamiento en el proceso analizado. Una de estas formas podría usar estadísticos que den cuenta de la distancia de la nueva población de patrones de entrada respecto de la usada durante el aprendizaje del modelo en actual en producción.

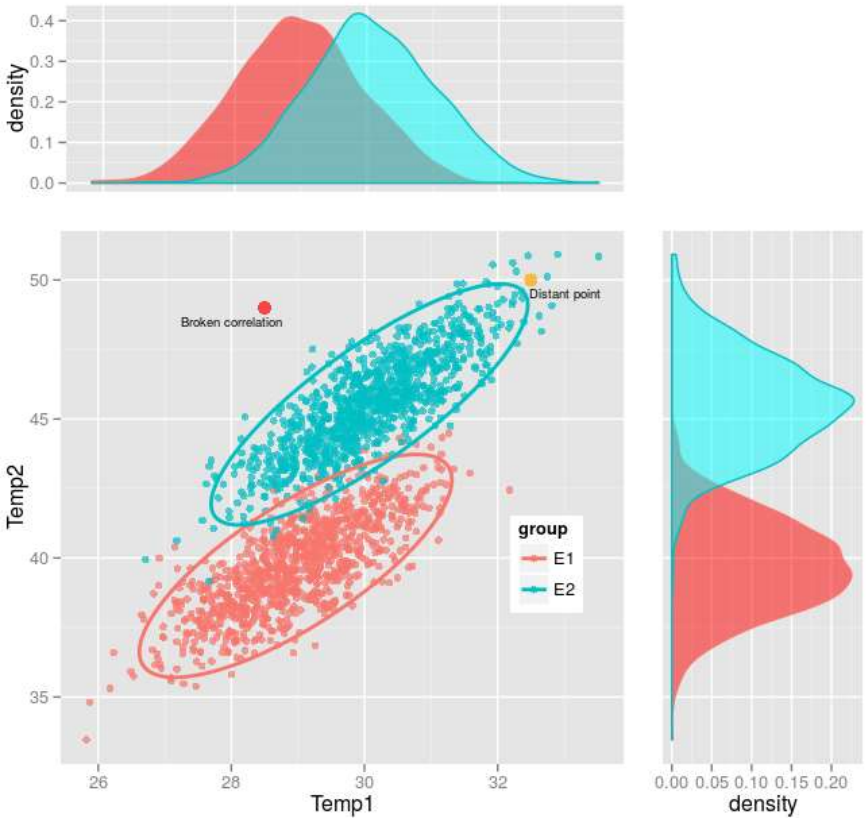
También cabe pensar en la posible especialización de estos modelos según sean los valores observados para aquellos factores que se saben que influyen en el comportamiento. Como ejemplo de estos factores podríamos pensar en el régimen, el material de trabajo en máquinas que trabajen diferentes materiales, el tipo de pieza a fabricar, etc., la implementación de modelos especializados para cada uno de los valores que puedan tomar los factores influyentes reducirá la dispersión de dichos valores, al extraer la variabilidad debida a dichos factores del modelo, lo que mejorará el poder de detección de verdaderas alertas, reduciendo a su vez la aparición de los falsos positivos.

En la siguiente figura el gráfico superior nos muestra el comportamiento natural de una señal, en él se aprecian dos límites uno superior y otro inferior a partir de estos cuales el sistema nos ofrecerá alertas. El problema es que operando de esta forma obtendríamos un número de alertas excesivo, incluso cuando no se trate de verdaderas alertas sino de comportamientos extremos pero naturales para el proceso en estudio. Los gráficos de control buscan ir más allá intentando ofrecer alertas únicamente cuando existe evidencia estadística de que dicho sistema está funcionando de forma diferente. Evidentemente, como todo modelo estadístico cometeremos un error pero dichos gráficos se establecen para cometer errores muy pequeños y lo que es más importante, en todo momento conocidos y mesurables desde un punto de vista estadístico. La figura de abajo nos muestra un gráfico de control de la media de la señal de arriba. En él se puede apreciar claramente una reducción del número de alertas (falsos positivos).



En ocasiones existen alertas que no son apreciables desde un punto de vista unidimensional, pues se producen a nivel multidimensional cuando los valores de un sensor, dejan de ser normales al considerarse en relación con los valores del resto de sensores obtenidos durante ese periodo. Esto se produce, bien porque se rompe la estructura de relación existente entre el conjunto de sensores o bien porque el punto en el espacio multidimensional está muy separado del resto de puntos.

La siguiente figura muestra gráficamente la problemática planteada arriba. En dicha figura se puede ver el comportamiento de dos sensores de temperatura (Temp1 y Temp2) en dos estados diferentes de la máquina (E1 y E2). En las figuras laterales se aprecia la distribución de las temperaturas a nivel unidimensional para cada estado de la máquina, la superior sería Temp1 mientras que la lateral representa Temp2. Por otra parte en la gráfica central se puede observar la relación existente entre ambas temperaturas a la vez. Se observa como un crecimiento de Temp1 supone un crecimiento de Temp2, y además se aprecia una repetición de este comportamiento en ambos estados de la máquina pero a diferentes niveles de temperatura. Por otra parte en la gráfica se han destacado dos puntos (rojo, naranja) que muestran datos anómalos, no detectables desde un plano unidimensional, que resaltan a nivel multidimensional. En estos se observa una rotura de correlación (rojo), valor de la Temp1 muy alto para un valor tan bajo de Temp2, y un punto distante (naranja) dentro al clúster multidimensional de puntos.



3.1.1 Autoencoders

Los autoencoders son redes neuronales artificiales que arquitectónicamente consisten en una red no recurrente similar a un perceptrón multicapa (MLP), que tiene una capa de entrada, una capa de salida y una o más capas ocultas conectando a ambas. La capa de salida tiene el mismo número de nodos que la capa de entrada. El objetivo más común de este tipo de modelos consiste en reconstruir su propia entrada.

La siguiente figura muestra la estructura general de un autoencoder. Conforme se puede apreciar se compone de dos partes bien diferenciadas: la primera sería el ***encoder*** que comienza con la entrada (input **X**) y acaba con el valor codificado (code **Z**), mientras que la segunda, ***decoder***, comienza por la codificación (code **Z**) y termina en la salida (outut **X'**). Conforme se puede apreciar los valores de **Z** se corresponderían con valores reducidos en dimensionalidad y la idea es que ante cada nueva entrada X se obtenga una salida X' que deberá de ser idéntica o casi idéntica a la entrada para una situación que sea considerada como normal.

Como el autoencoder es aprendido a partir de los valores de los sensores de una máquina cuando dicha máquina está funcionando correctamente, éstos modelos aprenden únicamente dicho funcionamiento normal y en estas circunstancias las distancias entre X y X' serán pequeñas pero se espera que cuando algo extraño ocurra en una máquina el valor de la entrada sea muy distante del valor obtenido a la salida, debido a que dicho patrón no fue aprendido por el modelo durante el proceso de aprendizaje. Así a partir del valor de distancia podremos ofrecer alertas cuando dichas distancias sean grandes.

Para realizar el gráfico de control se puede hacer uso de los valores codificados (**Z)** y evaluar sobre estos el estadístico T^2 de Hotelling como estadístico que ofrece una única medida unidimensional representativa de los valores multidimensionales. Este estadístico es una generalización multidimensional de una t-Student en el plano unidimensional, que al partir de muestras multidimensionales considera además de los valores la estructura de relación existente entre las múltiples dimensiones. Para ello, sea **Y** una variable multidimensional con m dimensiones:

$$Y = Y, \dots, Y_m \sim \mathbb{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

donde dicha variable se distribuye conforme a una variable multidimensional con m medias representadas en el vector **μ** y matriz de varianzas y covarianzas **∑**

A partir de aquí se define el estadístico media muestral $\bar{\mathbf{Y}}$ a partir de **n** muestras multidimensionales tomadas durante un periodo preestablecido de tiempo:

$$\bar{Y} = \frac{Y_1 + \cdots + Y_n}{n}$$

obteniendo a partir de aquí el estadístico:

$$t^2 = n(\bar{Y} - \mu)'\Sigma^{-1}(\bar{Y} - \mu)$$

donde $t^2 \sim T^2_{m,n-1}$ o equivalentemente $t^2 \sim \frac{n-m}{m(n-1)} F_{m,n-m}$ Esto nos identifica claramente la distribución de probabilidad del estadístico, que nos permitirá establecer los límites de control en función de un riesgo probabilístico asumido por el usuario.

El valor de μ se estima a partir de la media muestral además de la matriz de varianzas-covarianzas Σ que se estima también a partir de los valores muestrales durante el proceso de entrenamiento del modelo como:

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})(Y_i - \bar{Y})'$$

Otro estadístico multidimensional a evaluar es el error del modelo SPE. Calculado este como la diferencia entre el valor real menos el valor predicho.

Mientras que en valor elevado de T^2 nos dará indicios de punto distante, un valor elevado del estadístico SPE nos informa sobre posible rotura en la correlación de los valores o sensores.

3.2 Predicción de indicadores OEE

Los modelos de predicción son funciones que de forma general toman una serie de evidencias y en base a dichas evidencias predicen el valor una variable objetivo. Matemáticamente y de forma general pueden denotarse como:

$$Y = f(x_1,\ldots,x_m)$$

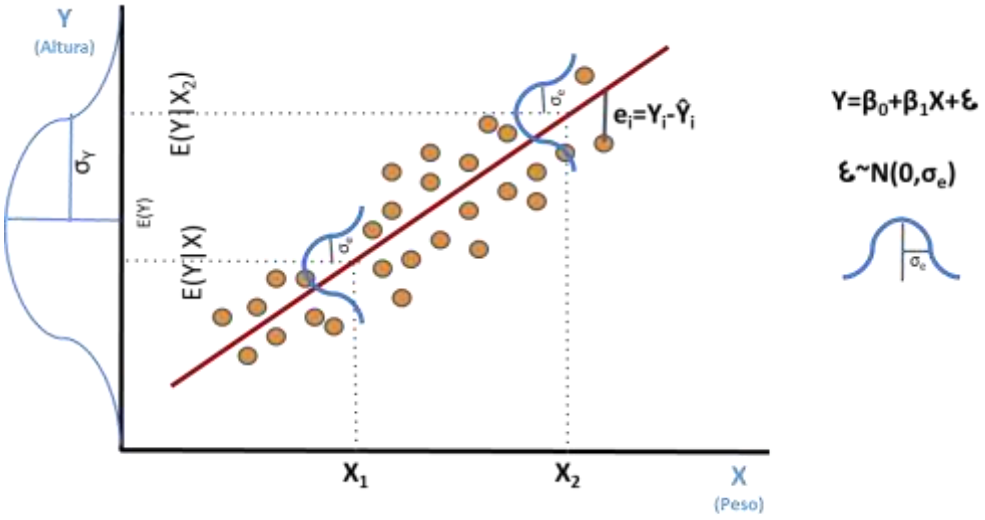
Existe una gran infinidad de modelos de predicción, como pueden ser árboles de decisión, máquinas de soporte general, modelos de regresión, etc.

De entre todos los existentes vamos a escoger los modelos de regresión como ejemplo para poder ofrecer una explicación de las diferentes partes que los componen. La siguiente fórmula nos muestra la estructura general de un modelo de regresión:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m + \varepsilon$$

donde:

- **y**: Variable objetivo.
- $\mathbf{x_i}$: Variable independiente o evidencia.
- β_i: Son los parámetros del modelo resultado del aprendizaje durante la fase de estimación.
- $\boldsymbol{\varepsilon}$: Es el error que comete finalmente el modelo en sus estimaciones y que se distribuye conforme a una distribución Normal con media igual a 0 y desviación típica σ_e ($N(0,\sigma_e)$).



La anterior figura nos indica que si deseamos predecir el valor de Y pero no tenemos ninguna información sobre algún otro aspecto, entonces la mejor estimación será el valor esperado de Y (E(Y)), pero el error que se cometerá será bastante mayor (σ_y) que si se dispone de la evidencia X y de un modelo que relacione las características X con la Y deseada, en cuyo caso el error de estimación se corresponderá con (σ_e). Así pues, conforme menor sea el valor σ_e, mejor será el error de predicción del modelo y mejores predicciones nos ofrecerá éste. La idea por lo tanto consiste en buscar aquellas características X, que nos ayuden a predecir Y con objeto de ir bajando σ_e.

Dependiendo de la naturaleza de la variable hablaremos de:

- **Modelo de predicción**: cuando la variable **y** pertenece al conjunto de los números naturales (Z) o al conjunto de los números reales (R).
- **Modelo clasificador**: cuando la variable y representa a dos clases (0,1), en cuyo caso se conoce como modelo binomial, o representa a n clases (1,2,…,n-1,n), denotándose en este caso como modelo multinomial.

Las series temporales son modelos de predicción que tienen como objetivo averiguar los valores futuros de una variable en función de sus valores pasados. De forma general, estos modelos pueden expresarse como una función que toma como variables explicativas los valores de la variable a predecir en instantes pasados y como variable objetivo el valor de la variable en un instante futuro:

$$x_{t+1} = f(x_t, x_{t-1,}\ldots,x_{t-m})$$

dicha función expresada como un modelo de regresión quedaría:

$$x_{t+1} = \beta_0 + \beta_1 x_t + \beta_1 x_{t-1} \ldots + \beta_m x_{t-m} + \varepsilon$$

Existen diversas técnicas que permiten la modelización de series temporales pero de entre todas las existentes queremos aquí destacar dos, una que aplica técnicas estadísticas clásicas y otra basada en redes neuronales:

- ARIMA: Modelos autoregresivos de media móvil.
- LSTM: Long Short Term Memory.

3.2.1 ARIMA

Los modelos ARIMA, o también conocidos como modelos autoregresivos integrados de media móvil, son modelos cuya fórmula matemática final ofrecen una descripción de un proceso estocástico estacionario en relación a dos polinomios, uno relacionado con la parte autoregresiva de una señal (AR) y otro relacionado con la media móvil (MA). Mientras que la parte AR supone la regresión de la propia variable desfasada en el tiempo, la segunda parte, la de la Media móvil (MA), supone la regresión del término de error como una combinación lineal de los términos de error ocurridos durante diferentes instantes pasados. Los modelos ARIMA, también permiten la eliminación de la tendencia de ahí la I que aparece en su denominación.

Estos modelos se pueden aplicar a la vez, tanto sobre la parte regular de la señal como sobre la parte estacional, con lo que permiten la conversión de una señal en estacionaria tras descontar todos los efectos, tanto regulares, como estacionales además de sus posibles tendencias.

Si se denota a una señal como un modelo AR(p), tendremos un modelo autoregresivo de orden p, cuya notación matemática resultante sería:

$$x_t = c + \sum_{i=1}^{p} \varphi_i x_{t-i} + \varepsilon_t$$

donde $\varphi_1,\ldots,\varphi_p$ son los parámetros autoregresivos del modelo AR(p), c es una constante, y ε_t el ruido blanco.

Por el contrario, si denotamos a un modelo como MA(q), tendremos un modelo de media móvil de orden q, cuya notación matemática resultantes sería:

$$x_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

donde $\varphi_1,\ldots,\varphi_p$ son los parámetros autoregresivos del modelo MA(q), μ es el valor esperado de x_t, y ε_t, $\varepsilon_{t-1,\ldots}$ son los términos relacionados el ruido blanco.

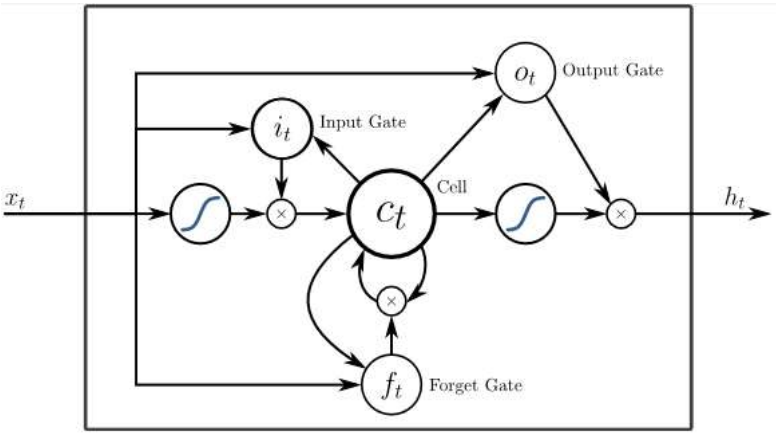
También podemos encontrar un modelo ARMA(p,q), un modelo de este tipo contendrá p términos autoregresivos y q términos de media móvil, y estará formado por la suma de un modelo AR(p) y un modelo MA(q).

$$x_t = c + \sum_{i=1}^{p} \varphi_i x_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

Si además se introduce un número de diferenciaciones mayor que cero a la señala original con objeto de eliminar además la tendencia entonces tendremos lo que se conoce como modelo ARIMA.

3.2.2 Redes Neuronales LSTM

Este tipo de redes recurrentes se conforma con neuronas conocidas como Long short-term memory (LSTM). Dichas neuronas se componen de una celda, una compuerta de entrada, otra de salida y una relacionada con el olvido de los datos observados anteriormente. La celda se encarga de recordar los valores pasados. El resto de compuertas están pensadas como neuronas artificiales comunes y de manera intuitiva pueden ser vistas como reguladores del flujo de valores que fluyen a través de las conexiones del LSTM. El término Long short-term se refiere al hecho de que pueden recordar evidencias pasadas que pueden durar un periodo de tiempo largo, lo que las dota de un gran potencial a la hora de modelizar problemas de series temporales.



Diseño de celda LSTM[3]

Las ecuaciones compactas de este tipo de red son:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c c_{t-1} + b_c)$$

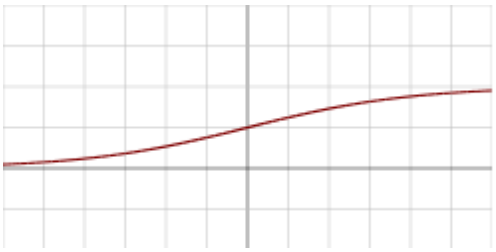
$$h_t = o_t \circ \sigma_h(c_t)$$

donde:

- σ_g: es la función de activación sigmoide.
- σ_c: es la función de activación tangente hiperbólica.

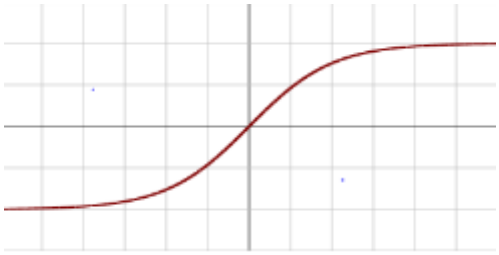
- σ_h: es la función de activación lineal.

Seguidamente se muestran las gráficas con las tres funciones de activación mencionadas anteriormente.



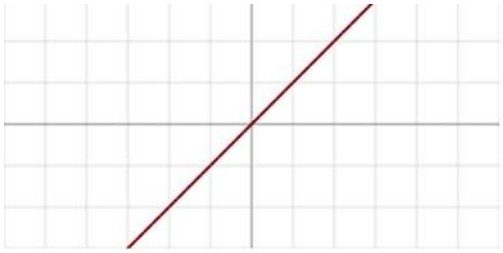
Función sigmoide[4,5]

$$\sigma_g = \frac{1}{1 + e^{-x}}$$



Función tangente hiperbólica[5,7]

$$\sigma_c = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Función lineal[5,6]

$$\sigma_h = x$$

3.3 Recomendación de acciones de mantenimiento

3.3.1 Introducción

Un motor de recomendación viene determinado por 3 características [1]: el algoritmo, la interacción del usuario, y los modelos de usuario. Las cuatro técnicas principales de filtrado que se utilizan en dichos sistemas de recomendación son:

Esquema de técnicas de filtrado.

Los sistemas de recomendación habitualmente requieren tratar con información tanto abstracta como objetiva. Existen diversas técnicas que se pueden aplicar para crear sistemas de recomendación que pueden dividirse según diferentes categorías. Los sistemas de recomendación basados en el contenido intentan realizar recomendaciones de objetos similares a aquellos seleccionados por el usuario en el pasado, sin embargo los sistemas de recomendación colaborativos se encargan de identificar usuarios con gustos similares, a un usuario dato, y recomendar los objetos que los usuarios con gustos parecidos han seleccionado[8]. Los sistemas recomendadores basados en el conocimiento realizan las recomendaciones en base a una serie de reglas establecidas de forma explícita por un experto humano. Por último los sistemas híbridos intentan combinar los tres sistemas anteriores de diversas formas, para con ello, mejorar en las recomendaciones tomando lo bueno de cada parte que se combina e intentando paliar los diferentes problemas que cada cual plantea.

3.3.2 Filtros colaborativos

Los filtros colaborativos [5] son un método automático para realizar predicciones que hablen acerca de los intereses de un usuario. Para ello el modelo aprende a partir de la similitud de unos individuos con otros en función de las preferencias. En otras palabras los filtros colaborativos suponen que si un individuo S tiene el mismo gusto que un individuo T, respecto a ciertos objetos, entonces es más probable que S tenga el mismo gusto que T en otros objetos que otro individuo que sea más distante en los gustos.

Existen dos métodos principales utilizados en los filtros colaborativos. El primero de los enfoques se basa en el recuerdo a través de la memoria, y en él se utilizan las preferencias de los usuarios, almacenadas éstas en una base de datos para hacer una recomendación. Un segundo enfoque es el basado en el modelo, este tipo de enfoque parte de datos históricos a partir de los cuales se infieren los modelos que se encargarán de ofrecer las recomendaciones. En el primer caso un método de uso común es el basado en el vecino más cercano, mientras que para el segundo caso se suele aplicar la Factorización de matrices.

La ventaja principal de los filtros colaborativos consiste en que no requieren de conocimiento previo, pero a cambio tienen la desventaja de que al principio no existe posibilidad de recomendar pues no existe historia de usuario.

3.3.3 Clustering

El *clustering* es un proceso de agrupación de objetos basado en una medida de similitud. Desde este punto de vista un miembro que pertenece a un clúster es más parecido entre sí con los miembros de su clúster que con cualquier otro miembro de otro clúster.

El clustering tiene dos enfoques diferentes: el primero conocido como *hard clustering* se produce cuando los individuos se clasifican en un único clúster, en el segundo de ellos conocido como *soft clustering* o *fuzzy clustering* las observaciones se asignan a varios clústers de forma que un miembro puede pertenecer a varios de ellos, en diferente grado según los valores de simulitud, y no a uno único.

En la realidad que nos envuelve es bastante frecuente la bimodalidad, entendida ésta como la interacción conjunta entre dos tipos de entidades, por ejemplo la preferencia de un usuario por un documento en una búsqueda está afectada tanto por las características del usuario como por las características del documento. Con frecuencia estas características bimodales se representan mediante el uso de una matriz en las que cada dimensión representa uno de los diferentes tipos de entidades.

El coclústering (o biclústering) es un término que relaciona a un clústering simultáneo de filas y columnas en una matriz. Mientras que las técnicas de clústering clásicas asumen sobre la pertenencia de un objeto a un clúster depende únicamente de su similitud con otros objetos del mismo tipo, las técnicas de coclústering pueden verse como métodos que permiten la agrupación de dos tipos de entidades diferentes, basándose para ello en la similitud de las interacciones. Si suponemos un ejemplo en el que los usuarios seleccionan documentos entonces estos usuarios estarían representados en las filas de la matriz mientras que las columnas de ésta representarían a los diferentes documentos. Cada una de las celdas de esta matriz contendría como información objetiva el grado de similitud existente entre el usuario y el documento, resultado de la intersección de fila y columna.

Así por ejemplo, suponiendo que dos usuarios se parecen, si la búsqueda de los documentos que realizan durante el día son parecidas, entonces podría extrapolarse sobre un tercer tipo de entidad como puede ser la compra de un móvil, a partir de esta información conocida de las búsquedas de documentos durante el día, pues conociendo el último móvil que ha comprado uno de ellos se le puede sugerir al otro.

3.3.4 Vecino más cercano

Para poder realizar un recomendador basado en el vecino más cercano se requiere de una matriz donde las filas representen a los usuarios y las columnas a los elementos que se evalúan para cada usuario. Las celdas de dicha matriz contendrán valores objetivos sobre la elección de dichos elementos por parte de cada usuario.

El proceso consiste en buscar los usuarios más cercanos a un usuario dado y recomendar al nuevo usuario los elementos de los usuarios más próximos que no han sido elementos seleccionados por el nuevo usuario.

Este algoritmo requiere de una función de similitud, que ofrecerá una medida objetiva de la distancia entre un usuario y los usuarios más cercanos. Esta **función de similitud** ofrece el orden de proximidad de los diferentes usuarios a un usuario dado, lo que se convierte en un objetivo y relevante a la hora de ofrecer las recomendaciones adecuadas, pues cuanto más

próximos estén los usuarios mejores serán, a priori, las recomendaciones ofrecidas para el nuevo usuario.

Otra, es la **función de predicción**, que nos va a permitir combinar el resultado anterior con el fin de predecir las puntuaciones de las recomendaciones para el elemento activo.

A continuación se muestra un ejemplo de ambas funciones:

Supongamos que $\mathbf{P = \{p_1,\ldots,p_n\}}$ es el conjunto de personas, que $\mathbf{E = \{e_1,\ldots,e_m\}}$ es el conjunto de elementos, y **G** es una matriz de ***nxm*** evaluaciones $\mathbf{g_{i,j}}$, donde $\mathbf{i \in 1\ldots n}$, $\mathbf{j \in 1\ldots m}$, **x** e **y** son dos usuarios, y $\bar{\mathbf{g}}_{\mathbf{x}}$ la evaluación media del individuo x.

Función de Similitud: Correlación de Pearson

$$\mathcal{S}(x,y) = \frac{\sum_{e\in E}\left(g_{x,e} - \overline{g_x}\right)\left(g_{y,e} - \overline{g_y}\right)}{\sqrt{\sum_{e\in E}\left(g_{x,e} - \overline{g_x}\right)^2}\sqrt{\sum_{e\in E}\left(g_{y,e} - \overline{g_y}\right)^2}}$$

Función de Predicción: Evaluación del usuario **x** sobre el elemento **e**

$$\mathcal{P}(x,e) = \overline{g_x} + \frac{\sum_{y\in E}\mathcal{S}(x,y)\left(g_{y,e} - \overline{g_y}\right)}{\sum_{y\in E}\mathcal{S}(x,y)}$$

Si en lugar de trabajar con usuarios, trabajásemos con fallos sobre máquinas y acciones de mantenimiento realizadas sobre una máquina cuando se ha producido un fallo determinado, entonces el algoritmo recomendaría las acciones de mantenimiento a realizar ante la presencia de un nuevo fallo sobre una máquina nueva en base a las acciones realizadas sobre fallos y máquinas similares cuando se produjeron fallos sobre dichas máquinas. A continuación [6,7] se muestra el pseudocódigo de un algoritmo que establece los k mejores objetos para un usuario basándose en un KNN.

Esto mismo es extrapolable a otros casos de uso como por ejemplo la salud. Si disponemos de una serie de pacientes a los que ante la presencia de un problema se les ha aconsejado una serie de medidas paliativas que han mejorado su estado de salud. Entonces es posible recomendar a un nuevo paciente una serie de medidas paliativas en base al conocimiento aprendido de los pacientes más próximos a él en similitud.

Algorithm 1: General outline of neighborhood algorithms

input : Number of items to be recommended $N \in \mathbb{N}$,
Number of neighbors used for ranking $k \in \mathbb{N}$,
User to recommend items to u,
List of all items $Items$,
User-Item matrix of ratings R

output: N items to be recommended

foreach $item \in Items$ **do**
  if $item \notin u.rated\_items$ **then**
    $item.rank \leftarrow$ rank_according_to_nearest_neighbors$(k, u, item)$

descending_rank_sort$(Items)$

return top$(N, Items)$

El siguiente algoritmo establece la recomendación para cada usuario basándose en los objetos a recomendar.

Algorithm 2: Weighted-Rules Recommendation

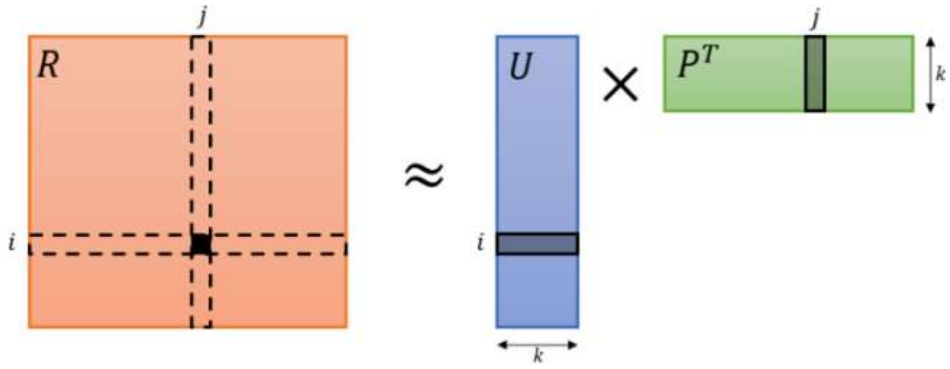
input : Set of train users $\mathcal{U}$, Test user $U \in \mathcal{U}$, Set of association rules $\mathcal{R}$, Number of items to recommend $N \in \mathbb{N}$

output: Top-N recommendations $R(U) \subseteq \mathcal{I}$, $|R(U)| \leq N$

$\mathcal{R}^+ \leftarrow \{(X \Rightarrow Y) \in \mathcal{R} \mid X \subseteq U\}$
$C \leftarrow$ init_table()
for $(X \Rightarrow Y) \in \mathcal{R}^+$ **do**
  foreach $i \in (Y \setminus U)$ **do**
    if $i \notin C$ **then**
      $C[i] \leftarrow 0$
    $C[i] \leftarrow C[i] +$ measure$((X \Rightarrow Y), \mathcal{U})$
$S \leftarrow$ descending_sort_by_value(C)
$R(U) \leftarrow \emptyset$
for $i \leftarrow 1$ **to** N **do**
  $R(U) \leftarrow R(U) \cup \{S[i]\}$
return $R(U)$

3.3.5 Factorización de matrices (MF)

Uno de los algoritmos más populares a la hora de resolver problemas de *coclústering* es la factorización de matrices (MF), que en su forma más sencilla se asume una matriz $R \in \mathbb{R}^{mxn}$ de medidas donde disponemos de m usuarios y n objetos. La aplicación de esta técnica sobre la matriz R nos llevará finalmente a una descomposición de R en dos matrices $U \in \mathbb{R}^{mxk}$ y $P \in \mathbb{R}^{nxk}$. La multiplicación $R \approx UP$.



Este algoritmo introduce un nuevo parámetro, k, conocido como el rango de la factorización y que es una de las dimensiones de las nuevas matrices U, P. Formalmente, cada celda $R_{i,j}$ se factoriza a partir del producto escalar $u_i \cdot p_j$ siendo $u_i, p_j \in R^k$. Se assume así que cada valor en las diferentes celdas de R está afectado por k efectos. Además estos k factores están relacionados con los usuarios y los objetos respectivamente a través de estos k factores.

Desde un punto de vista algebraico para obtener la factorización de la matriz original R vamos a buscar las matrices U,P que minimicen la siguiente función objetivo:

$$J = \|R - UP^T\|_2 + \lambda(\|U\|_2 + \|P\|_2)$$

El primer término en la función objetivo corresponde con el error cuadrático medio (MSE) de la distancia existente entre la matriz original R y su aproximación a través de la expresión UP^T. La segunda parte corresponde con un término de regularización que se introduce con objeto de no producir sobreaprendizaje en el modelo y minimizar además el efecto de la presencia de datos anómalos.

Este problema de optimización se resuelve mediante técnicas de Machine Learning [9,10] y algoritmos de optimización [11,12,14]. Cabe destacar aquí que la aplicación de la factorización de matrices nos ha introducido dos nuevos hiperparámetros: un primer parámetro k, relacionado con el número de factores que suponemos que explican o influyen en el objetivo buscado y un segundo parámetro λ que influye en la importancia de los diferentes individuos a la hora de minimizar la función objetivo.

Algorithm 3: Matrix Factorization based Recommendation

input : Set of users U, items Y,
Matrix of users-items rankings R,
Number of latent features l,
Number of items to be recommended $N \in \mathbb{N}$,
output: Top-N recommendations $R(U) \in \mathcal{I}^N$

for $i \leftarrow 1$ **to** numIterations **do**
  foreach *user* $\in U$ *and item* $\in Y$ *with rating* $R[u,i]$ **do**
    $predicted\_rating = U[u,] * M[i,]^t$
    $err = R[u,i] - predicted\_rating$
    $U[u,] = adjust\_by\_gradient\_descent(U[u,])$
    $M[i,] = adjust\_by\_gradient\_descent(M[i,])$

return N items with highest predicted ranking

3.3.6 Mínimos cuadrados alternados (ALS)

Tomando la función objetivo J, se puede observar que debemos de aprender dos tipos diferentes de variables, las de U y las de P. El hecho de que tanto U como V sean valores desconocidos convierte a la función J en una función no convexa. Sin embargo si una de las matrices se fija entonces el problema se reduce a un problema de regresión lineal. Así pues, el algoritmo ALS busca las ventajas de esta propiedad, realizando la estimación de ambas matrices en un proceso iterativo compuesto de dos fases: en una primera fase se fijan los valores de P y estiman los valores de U y en una segunda fase son los valores de U los que se fijan y se obtienen los de P. Evidentemente este algoritmo caerá en un óptimo local de la función J cuyo valor de importancia estará influido por los valores que inicialmente se hayan impuesto a las matrices U y P.

3.3.7 Descomposición de valores singulares frente a la factorización de matrices (SVD vs MF)

SVD [9] es una técnica de descomposición de matrices que descompone cualquier matriz en 3 matrices $U \in R^{mxk}$, $\Sigma \in R^{kxk}$ y $V \in R^{nxk}$ tal que $A=U\Sigma V^T$.

La descomposición SVD tiene una serie de características que la hacen muy apropiada:

1. Σ es una matriz diagonal, cuyas entradas se corresponden con la fortaleza de cada uno de los k factores. Como los k factores obtenidos están ordenados de mayor a menor variabilidad explicada, esta información es útil a la hora de establecer el número idóneo, k de factores a considerar.
2. U es una matriz ortonormal (sus columnas son ortogonales y su norma vale 1) que en el ejemplo expuesto arriba se encargaría de relacionar a los usuarios con cada uno de los k factores.
3. V es una matriz ortonormal que relacionaría a los documentos con los factores.
4. La descomposición SVD ofrece una solución única.

Esta aproximación presenta dos problemas:

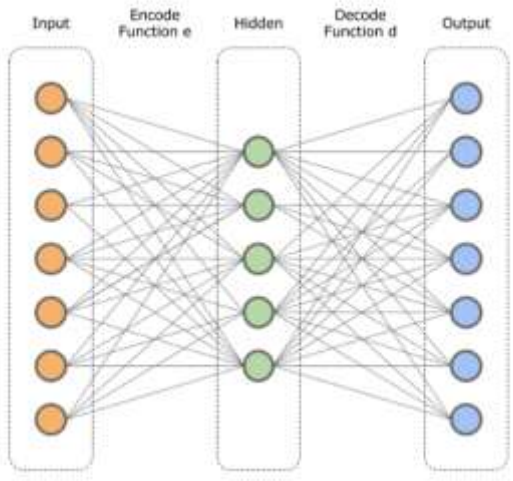
1. La complejidad del cómputo.
2. Para poder realizar su cálculo la matriz ha de ser densa sin valores faltantes.

Sin embargo a pesar de las dos problemáticas comentadas, actualmente es posible ofrecer soluciones a ambas aplicando algoritmos de cálculo distribuidos basados en Big Data [20] y solucionar el problema de las matrices dispersas mediante la aplicación de técnicas de imputación de datos faltantes [15,16,17,18,19].

3.3.8 Deep autoencoders como filtros colaborativos

Un autoencoder es una arquitectura de red neuronal que alcanza el estado del arte en el área de los filtros colaborativos. Arquitectónicamente una Autoencoder es una rede neuronal alimentada hacia delante que tiene una capa de entrada, una capa oculta y una capa de salida. La capa de salida tiene el mismo número de neuronas que la capa de entrada con el propósito de reconstruir en la salida la propia entrada introducida en un momento dado. Esto hace de los autoencoders un sistema de aprendizaje no supervisado.

Una de las características de los autoencoders consiste en que la capa oculta tiene un número de neuronas menor que la capa de entrada. Esto fuerza la creación de una representación comprimida o reducida de los datos en la capa oculta.



Arquitectura clásica de un autoencoder[22]

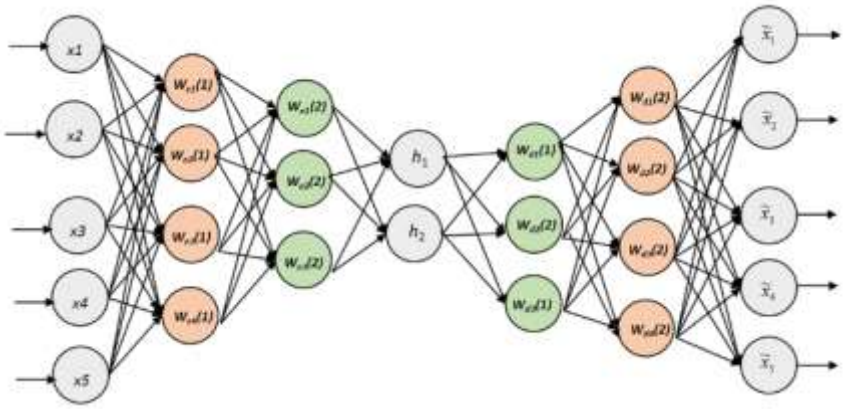
La transición desde la entrada a la capa oculta se conoce como la fase de codificación y la transición que va desde la transición de la capa oculta a la capa de salida se conoce como la etapa de decodificación. Esto queda representado matemáticamente como:

$$\phi: X \to Z: x \mapsto \phi(x) = \sigma(Wx + b) = z$$

$$\varphi: Z \to z: z \mapsto \varphi(z) = \sigma\left(\widetilde{W}z + \tilde{b}\right) = x'$$

La asignación se realiza multiplicando el vector de datos de entrada **x** por una matriz de pesos, adicionando un sesgo **b** y aplicando al vector resultante una función de activación **σ**.

La extensión natural de los Autoencoders simples son los Deep Autoencoders, cuya diferencia fundamental con los primeros radica en un mayor número de capas ocultas que permiten aprender relaciones entre la entrada y la salida mucho más complejas.



Arquitectura Deep Autoencoder[22]

Los filtros colaborativos, de manera general, pueden verse como métodos de rellenado de datos faltantes, cuando las variables medidas, o parte de éstas, tienen que ver con las recomendaciones buscadas. Para ello es interesante que para realizar dicha imputación se tome en consideración el contexto no faltante a la hora de imputar dichos datos para imputar siempre según la correlación observada con el resto de características, de las cuales en un momento dado si que se dispone de evidencias. Cuando las características, o algunas de éstas, tienen que ver con el tipo de recomendaciones buscadas entonces dicho rellenado puede verse como un filtro colaborativo.

La arquitectura planteada por un autoencoder, y el objetivo de su aprendizaje, los hace ideales para el fin aquí planteado.

3.3.9 Sistemas basados en el contenido

Los sistemas basados en el contenido recomiendan elementos al usuario basándose en su perfil, su comportamiento pasado y la característica de los elementos a seleccionar.

El perfil del usuario se puede obtener a partir de un histórico y de sus interacciones con el sistema. También se puede obtener preguntando explícitamente al usuario sobre sus intereses y preferencias, en este caso pueden ser de utilidad los métodos AHP y PROMETHEE que se comentan bajo, en el anexo.

Tal y como se comenta en [9] los sistemas basados en el contenido crean recomendaciones más personalizadas incluyendo el perfil del usuario y la descripción del elemento. Este tipo de sistemas pueden generar recomendaciones sobre elementos que no hayan sido nunca evaluados, cosa que no ocurría con los basados en filtros colaborativos.

Los sistemas basados en el contenido requieren de técnicas apropiadas a la hora de representar los objetos y producir los perfiles de usuario, además de técnicas que permitan la comparación de los perfiles con los objetos. El proceso de recomendación se realiza en tres pasos, cada uno de los cuales se realiza por un componente separado:

1. **Analizador del contenido:** Este componente tiene como objetivo extraer características a partir de información desestructurada.
2. **Descubridor de perfiles:** Se encarga de recolectar datos representativos de las preferencias del usuario para conformar su perfil a partir de la aplicación de técnicas de Machine Learning [10].
3. **Componente de filtrado:** Se encarga de ofrecer objetos de interés a partir de la información obtenida del perfil.

En este tipo de sistemas, el aprendizaje de las recomendaciones se aprenden a través de modelos que se aprenden de forma automática mediante el uso de algoritmos de Machine Learning [9].

El aprendizaje en este tipo de sistemas se realiza en dos fases. En una primera fase, conocida como **fase de aprendizaje**, se aprenden los modelos de interés de forma automática a partir de las patrones del contexto extraído del contenido y de los perfiles de usuario que pueden estar presentes en una base de datos o corpus de entrenamiento. En una segunda fase se utilizarán dichos modelos para predecir las recomendaciones ante la entrada de nuevos individuos.

Este tipo de sistemas puede verse desde dos puntos de vista según sea el objetivo que persiguen. El primer punto de vista sería el de **pronóstico**, que consistiría en realizar predicciones sobre anomalías o valores objetivos que resulten de interés a la hora de alertar sobre algún aspecto como por ejemplo el interés de un usuario por un documento, película o el hecho de que una máquina falle desde un punto de vista industrial. El segundo punto de vista es el de **prescripción** que consistiría en la obtención de una receta o consejo que permita mejorar o aconsejar sobre el objetivo perseguido y que se está pronosticando por los modelos de pronóstico como anómalo o extraño.

La receta resultante de la **prescripción** sería lo que se entiende como recomendación, pues su objetivo será sugerir las acciones a realizar, para mejorar el objetivo perseguido, en base al contexto observado y predicho por los modelos de pronóstico y la información de contexto.

Para poder evolucionar del pronóstico a la prescripción una posibilidad sería hacer uso de simuladores y de optimizadores que actúen sobre los factores de dichos simuladores. En este caso los optimizadores se encargarían de cambiar dichos factores que corresponderían con aquellos parámetros de configuración que un usuario real podría cambiar en una máquina real, si estamos en un entorno industrial, y observar que resultado objetivo se produciría sobre la máquina simulada. Los optimizadores actuando de esta forma se encargarían de buscar la configuración óptima que maximiza o minimiza, según sea mi interés, la función objetivo buscada, y como resultado ofrecerían una recomendación/receta con los valores correctos de configuración de la máquina.

La posibilidad de disponer de un simulador, permite que los algoritmos de optimización interactúen con dicho simulador y no con la máquina real, lo que muchas veces podría suponer un gasto prohibitivo e inabordable. Estos optimizadores se encargarían de buscar los valores adecuados de los parámetros de configuración de la máquina real, que mejoren las funciones objetivo deseadas, interactuando sobre estos simuladores y no sobre la máquina real. Al final estos algoritmos ofrecerían los valores de los parámetros de configuración de la máquina (prescripción/receta) con los que se ha encontrado un mínimo o máximo de la función o funciones objetivo deseadas. Como optimizadores podría hacerse uso, entre otros, tanto de algoritmos bioinspirados [11,12] como de algoritmos de aprendizaje por refuerzo [13].

3.3.10 Sistemas basados en el conocimiento

Los recomendadores basados en el conocimiento requieren de la construcción de modelos a partir de dicho conocimiento. En este tipo de sistemas existen dos tipos aproximaciones bien diferenciadas que dependen del grado de automatización en que dicho conocimiento es modelado. Así hablaremos de sistemas expertos cuando el conocimiento es inferido a través de reglas introducidas manualmente por un experto humano, mientras que hablaremos de sistemas de aprendizaje automático cuando dicho conocimiento se infiere a partir de patrones, habitualmente etiquetados con los objetivos finalmente deseados. A diferencia de los sistemas anteriores éstos ya parten de un conocimiento previo por lo que no sufren del problema de inicialización presente en los tipos de sistemas comentados anteriormente.

Este tipo de sistemas está pensado para ser integrados con cualquiera de los otros creando así sistemas híbridos que contemplan dos o más enfoques diferentes, mediante la combinación de sistemas.

Por contra, puesto que este tipo de sistemas suponen a priori la disponibilidad de un corpus de conocimiento, suele ser necesario un proceso previo, y habitualmente pesado, de recolección, fusión y depuración de los datos de entrada, lo que requiere esfuerzo en tiempo y recursos. Además para el caso específico de los sistemas expertos se añade el inconveniente del esfuerzo requerido a la hora de mantener y actualizar las diferentes reglas existentes por parte del experto humano ante nuevas evidencias o diferentes regímenes de funcionamiento si por ejemplo se está hablando de máquinas.

Este tipo de sistemas requieren un gran esfuerzo en la introducción del conocimiento de forma explícita. Los desarrolladores encargados de estos tipos de sistemas deben de dedicar grandes esfuerzos para, por una parte alimentar el sistema de información con las diferentes evidencias que se producen y por otra parte establecer las diferentes reglas que toman como entrada las diferentes evidencias presentes en la fuente dicha información y ofrecen recomendaciones en base a dichas evidencias.

En este tipo de sistemas las reglas se aprenden a partir de una base de conocimiento y la experiencia de un experto. En ocasiones puede requerirse también de información por parte del usuario para la que los métodos AHP y PROMETHEE, comentados en el anexo posterior, pueden ser de gran utilidad.

Los avances actuales van en la línea de automatizar el proceso de aprendizaje de las reglas y restricciones a partir de la base del conocimiento, lo que reduciría enormemente el esfuerzo necesario a la hora de trabajar con este tipo de aproximaciones. Para poder crear dichos sistemas expertos automáticos existen diferentes técnicas que pueden utilizarse entre las que cabe destacar las aproximaciones bioinspiradas que combinan técnicas gramaticales con programación genética [11,12] o árboles de decisión [9].

3.3.11 Otras técnicas

En el ámbito de la toma de decisiones de índole complejo las técnicas AHP y PROMETHEE han mostrado una gran eficacia aportando soluciones en diferentes campos donde se estaba en situaciones complejas que iban desde valoraciones abstractas hasta opiniones en conflicto, ofreciendo soluciones de actuación que en muchas ocasiones podrían verse como recomendaciones.

3.3.11.1 AHP [21]

Se trata de una técnica estructurada para organizar y analizar decisiones complejas, basándose en conceptos matemáticos y psicológicos. Fue desarrollada en los años 70 por Thomas L. Saaty. Tiene una aplicación particular en la toma de decisiones en grupo y se ha utilizado en gran variedad de situaciones, en campos relacionados con el gobierno, industria, salud, construcción de barcos y educación.

En lugar de ofrecer una decisión correcta, AHP ayuda a quien ha de tomar una decisión a encontrar la que mejor encaje con sus meta y su entendimiento del problema. Ofrece un marco entendible y racional de estructuración del problema de decisión que permite representar y cuantificar sus elementos hacia una meta además de ofrecer la evaluación de soluciones alternativas.

Los usuarios de AHP descomponen su problema de decisión en una jerarquía de subproblemas más entendibles, cada uno de los cuales puede ser analizado independientemente. Los elementos de la jerarquía pueden relacionarse con cualquier aspecto del problema de decisión, sea este tangible o intangible, cuidadosamente medido o estimado a groso modo, bien o poco entendido, etc.

Una vez la jerarquía se ha construido, los que han de tomar la decisión evalúan cada uno de los elementos que conforman dicha jerarquía de forma sistemática realizando comparaciones dos a dos, de cada una de las alternativas de un nivel con las restantes. Cuando se realizan las comparaciones, las evaluaciones pueden basarse en mediciones concretas presentes en los datos pero también pueden usarse opiniones personales sobre los elementos comparados.

AHP convierte estas evaluaciones a valores numéricos que finalmente pueden ser procesados y comparados sobre el rango entero del problema. Como resultado final se obtiene un peso de prioridades final para cada uno de los elementos de la jerarquía, ello permite que problemas diversos y en ocasiones de difícil medición puedan ser comparados unos con otros de forma

racional y consistente. Esta característica es la que distingue al método AHP de otras técnicas de toma de decisiones.

En el paso final del proceso, se calculan las prioridades para cada una de las decisiones alternativas. Estas prioridades permiten conocer tanto la alternativa preferida desde diferentes puntos de vista como la distancia al resto de alternativas lo que permite ofrecer una lista ordenada de alternativas.

La técnica puede utilizarse de forma individual, pero es más útil cuando es aplicada con equipos de gente trabajando en problemas complejos, especialmente aquellos que incluyen percepciones y juicios cuya resolución supone repercusiones a largo plazo. Tiene ventajas únicas cuando los elementos relevantes de la decisión son difíciles de cuantificar o comparar, o donde la comunicación entre los miembros de los equipos es difícil por motivos de especialización, terminología o perspectivas.

Las situaciones de decisión en las que el método AHP se ha aplicado se pueden agrupar en:

- **Elección:** Selección una alternativa frente a un conjunto.
- **Clasificación:** Ofrecer una lista ordenada, de mayor a menor preferencia.
- **Priorización:** Determinación del mérito relativo de los miembros de un conjunto de alternativas.
- **Relocalización de recursos:** Reparto de recursos entre un conjunto de alternativas.
- **Benchmarking:** Comparación de los procesos de una organización con los de aquellas organizaciones mejor posicionadas.
- **Gestión de la calidad:** Tratar con los aspectos multidimensionales de la calidad y de la mejora de la calidad.
- **Resolución de conflictos:** Solucionar conflictos entre partes con posiciones y objetivos aparentemente incompatibles.

Algunos usos de AHP han sido comentados en la literatura. Entre ellos se pueden destacar:

- Selección de un tipo de reactor nuclear.
- Decidir la major manera de reducir el impacto global del cambio climático.
- Cuantificar la calidad general de los sistemas de software.
- Seleccionar la Universidad.
- Decidir donde ubicar las plantas de fabricación externas.
- Evaluar los riesgos en operaciones de tuberías de petróleo a campo a través.
- Decididir la mejor gestión de las cuencas hidrográficas de U.S.

3.3.11.2 PROMETHEE y Gaia [2,3,4]

Desarrollado a principios de los años 80 y continuamente estudiado y refinado desde entonces. Este método de toma de decisiones ha sido utilizado alrededor de todo el mundo en una amplia variedad de escenarios de decisión, en campos tales como empresas, instituciones gubernamentales, transportes, salud y educación. Al igual que el método anteriormente comentado el objetivo de este método no consiste en encontrar la mejor solución sino en aquella que mejor se acopla con los objetivos y la comprensión del problema.

La aproximación descriptiva, denominada Gaia[3], permite a quienes han de tomar una decisión visualizar las características principales de un problema de decisión, lo que facilita identificar los conflictos o sinergias entre criterios, identificar grupos de acciones y resaltar actuaciones interesantes.

Promethee ha sido utilizado en muchos contextos de toma de decisiones alrededor del mundo. En el año 2010 se publicó una lista no exhaustiva de publicaciones científicas acerca de las extensiones, aplicaciones y discusiones relacionadas con el método Promethee[4].

Al igual que el método anterior puede ser utilizado de manera única por un único individuo o por un equipo con diferentes criterios y perspectivas y además también se ha aplicado en la toma de decisiones en problemas similares a AHP.

4 Construcción del Motor de Prognosis

4.1 Detección de anomalías

Para la implementación del sistema de detección de anomalías se ha hecho uso de la librería Keras funcionando sobre TensorFlow.

Keras es una librería de alto nivel que puede funcionar sobre varias librerías de DeepLearning, de nivel más bajo, como pueden ser TensorFlow, anteriormente mencionada y Theano. Esto la hace idónea pues la independiza en cierta manera de la librería de DeepLearning que se esté utilizando por debajo.

La ventaja de utilizar TensorFlow frente a otro tipo de librerías de Machine Learning como Skitlearn radica en el hecho de que los cálculos de la red se pueden realizar, de manera transparente, sobre GPUs sin tener que cambiar o modificar el código, además TensorFlow ha portado Skitlearn y infraestructuras de cómputo distribuido como puede ser SPARK ha portado también TensorFlow con lo que se está buscando, además del pararelismo con GPUs, el paralelismo en infraestructuras distribuidas.

A continuación se muestra un ejemplo de definición de autoencoder en Keras:

```
from keras import initializers
from keras.layers import Input, Dense

def createStructureModelAE (isize, lnh, nenc):

     # Especificación neuronas en cada capa oculta
     for eh in lnh:
         if eh != 0:
            lnhAux.append(eh)

     lnh = lnhAux

     # Entrada
     input_d = Input(shape=(isize,))
     encoded = None

     for ih in range(len(lnh)):
         kernel_initializer=initializers.random_normal(mean=mean, stddev=std, seed=ih)
         bias_initializer=initializers.Ones()

         activation='tanh'
         if (ih == 0):
```

```
                encoded = Dense(lnh[ih],
                                activation=activation,
                                kernel_initializer=kernel_initializer,
                                bias_initializer=bias_initializer)(input_d)
            else:
                encoded = Dense(lnh[ih], activation=activation,
                                kernel_initializer=kernel_initializer,
                                bias_initializer=bias_initializer)(encoded)

        kernel_initializer=initializers.random_normal(mean=mean, stddev=std, seed=40)
        bias_initializer=initializers.Ones()

        activation='relu'

        encoded = Dense(nenc,
                        activation=activation,
                        kernel_initializer=kernel_initializer,
                        bias_initializer=bias_initializer)(encoded)

        for ih in range(len(lnh),0,-1):
            kernel_initializer=initializers.random_normal(mean=mean, stddev=std, seed=ih+1000)
            bias_initializer=initializers.Ones()

            activation='tanh'
            #activation='softmax'
            #activation='relu'

            decoded = Dense(lnh[ih-1],
                            activation=activation,
                            kernel_initializer=kernel_initializer,
                            bias_initializer=bias_initializer)(encoded)

        kernel_initializer=initializers.random_normal(mean=mean, stddev=std, seed=4000)
        bias_initializer=initializers.Ones()

        activation='relu'

        decoded = Dense(isize,
                        activation=activation,
                        kernel_initializer=kernel_initializer,
                        bias_initializer=bias_initializer)(decoded)


        # Modelo que relaciona una entrada con su reconstrucción exacta.
        # ---------------------------------------------------------------
        autoencoder = Model(input_d, decoded)

        # Creación de un encoder que relaciona la entrada con la representación codificada
        # ---------------------------------------------------------------------------------
        encoder = Model(input_d, encoded)

        # create the decoder model
        decoder = Model(input_d, decoded)

        # Compilación del autoencoder
        # ---------------------------
        autoencoder.compile(optimizer='adam', loss='mse')

        return encoder, autoencoder
```

A continuación se muestra un ejemplo de entrenamiento:

```
def training (autoencoder, x_train, x_test, nepocas, bs, sf=True):

        # Especificación neuronas en cada capa oculta
```

```
        autoencdoer.fit(x_train,
                        x_train,
                        epochs=nepocas,
                        batch_size=bs,
                        suffle=sf,
                        validation=(x_test, x_test))

        score=autoencer.evaluate(x_test, x_test)

        return autoencoder, score
```

A continuación se muestra un ejemplo del modelo autoencoder en predicción:

```
   def predict (autoencoder, xd):

        # Especificación neuronas en cada capa oculta
        pred=autoencdoer.predict(xd)

        return pred
```

4.1.1 Generalización mediante pseudocódigo

A continuación se muestra el funcionamiento de este tipo de modelos en pseudocódigo:

```
# Creación de la arquitectura del autoencoder:

  encoder, autoencoder = createStructureModelAE (tamaño entrada,
                        neuronas en cada capa oculta,
                        número de neuronas de la capa más interna)
# Entrenamiento:

      x_train, x_test = obtenDatos(ficheroEntrada)
      autoencoderEst, score = training(autoencoder,
                                       x_train,
                                       x_test,
                                       numero épocas,
                                       tamaño batch,
                                       shuffle=True)


# El pseudocódigo del modelo en producción sería:

        autoencoder = CargaModelo()
        while (Verdad):
              xd = obtén evidencias del momento
              pred = predict(autoencoder, xd)
              buscaAlerta(pred)
              duerme(x segundos)
```

4.1.2 Trabajo futuro

A continuación se muestra el conjunto de objetivos de mejora a partir de los resultados de investigación de modelos estadísticos para la detección de anomalías basado en Autoencoders.

- Estimación automática del número de componentes a utilizar en el cálculo del estadístico T2PCA.

- Evaluación del modelo de autoencoders con técnicas clásicas de detección de anomalías, sobre corpus idénticos.
- Optimización de hiperparámetros de nivel más interno relativos a las funciones de optimización y a las funciones de activación.
- Utilización de los modelos de autoencoders para imputación de datos faltantes.
- Lanzamiento de la estimación y optimización de autoencoders sobre Spark.
- Implementación de algoritmos bioinspirados para la optimización de hiperparámetros.
- Estimación iterativa del modelo de detección de anomalías para limpiar datos anómalos durante el proceso de aprendizaje.

4.2 Predicción de indicadores OEE

Para la implementación del sistema que permite predecir el estado de los indicadores OEE se ha utilizado la librería ***Scikit-learn***.

Scikit-learn es un framework para el lenguaje de programación Python que permite desarrollar algoritmos de machine learning en base a sus librerías. Entre todas estas librerías se incluyen algoritmos de Clasificación, Regresión y Clustering.

El sistema que calcula los indicadores OEE está basado en un algoritmo de regresión, más concretamente en el ***MLPRegressor***, que implementa una red neuronal. Este algoritmo funciona a través de la configuración de las distintas capas y las neuronas que tiene cada una de estas capas. Cada capa recibe una entrada de datos, que a s u vez puede ser la salida de otra capa.

A continuación se muestra el algoritmo que permite predecir un indicador en instantes futuros:

```
from sklearn.externals import joblib
from sklearn.neural_network import MLPRegressor from sklearn.preprocessing import
PolynomialFeatures

import numpy as np

# MACHINE LEARNING
## [POLINOMIAL]
modelPolynomial = PolynomialFeatures( degree = 2 ) data_pol = modelPolynomial.fit_transform(
data )

## [PLS]
modelPLSRegression = joblib.load( "indicador_PLSRegression.pkl" ) data_pls =
modelPLSRegression.transform( data_pol )

## [MODEL]
Model = joblib.load( "indicador_Model.pkl" ) prediction = Model.predict( data_pls )


for i in range( len( prediction ) ):
        if prediction[i][0] < 0.0:
                predict[i][0] = 0.0
```

5 Validación del Motor de Prognosis

5.1 Detección de anomalías

La **validación interna** del modelo de detección de anomalías mediante autoencoder se realizó utilizando un corpus de entrenamiento de prueba para buscar la hipótesis de partida sobre que los scores son normales y detectar cuando no lo sean, conformando así la anomalía. Dichos autoencoders nos ofrecen alertas a nivel multidimensional de los sensores presentes en las diferentes máquinas.

Puesto que este tipo de sistemas de detección de alertas ofrece la posibilidad de analizar alertas cuando se dispone de la información procedente de más de un sensor, situación que lo pone en ventaja frente a las técnicas clásicas de detección de alertas unidimensionales que miran la información presente a nivel unidimensional sin tener en cuenta la relación de cada sensor con el resto de sensores ofrecidos por el entorno.

Se ha de tener en cuenta que un mayor grado de multidimensionalidad supone una mejora en el cumplimiento de las hipótesis iniciales de los test estadísticos que finalmente se aplicarán en este tipo de sistemas.

En el problema que nos ocupa, y dado que no se dispone de multidimensionalidad en los datos de las diferentes cubetas, al considerar cada una de éstas por separado, se ha optado por simular dicha multidimensionalidad adicionando un estadístico del propio sensor. En este caso se ha optado por ofrecer como nuevo sensor el estadístico máximo del único sensor de medida de una cubeta del que se dispone de datos, que en nuestro caso ha sido la temperatura. Para ello se han agrupado las mediciones de dicho sensor según una frecuencia de tiempo previamente considerada y de cada agrupación se ha considerado la media y el máximo. Operando de esta manera estamos creando una situación de multidimensionalidad, que aunque baja, nos ofrece la posibilidad de tratar dicha información desde un punto multidimensionalidad, pudiendo ofrecer así el potencial de los estadísticos multidimensionales que se han planteado anteriormente en este documento.

Conceptualmente un autoencoder consiste en una red neuronal donde la entrada es idéntica a la salida. Al tratarse de una red neuronal se ha de fijar previamente la arquitectura interna a ser estimada. La estructura interna preestablecida influye en la precisión de los resultados finales al igual que otro tipo de parámetros prefijados durante el aprendizaje de la red. Esta elevada cantidad de factores influyentes en los resultados finales conlleva una fase previa de optimización de hiperparámetros para ayudarnos a averiguar aquellos más prometedores a la hora de ofrecer una mínima calidad final.

Para la optimización de los hiperparámetros se ha hecho uso de un modelo de optimización en dos etapas: una primera etapa de búsqueda rápida y global de parámetros y una segunda etapa, que a partir del mejor punto global encontrado realiza una búsqueda fina y local con el objetivo de buscar un punto próximo que ofrezca un mejor valor objetivo.

Para la primera de las búsquedas, la búsqueda global, se ha hecho uso de una generación cartesiana y discreta de puntos candidatos a ser evaluados. Puesto que dicho conjunto puede ser bastante grande se ha realizado la búsqueda de los N puntos dentro de este conjunto que cubran el mayor espacio multidimensional disponible y que además estén más distantes entre si, con objeto de cubrir dicho espacio de la mejor manera posible. Para ello se busca en dicho espacio multidimensional aquellos puntos que sean más ortogonales y estén más separados.

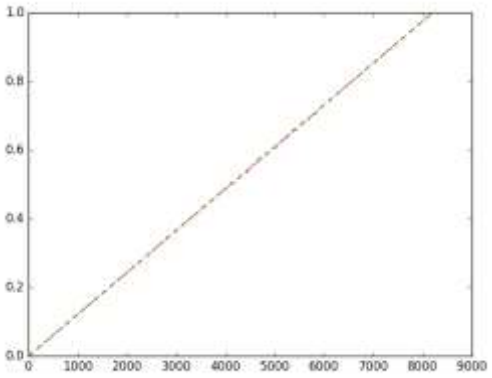
Una vez establecido el conjunto de N puntos con el que se va experimentar se realiza una evaluación rápida del punto. Tras esta evaluación se elige el mejor de los N puntos y se realiza una búsqueda local de dicho punto seleccionado aplicando una técnica de optimización de

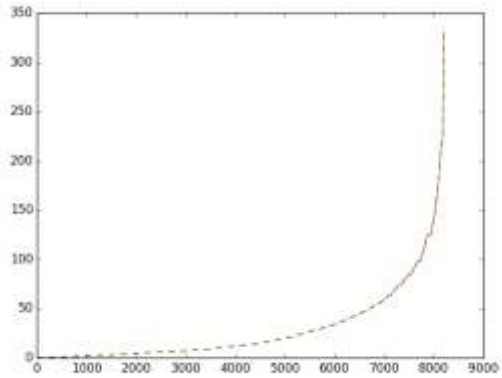
descenso por gradiente, el resultado final de este proceso consiste en la obtención de la configuración interna de la red junto a los parámetros de aprendizaje óptimos locales para un autoencoder y una configuración dada.

Una vez entrenado el autoencoder se puede poner en producción para la detección de alertas. Para ello se hará uso de una serie de estadísticos bien conocidos en la literatura, y que ya hemos apuntado en el apartado de arriba. Estos test estadísticos nos ofrecerán las alertas, cuando el estadístico calculado en un momento dado supere un límite superior previamente establecido a partir de un riesgo de primera especie, α, asumido por el usuario.

Estos test multidimensionales parten de la hipótesis de que las variables implicadas en su cálculo se distribuyen conforme a una distribución normal. Puesto que no existen garantías de que esto se cumpla en todos los casos se han creado una serie de modelos que permiten la transformación de las diferentes variables para poder garantizar así dicha hipótesis de partida en todo momento, independientemente de la distribución de probabilidad de los datos en cada momento.

Se han calculado varios modelos para garantizar la normalidad mencionada anteriormente, uno para cada uno de las diferentes variables externas implicadas en el estudio, además de variables internas de la red ofrecidas por las neuronas existente en la capa más oculta. Las siguientes gráficas muestran los resultados de dos de esos modelos. En dichas gráficas se puede observar la predicción en Rojo y los valores reales a predecir que permite garantizar. Se observa, una buena estimación pues las líneas roja y verde solapan.



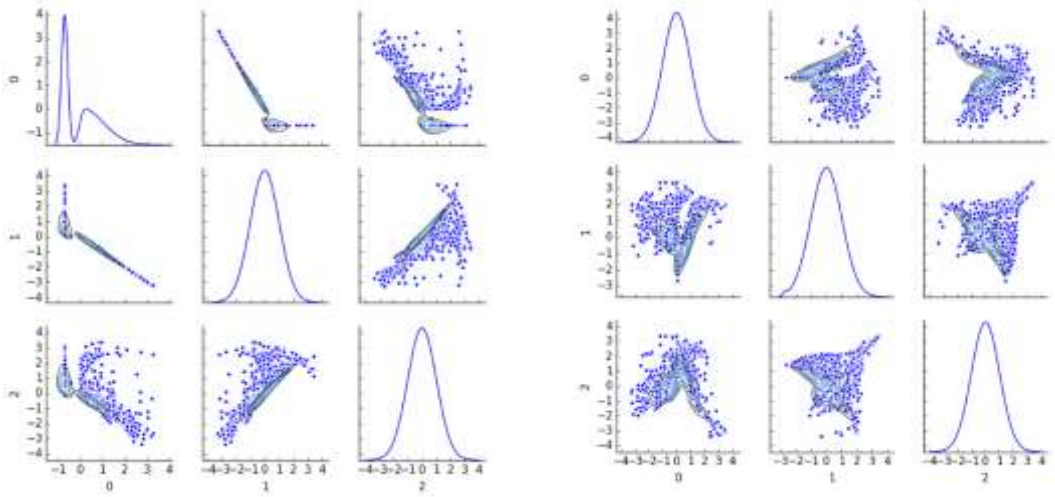


Uno de los modelos que se ha estimado contiene 3 neuronas en la capa central más oculta. Los valores que se observen en dichas neuronas nos van a ser de utilidad pues son valores que utilizaremos para realizar el cálculo del estadístico T2. Un valor elevado de este estadístico nos indica que el individuo evaluado representa a un individuo alejado del resto, y por lo tanto ante una alerta de este tipo deberemos de reaccionar, pues se tratará de un comportamiento alejado del comportamiento habitual.

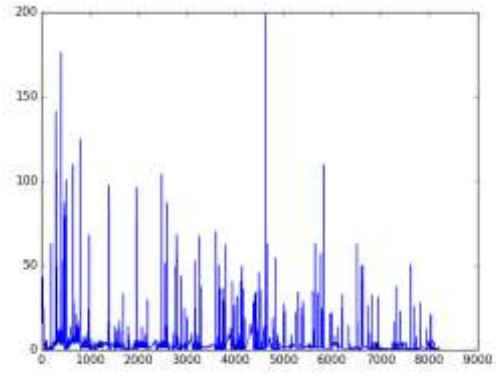
Las siguientes figuras muestran las distribuciones de los puntos extraídos a partir de los valores naturales de las neuronas centrales, figura de la izquierda, y los valores PCA de los valores de las neuronas, figura de la derecha. Cabe destacar, que mientras que en la figura de la izquierda se puede observar una distribución bimodal en la neurona 0 y una elevada correlación entre los valores obtenidos para las diferentes neuronas, en la figura de la derecha resultado de una transformación PCA de los valores de las neuronas, se garantiza mejor la normalidad de los datos además de la ortogonalidad o baja correlación entre los valores obtenidos para las diferentes neuronas. Esto conlleva además que para el cálculo del T2 en el primer caso

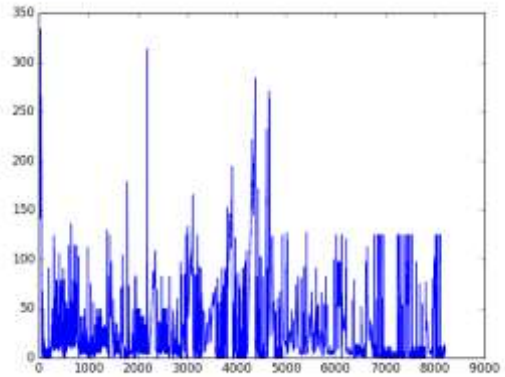
deberemos de invertir una matriz de varianzas covarianzas para su cálculo, pues no se puede asumir ortogonalidad entre los valores ofrecidos por las neuronas ocultas, mientras que para el caso de T2PCA tan sólo es necesario considerar la varianza de cada variable de forma independiente al resto de variables, esto nos facilita enormemente los cálculos de dichos estadísticos.

Además con T2PCA se garantiza mejor la normalidad de los datos pues se observan campanas de Gauss en todos los casos. El cumplimiento de la normalidad es fundamental pues se trata de una hipótesis previa que se ha de cumplir para poder estimar correctamente el estadístico T2 (T2PCA) y obtener conclusiones acordes con las hipótesis.



Las siguientes figuras se corresponden con los valores de los estadísticos T2, figura izquierda, y T2PCA, figura derecha.



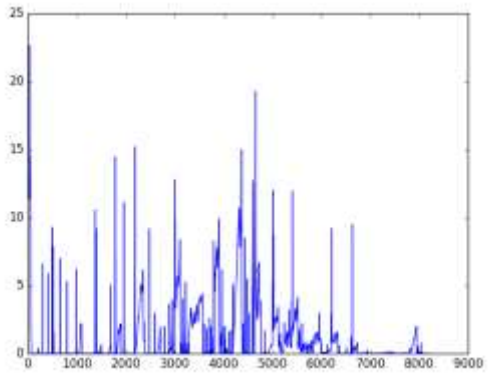


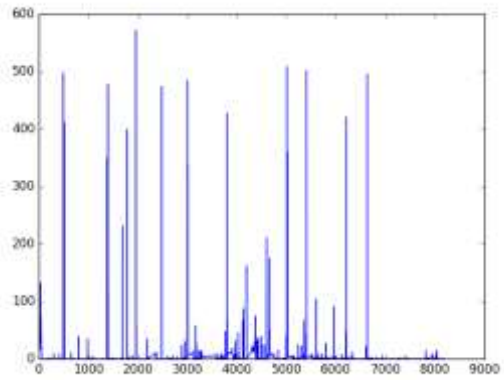
A continuación se muestran las gráficas del error de la predicción, calculado a partir de una distancia euclídea (SPE), figura de la izquierda, y el error de predicción calculado a partir de una distancia de Mahalanobis (SPE_MAH). Una de las diferencias fundamentales entre la distancia euclídea y la distancia de Mahalanobis, es que mientras que en la distancia euclídea se consideran las diferentes dimensiones como independientes, en la distancia de Mahalanobis se considera para su cálculo la posible existencia de correlación entre las diferentes dimensiones. Aprovechando esta situación utilizaremos el valor SPE como

estadístico indicador de que el individuo evaluado rompe con la estructura de correlación aprendida por el modelo a partir de la muestra de aprendizaje, cuando dicho individuo toma un valor elevado para dicho estadístico, pues en dicha situación se trataría de un individuo ajeno al resto de individuos sobre el que no es posible conocer a priori su estructura de correlación, y por lo tanto no tiene sentido aplicar la matriz de varianzas-covarianzas aprendida durante el aprendizaje.

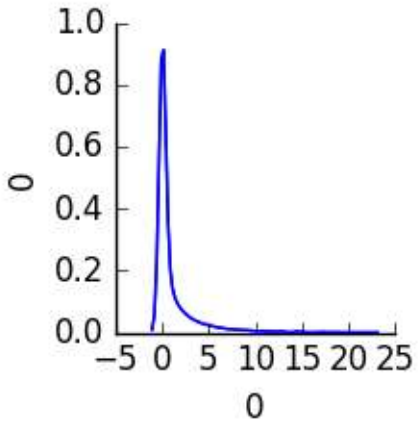
El estadístico SPE también nos va a servir para poder calcular el VIP_SPE que es un valor medido en porcentaje que ofrece información sobre la importancia que cada sensor tiene en una alerta de tipo SPE dada.

A partir de aquí y dado que en una alerta de tipo T2 o T2PCA, se supone que nos ofrece información inherente a la lejanía de un individuo respecto del conjunto pero no a una rotura de correlación, aprovecharemos el estadístico SPE_MAH para ofrecer el valor VIP_SPE_MAH que es un valor que ofrece información en forma porcentual sobre la importancia que cada sensor tiene en una alerta de este tipo. Pues en este último tipo de alertas, se supone que la correlación no se rompe y se puede asumir la ya aprendida previamente durante el proceso de aprendizaje del modelo.





El cálculo de los valores umbrales, a partir de los cuales se ofrecerá una alerta del tipo concreto, se calcula a partir del cálculo de un percentil obtenido a partir del riesgo de primera especie α, establecido previamente por el usuario, calculando el valor del percentil como: 1-α. Para ello se estimará un modelo que relacione los percentiles del estadístico con sus valores, a partir de la función de distribución del estadístico concreto, obtenida ésta directamente a partir de la muestra de aprendizaje del modelo. La siguiente figura nos muestra la función de distribución del estadístico T2 a partir de la cual se aprenderá dicha relación.



Ya en producción, una vez el modelo ha sido estimado, se utilizará para realizar la detección de alertas, para ello el autoencoder tomará las mediciones correspondientes en un instante dado y ofrecerá como salida un JSON con toda la información necesaria para ese momento dado. Esta información se pasará a la aplicación adecuada que la trate de la forma más pertinente posible: gráficas, mensajes, etc según sea el caso de uso.

De dicha información cabe destacar:

- **T2:** Es el valor del estadístico T2 de Hotelling, calculado éste a partir de los valores obtenidos en las neuronas más internas del autoencoder.
- **UCL_T2:** Es el valor umbral máximo en el que hemos de fijarnos para ofrecer una alerta de tipo T2. Dicho umbral se calcula a partir de un riesgo de primera especie **α** que es un parámetro de entrada previamente considerado por el usuario.
- **VIP_T2:** Relevancia porcentual de la información aportada por cada sensor en la alerta de tipo T2.
- **FT2:** Fiabilidad de la alerta. Dicha fiabilidad se calcula en base a la distancia del valor del estadístico observado respecto del límite superior **UCL_T2** y tomará valores entre 0 y 1. Donde 0, indicará baja fiabilidad y 1 máxima fiabilidad.
- **T2PCA:** Es el valor del estadístico T2 de Hotelling, calculado éste a partir de la transformación multidimensional PCA de los valores obtenidos en las neuronas más internas del autoencoder.
- **UCL_T2PCA:** Es el valor umbral máximo en el que hemos de fijarnos para ofrecer una alerta de tipo T2PCA. Dicho umbral se calcula a partir de un riesgo de primera especie **α** que es un parámetro de entrada previamente considerado por el usuario.
- **VIP_T2PCA:** Relevancia porcentual de la información aportada por cada sensor en la alerta de tipo T2PCA.
- **FT2PCA:** Fiabilidad de la alerta. Dicha fiabilidad se calcula en base a la distancia del valor del estadístico observado respecto del límite superior **UCL_T2PCA**.
- **SPE:** Es el valor del error del modelo calculado a partir de una distancia multidimensional. Se encarga de medir la rotura de correlación en los valores multidimensionales observados en un momento dado.

- **UCL_SPE:** Es el valor umbral máximo en el que hemos de fijarnos para ofrecer una alerta de tipo SPE. Dicho umbral se calcula a partir de un riesgo de primera especie **α** que es un parámetro de entrada previamente considerado por el usuario.
- **VIP_SPE:** Relevancia porcentual de la información aportada por cada sensor en la alerta de tipo SPE.
- **FSPE:** Fiabilidad de la alerta. Dicha fiabilidad se calcula en base a la distancia del valor del estadístico observado respecto del límite superior **UCL_SPE**.

A continuación se muestra un ejemplo de parte de la salida de una alerta.


```
{"T2": [28.4822740528948],
 "UCL T2": 30.500134825668475,
 "VIP_T2": [[71.07375209904264, 28.926247900957367]],
 "FT2": [0.4874519330480571],
 "T2PCA": [283.5800321217907],
 "UCL T2PCA": 214.65647873733315,
 "VIP T2PCA": [[69.99402052823658, 30.00597947176342]],
 "FT2PCA": [0.5834478978496411],
 "SPE": [14.772409997721255],
 "UCL_SPE": 10.483223180732223,
 "VIP_SPE": [[53.89563996313755, 46.10436003686245]],
 "FSPE": [0.5864471099823771]}
```

Para entrenar el modelo autoencoder, se ha de lanzar el comando:

```
# ------------------------------------------------------------------------------------------
# ARGUMENTS:
#    - t: Training csv file.
#    - o: Output pkl file where the model is stored.
#    - i: Number of maximum iterations during training.
#    - c: Number of components (inner neurons in the most deep layer in the AE).
#    - a: Encoder and Decoder architecture. In the example 20,10,c,10,20
#    - e: Number of epochs to train the model.
#    - p: Number of global points to consider in a previous test.
#    - g: This parameter appears if a gaussian distribution transformation of data is required.
# ------------------------------------------------------------------------------------------
python3 autoencoder.py -t <fichero csv> -o <fichero modelo> -i 100 -c 4 -a 20,20,20 -e 5000 -p 15
-g
```

Los parámetros –i, -c, -a, -e indican máximos, pero durante el proceso de aprendizaje el modelo optimizará el número idóneo de neuronas en la capa más oculta (-c) y el número de capas ocultas y el número de neuronas en cada capa (-a).

A continuación se muestra el pseudocódigo del modelo en producción:

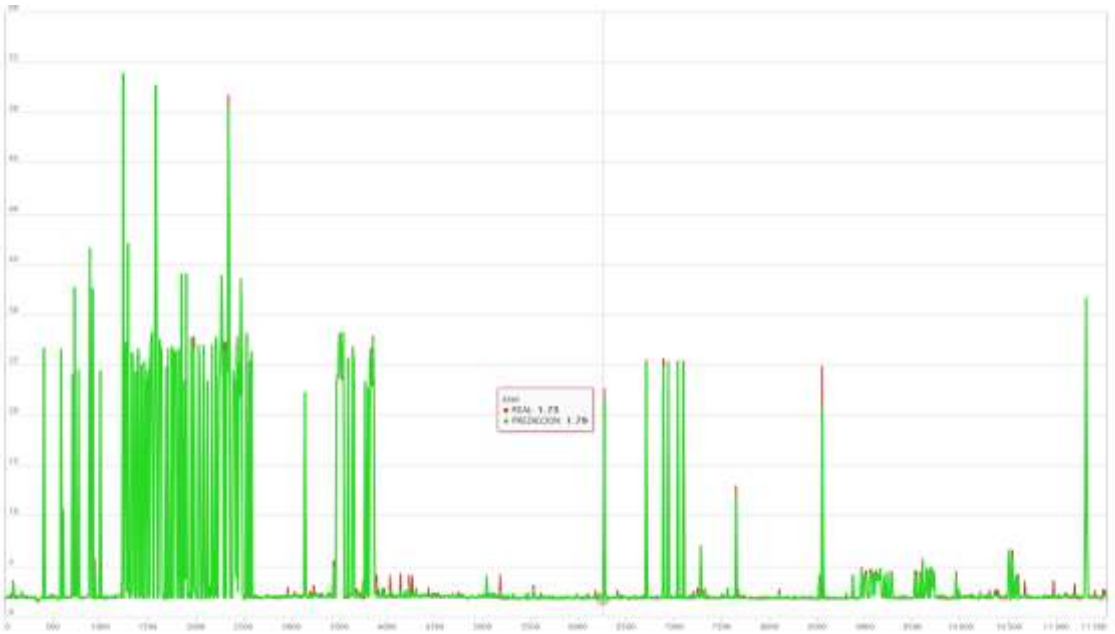
```
cDA=loadModel(modelo persistido)
   while (True):
          d = obtenDatosInstante()
          json=cDA2.predict(d, αT2, αT2PCA, αSPE)
          trataAlerta(json)
          duerme(instante temporal en segundos)

   cDA.destroy()
```

5.2 Predicción de indicadores OEE

La validación en el caso de los modelos de predicción basados en regresores logíticos se realizó de forma final, midiendo los indicadores de precisión, coeficiente de determinación y error cuadrático medio.

- ***Precisión (Accuracy):*** suele utilizarse en los problemas de clasificación para determinar la calidad de un modelo a partir de los resultados obtenidos en la matriz de confusión. En los casos de regresión, la precisión hace referencia al porcentaje de valores acertados respecto a los totales.
- ***Coeficiente de Determinación (R^2):*** determina la calidad del modelo indicando la variedad de resultados que puede explicar. Su valor se encuentra entre 0 y 1.
- ***Error cuadrático medio (RSME):*** indicador que mide la diferencia entre el estimador y lo que se está estimando. La diferencia se produce debido a la aleatoriedad o porque el estimador no tiene en cuenta la información que podría producir una estimación más precisa.



En la gráfica podemos observar que los valores resultantes de la fase de validación del modelo de regresión (color verde) suscriben casi a la perfección los valores reales, llegando a un accuracy del 94%, un R^2 de 0.74 y RSME de 13.45 . Eso se debe principalmente a un sobreentrenamiento del regresor debido a que algunas de las variables independientes guardan relación con la variable objetivo. El proceso de validación final se traslada al proyecto piloto, entrenando el modelo con los datos de la empresa piloto.

6 Bibliografía

6.1 Deep Learning

[1] DeepLearning; www.deeplearningbook.org

[2] Martín Abadi et al.; TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. (Preliminary White Paper, November 9 2015)

[3] Art; https://en.wikipedia.org/wiki/Adaptive_resonance_theory

[4] Autoencoder; https://en.wikipedia.org/wiki/Autoencoder

[5] Gan; https://en.wikipedia.org/wiki/Generative_adversarial_networks

[6] SOM; https://en.wikipedia.org/wiki/Self-organizing_map

[7] https://keras.io/

[8] https://www.tensorflow.org/

[9] https://docs.microsoft.com/en-us/cognitive-toolkit/

[10] http://www.deeplearning.net/software/theano/

6.2 Técnicas de predicción

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

[2] https://en.wikipedia.org/wiki/Long_short-term_memory

[3]https://upload.wikimedia.org/wikipedia/commons/5/53/Peephole_Long_Short-Term_Memory.svg

[4] https://upload.wikimedia.org/wikipedia/commons/5/5b/Activation_logistic.svg

[5] https://en.wikipedia.org/wiki/Activation_function

[6] https://upload.wikimedia.org/wikipedia/commons/9/9e/Activation_identity.svg

[7] https://upload.wikimedia.org/wikipedia/commons/c/cb/Activation_tanh.svg