Monitorización distribuida de un proceso para mejora de su calidad mediante técnicas de inteligencia computacional/deep learning: VAE

Trabajo de fin de grado
Ingeniería industrial electrónica y automática

Javier Merino Ortiz

Ingeniería de Sistemas y Automática
Bajo la supervisión de Mª Jesús de la Fuente Aparicio
Universidad de Valladolid
Septiembre 2024

Índice general

Resumen

En este trabajo se estudiarán diferentes técnicas de detección de fallos basadas en datos para el control de calidad en plantas industriales. Dada la gran cantidad de datos presentes en la industria moderna, estos métodos pueden ayudar a extraer conocimiento útil de los mismos, en vistas a mejorar la calidad de los procesos. Se han usado los datos del proceso Tennessee-Eastman, muy usado en la literatura de detección de fallos, para entrenar los modelos y probar su rendimiento.

Se empieza por el análisis de componentes principales, un método de extracción de características lineal, que puede crear nuevas variables que contengan la mayor cantidad de información posible del proceso, con un menor número de dimensiones. Analizar la distribución de los datos reducidos mediante estadísticos multivariables nos permite comparar los de funcionamiento normal con posibles datos anómalos, detectando así posibles fallos en estos.

También se han usado métodos de aprendizaje automático/aprendizaje profundo para la reducción dimensional de los datos. Los autoencoders permiten un aprendizaje no supervisado de los datos de la planta, aprendiendo correlaciones no lineales de las variables del proceso, pudiéndose detectar con ellos anomalías.

Los autoencoders variacionales suponen una mejora respecto de los autoencoders normales, proporcionando un mejor entrenamiento y una detección más precisa y fiable.

Por último, se ha desarrollado una metodología de detección de fallos distribuida, que divide las variables de la planta en bloques de forma automática, realiza la detección en cada uno por separado y condensa su información mediante la inferencia bayesiana.

Abstract

In this work a number of data-driven fault detection techniques are studied, to ensure quality control in industrial plants. Given the great amount of data present in modern industry, these methods can help extract knowledge out of them, in order to improve process quality. The Tennessee-Eastman process dataset, of wide use in fault detection literature, has been used to train the models and to test their performance.

It begins by principal component analysis, a linear characteristic extraction method which can create new variables that contain the process information, reducing its dimensionality while keeping the maximum amount of information. Studying the distribution of the reduced data using multivariable statistical methods, the fault-free data can be compared with possible faulty data, thus allowing fault detection.

Machine learning/deep learning methods have also been used for dimensionality reduction. Autoencoders allow unsupervised learning of the process data, learning non linear correlation of its variables, and being able to detect anomalies.

Variational autoencoders constitute an improvement upon regular autoencoders, allowing a better training and a more precise and reliable fault detection.

Lastly, a distributed fault detection methodology has been developed, dividing process variables automatically into blocks, performing fault detection in each one of them separately and condensing the information using bayesian inference.

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1. Introducción

Con la llegada de la sociedad industrial, en cuyas fábricas el operario empezó a convivir cada vez más con la maquinaria, la revisión manual de los productos pasó a ser imposible, tanto por el gran volumen de los mismos como por su creciente complejidad. El esfuerzo por garantizar que los productos fabricados cumplieran con sus requisitos de uso fué cobrando cada vez más importancia. Con el tiempo, el control de calidad llegó a constituirse como disciplina propia, al darnos cuenta de que los problemas producidos por la variabilidad en las diversas producciones podían ser abordados siguiendo ciertos principios generales. La medición empezó a ser una herramienta esencial, siendo necesario prevenir las consecuencias negativas antes de que sucediesen, lo que requería de una observación constante de los procesos productivos, no realizada ya -al nivel más bajo- por trabajadores, sino por sensores.

Hoy en día, todos estos problemas siguen presentes a mayor escala, y la forma de la actual industria está caracterizada por un aumento sin precedentes de nuestra capacidad para generar, almacenar y transmitir información. Todas las variables imaginables están registradas, lo que abre nuevos caminos para tratar de extraer información útil de todas esas pilas de datos a priori ininteligibles. Esta abundancia de información, unida a la gran capacidad de cómputo de que disponemos hoy en día, ha dado lugar a la aparición del fértil campo del aprendizaje automático, cuyos modelos cada vez son más potentes y aplicables a nuevos ámbitos de la vida profesional y también de la personal. Sus productos, desde generadores de texto capaces de mantener conversaciones con nosotros hasta reconocedores faciales que aportan seguridad a nuestra tecnología, están hoy en todas partes.

La aplicación de los métodos de aprendizaje automatico a los datos de la industria permite no sólo sustituir la visión por sensores, sino también automatizar tareas que antes requerían del pensamiento humano, tales como determinar si las

variables de un proceso están actuando como deberían o no. Además de permitir la automatización de esta tarea, la IA nos permite analizar enormes cantidades de datos de muchas dimensiones, pudiendo las máquinas ayudarnos a encontrar patrones que un observador humano jamás podría haber encontrado sólo.

Asegurar la calidad, así como la seguridad en la producción son dos objetivos de cualquier industria moderna, que se consiguen mediante la implementación de métodos de detección y diagnóstico de fallos que detecten cualquier anomalía que aparezca en el funcionamiento de la planta. Este problema ha sido abordado desde diferentes puntos de vista, como son los métodos analíticos o basados en modelos, métodos basados en datos como el aprendizaje automático y métodos basados en el conocimiento.

En este trabajo, se intenta usar estas nuevas tecnologías para aumentar el rendimiento de la industria, buscando aplicar algunas herramientas del aprendizaje automático al control de calidad de plantas industriales, y en concreto a la monitorización de dichos procesos.

Para realizar un estudio de monitorización basado en datos, se usará primero una de las técnicas más utilizadas, basada en el control estadístico de los procesos: el Análisis de Componentes Principales (PCA).

Sin embargo, como hemos comentado, el campo con más futuro en el tratamiento de datos industriales para la detección de fallos y por tanto para el control de calidad son las técnicas de inteligencia artificial, más en concreto, las redes neuronales, donde las redes más complejas, también llamadas de "deep-learning", o aprendizaje profundo, son capaces de extraer patrones complejos y procesar grandes cantidades de datos. Entre estas técnicas se encuentran los autoencoders variacionales, que emplearemos en este trabajo.

1.2. Objetivos

Con la elaboración de este trabajo se pretende desarrollar técnicas de detección y diagnostico de fallos basadas en datos en plantas industriales. Para alcanzar este objetivo se han usado técnicas basadas en el control estadístico de procesos, como el Análisis de Componentes Principales y técnicas basadas en deep-learning como los Autoencoders Variacionales. Además, como las plantas industriales son complejas, y tienen muchas variables se implementarán métodos distribuidos que se centran en diferentes partes de la planta, ya que no todas las variables causan el mismo tipo de fallos.

Con el análisis obtenido para los diferentes métodos, se pretende realizar comparaciones bajo las mismas circunstancias computacionales de los resultados de los distintos métodos. Todas las metodologías desarrolladas en el presente trabajo han sido testadas en los datos de la planta química Tennessee Eastman, bench-

mark utilizada como caso de estudio y banco de datos en la literatura científica, permitiendo el desarrollar y validar técnicas de monitorización de procesos en un entorno complejo y realista.

1.3. Organización de la memoria

A continuación, se especifica brevemente la organización que seguirá la presente memoria, dividida en seis capítulos:

El primer capítulo recoge la presentación e introducción del trabajo, en las cuales se especifica el contexto y las metodologías seguidas, los objetivos que se persiguen y se da una breve explicación del esquema que sigue el documento.

Se sigue en el capítulo 2 con una introducción teórica, en la que se explican los conceptos básicos usados a lo largo de todo el trabajo, tomados de los campos del control de calidad, estadística, inteligencia artifical, teoría de la información etc.

Tras esto, en el capítulo 3 se introducen brevemente la planta y el proceso Tennessee-Eastman (TE), hablando de su origen y de sus variables y fallos. También se presentan los conjuntos de datos con los que trabajaremos y contra los que probaremos la eficacia de nuestros modelos.

En el capítulo 4, los conceptos del capítulo 2 son aplicados a la detección de fallos en los datos de la planta TE, desarrollando una serie de programas de aprendizaje automático y detección de fallos con las siguientes metodologías: Análisis de componentes principales, Autoencoder, Autoencoder variacional, Autoencoder variacional distribuido.

Por último, en el capítulo 5 se explican las conclusiones más importantes a las que se ha llegado en el desarrollo del trabajo.

Capítulo 2

Conceptos teóricos

2.1. Control estadístico de procesos

El control estadístico de procesos (CEP) designa a una serie de técnicas que usan la estadística para monitorizar la variabilidad en procesos, para que estos puedan ser controlados[1]. Se ubica dentro del marco del control de calidad, que busca que los productos industriales presenten una calidad alta y regular.

Entendemos por proceso el sistema de máquinas, personas, materiales y operaciones que se combina para fabricar un producto. Su definición es muy amplia, pero podemos considerarlos en general como compuestos por entradas, operaciones internas y salidas. La calidad suele definirse como la conformidad a unas especificaciones (que pueden ser un código técnico, las necesidades del cliente u otros estándares). La variación, el hecho de que las características de un proceso no sean siempre las mismas, es una parte inherente de los procesos, que ha de ser controlada para generar productos de una calidad estable.

La calidad de los productos disminuye porque alguna parte de su producción ha variado. Si nos aseguramos que las variaciones -ya que no se pueden eliminar del todo- permanezcan dentro de ciertos límites, podremos garantizar de que la calidad de nuestros productos permanece en el rango deseado.

La necesidad de un control de calidad sistemático surgió en la época industrial, con la llegada de la producción en masa, a principios del siglo XIX. En la producción artesanal, el artesano controlaba personalmente cada aspecto de la producción y podía arreglar los productos que tuvieran fallos. Esto dejó de ser posible cuando los productos pasaron a ser tantos y tan complejos que ya no era posible monitorizar personalmente todos los fallos ni arreglarlos.

Dentro de este contexto surgió la disciplina conocida como control estadístico de procesos (CEP), destinada a monitorizar la variabilidad de los procesos de fabricación de manera sistemática. Su base teórica fué establecida por Walter

Shewart, trabajando en los laboratorios Bell, a partir de los años 20. Shewart creó las gráficas de control y estableció dos normas para el tratamiento de los datos de los procesos:

1. Los datos no tienen significado fuera de su contexto.

2. Los datos contienen señal y ruido. Para extraer la información, uno debe separar en los datos la señal del ruido.

Shewart expresó así las ideas fundamentales del control de calidad científico en la industria [2]:

> *The object of industry is to set up economic ways of satisfying human wants and in so doing to reduce everything possible to routines requiring a minimum amount of human effort. Through the use of the scientific method, extended to take account of modern statistical concepts, it has been found possible to set up limits within which the results of routine efforts must lie if they are to be economical. Deviations in the results of a routine process outside such limits indicate that the routine has broken down and will no longer be economical until the cause of trouble is removed.*

Durante la 2ª Guerra Mundial, el gobierno de los Estados Unidos aplicó el control de calidad a su producción armamentística, usando gráficos de control para fabricar productos de mayor calidad, con menores costes y de forma más eficiente, lo que ayudó en gran medida a su éxito militar. Sin embargo, en los años que siguieron a la guerra, los estadounidenses relajaron sus exigencias técnicas, prefiriendo centrarse en la cantidad de productos fabricados en lugar de la calidad de los mismos.

Mientras tanto, en Japón, William Edwards Deming, un seguidor de las ideas de Shewart, fué aplicando el control de calidad a la industria japonesa, estableciendo estándares de calidad y técnicas para mejorarla. El uso de las estadísticas para controlar la calidad de la producción ayudó a convertir a Japón en uno de los productores mundiales de mayor nivel. Un elemento fundamental fué cambiar el enfoque que tenían de inspeccionar la calidad de los productos fabricados, centrándose más en el proceso que los genera. Se demostró ser más útil prevenir la mala calidad que simplemente detectarla.

Cuando la competición creció con la aparición del mercado global, el control de calidad se convirtió en una herramienta necesaria para cualquier país que quisiera competir en él, culminando con la publicación en 1987 de las normas ISO 9000, un conjunto de normas internacionales de control de calidad de carácter general para la producción de bienes y servicios. [3]

Aunque todo proceso presenta cierta variabilidad, debida a causas ambientales (que se suponen aleatorias), hay otro tipo de variabilidad no controlada[4], que es asignable a factores concretos, y que hay que eliminar. Se considera que un proceso está bajo control estadístico cuando la variabilidad que presenta es del primer tipo y fuera de control cuando también presenta del segundo. Las causas no asignables dan lugar a variabilidad regular y predecible, mientras que las asignables dan lugar a variabilidad irregular e impredecible.

Para poder cuantificar esto, el CEP necesita de métodos estadísticos, ya que la variabilidad no puede ser predicha y no todos los productos pueden ser observados, por lo que habrá que conocer el comportamiento del proceso haciendo inferencias con muestras aleatorias. Tomando muestras del proceso bajo control, podemos estimar su media y desviación. Y comparando muestras del proceso en funcionamiento con la media y desviación del proceso durante la operación normal, podremos observar si está o no bajo control estadístico, según lo mucho que difieran.

La media μ y desviación estándar σ^2 del proceso bajo control pueden estimarse a partir de una pequeña muestra de datos del proceso:

$$\hat{\mu} = \frac{\sum_{t=1}^{n} q_t}{n} \tag{2.1}$$

$$\hat{\sigma} = \frac{\sum_{t=1}^{n} (q_t - \hat{\mu})}{n-1} \tag{2.2}$$

Esto es una parte fundamental del control de calidad, pues una vez que un proceso logra producir un producto con una variación pequeña, aumentar la calidad centrando el producto en el diseño deseado suele ser más sencillo que reducir su variación.[1]

Además de controlar la variabilidad, se suele fijar un valor deseado, u objetivo, que intentaremos que los productos alcancen. El control estadístico de procesos buscará dar una señal cuando la media de la variable del proceso se haya desviado lo suficiente del objetivo.

La herramienta principal del control estadístico de procesos es el gráfico de control (figura: 2.1), que representa la variación temporal de una variable en el tiempo junto con los límites de control, conocidos como UCL (límite superior de control) y LCL (límite inferior de control), de forma que podamos observar si la variable los supera. Los gráficos de control permiten decidir cuándo intervenir en un proceso para modificar una evolución no deseada.

Saber escoger los límites de control es importante. Unos límites muy estrechos detectarán las variaciones inherentes al proceso como fallos, generando falsas alarmas cuando el proceso esté operando de forma normal; por el contrario, unos límites demasiado amplios podrían no detectar desviaciones significativas del funcionamiento normal.

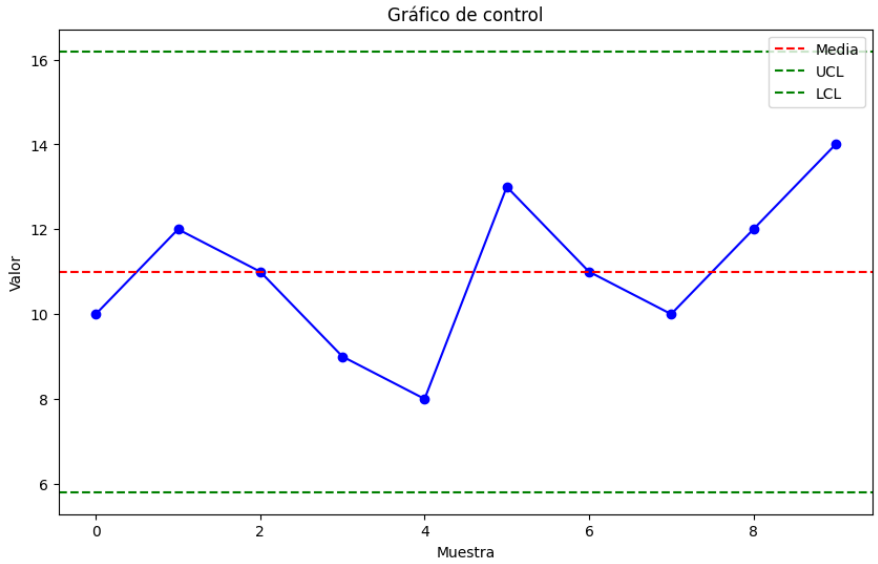


Figura 2.1: Gráfico de control

Para calcular unos buenos límites, tendremos que cuantificar matemáticamente la variación de la variable que estemos controlando. La desviación estándar σ, que mide lo alejado que un dato está de la media, sirve para elegir los límites de tolerancia, dentro de los cuales se considerará que los datos bajo control han de estar. Por ejemplo: para una tolerancia del 99 % (que asume que el 99 % de los datos estarán dentro de los límites), los límites de control superior e inferior vendrían dados por:

$$\mu \pm 1{,}96\frac{\sigma}{n} \tag{2.3}$$

En los casos en que el valor objetivo sea cero y sólo tengamos valores positivos, sólo tendremos un límite de control superior, y las gráficas de control nos serviarán para comprobar si lo supera en algún momento (fig: 2.2). Esto es lo que haremos más adelante para nuestra detección de fallos, ya que las variables con las que construiremos las gráficas de control serán estadísticos que midan cuánto se alejan las variables del proceso del funcionamiento normal.

Una vez establecidos los límites de control, bastará con tomar muestras de la variable que queremos controlar, considerando que está fuera de control si excede dichos límites. Como en la práctica es normal que incluso un proceso bajo control tenga (durante períodos muy cortos de tiempo) valores fuera de los límites de control (fig: 2.3), se considera que la variable está fuera de control cuando un número n de muestras está fuera de los límites (fig: 2.4), para evitar falsas alarmas.

La mayoría de sistemas modernos tiene muchas variables, por lo que moni-

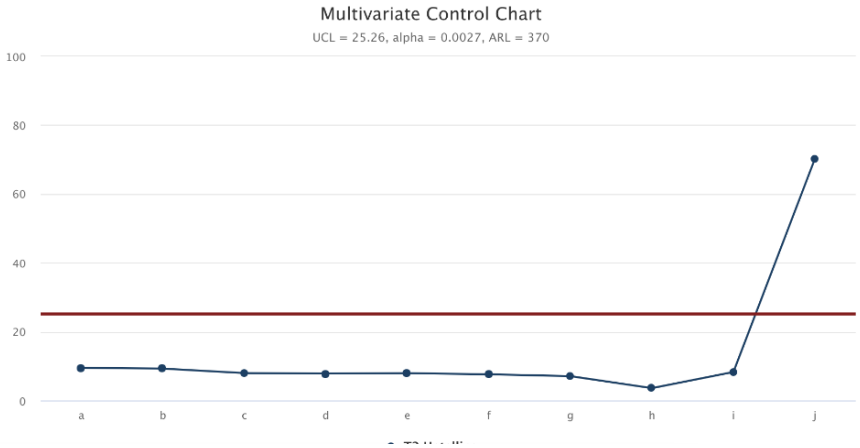


Figura 2.2: Gráfico de control multivariable con UCL [5]

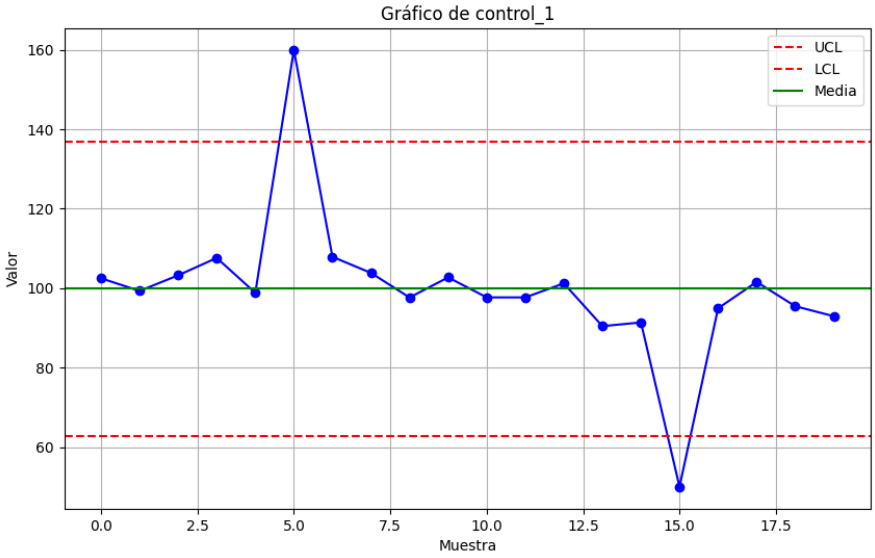


Figura 2.3: Proceso con falsas alarmas

torizarlas con gráficos de control como los que hemos visto hasta ahora sería demasiado trabajoso, además de que no quedaría claro cuándo el funcionamiento se debería considerar correcto y cuándo no. Si una de 80 variables excede su límite, ¿está el proceso fuera de control? ¿Y si son diez?

Por ello, debemos econtrar formas de extraer la información esencial de los datos que tenemos, tanto para poder interpretar más fácilmente la ingente cantidad de informacion que poseemos hoy en día como para poder determinar con cierta seguridad si un proceso está funcionando bajo control estadístico o no.

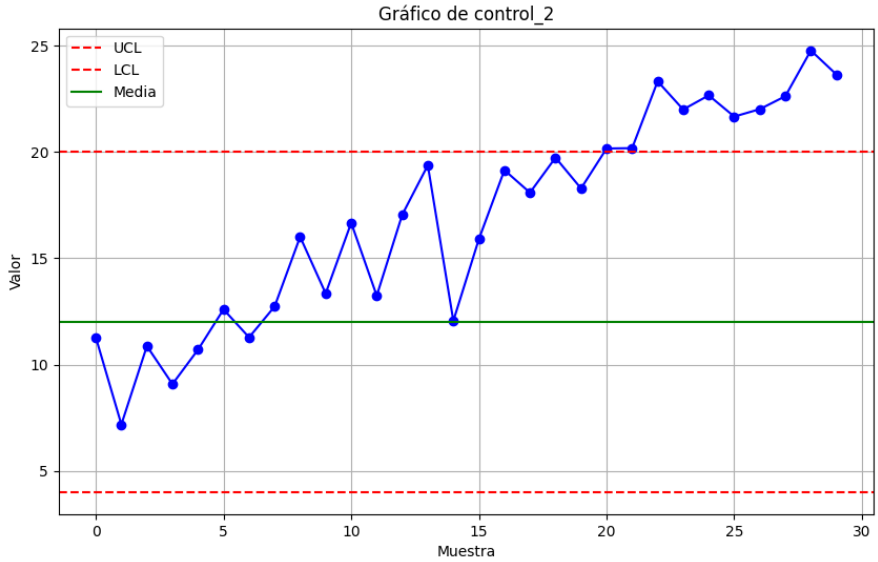


Figura 2.4: Proceso fuera de control

2.2. Detección de anomalías

Una anomalía es un valor o resultado que difiere respecto al tipo de valores esperado. Las anomalías sólo se pueden definir en contraste con los datos normales. En este contexto, una anomalía es simplemente un dato que no encaja con lo esperado; no implica que su aparición sea perjudicial (fig: 2.5) (si bien la detección de anomalías se puede usar para detectar fallos en procesos industriales, que es para lo que la usaremos en este trabajo).

La detección de anomalías se usa cuando no sabemos qué estamos buscando. Si tuvieramos ejemplos de datos anómalos, usaríamos métodos de clasificación. La detección de anomalías, por tanto, no hace suposiciones acerca de los datos anómalos, buscando en su lugar anomalías de todo tipo. Esto lo convierte en una técnica no supervisada [1].

Establecer la división entre datos normales y anómalos es uno de los problemas fundamentales de la detección de anomalías. ¿Cuán lejos de lo esperado debe estar un punto para que se considere anómalo? ¿Cuánta precisión nos podemos permitir? En general, tenemos que tener en cuenta tanto la forma de los datos a estudiar como el contexto, ya que algunas aparentes anomalías son explicables debido a factores normales recurrentes. Por ello debemos contar con suficientes datos, para no tomar por una anomalía algo que sucede de forma normal, como

[1] Hay otros criterios que incluyen métodos de clasificación dentro de la detección de anomalías, yo me baso en la definición expuesta en [7].

Figura 2.5: Oveja anómala [6]

puede ser un latido del corazón (fig: 2.6).

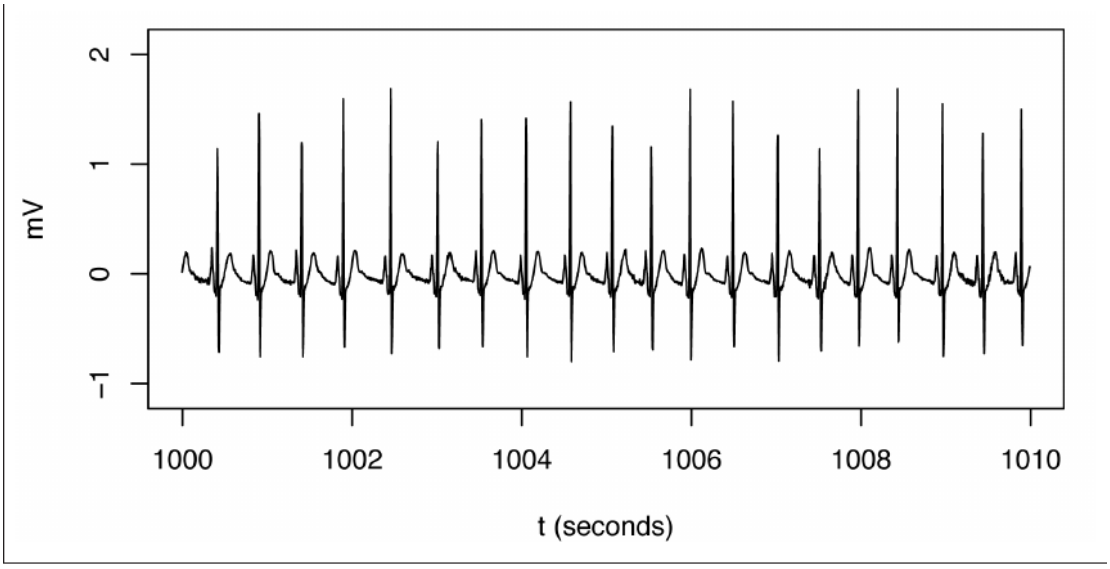


Figura 2.6: Electrocardiograma [7]

La detección de anomalías se compone de dos fases:

1. - Encontrar lo normal
2. - Detectar lo anormal

Conocer el comportamiento normal de un sistema complejo es una tarea difícil, que requiere entender bien su funcionamiento, por lo que los métodos de aprendizaje automático pueden resultar útiles, debido a su gran capacidad para extraer la información contenida en grandes cantidades de datos. Una forma habitual de definir el comportamiento normal es mediante modelos probabilísticos, según los cuales un evento anómalo sería aquel que fuese lo suficientemente improbable. El concepto de "suficientemente improbable"no tiene una definición objetiva, ya que en la detección de anomalías el criterio humano es un elemento fundamental, cuanto más sepamos del proceso, con mayor seguridad seremos capaces de distinguir su comportamiento normal del anómalo. En general, al buscar datos anómalos, estaremos buscando datos lo suficientemente alejados de la distribución que presentan los datos normales.

La variable que se usa para la detección de anomalías puede ser la medida de un sensor del sistema o (lo que usaremos nosotros) parámetros estadísticos que combinen la información de muchas variables.

2.3. Estadísticos multivariables

Como los datos que se manejan en las industrias reales son mediciones de gran cantidad de sensores, analizar las variables por separado no es útil, siendo necesario operar con datos de muchas dimensiones y capturar mediante parámetros estadísticos la información contenida en ellos, además de sus posibles correlaciones. En esta sección, tras introducir brevemente algunos conceptos preliminares de la estadística multivariable, explicaremos los dos parámetros que usaremos para determinar si una observación concreta es anómala o no: el estadístico T^2 de Hotelling y el estadístico del error residual o Q.

Los datos de procesos multivariables suelen organizarse en matrices como la siguiente:

$$X = \begin{pmatrix} X_1^1 & X_2^1 & ... & X_n^1 \\ X_1^2 & X_2^2 & ... & X_n^2 \\ ... & & & \\ X_1^m & X_2^m & ... & X_n^m \end{pmatrix} \tag{2.4}$$

donde las n columnas corresponden a las n variables y las m filas a las m mediciones, instancias o individuos cuyos datos se están monitorizando. Típicamente estas variables están correlacionadas, lo que significa que parte de la información que contienen no está solamente en sus valores sino también en sus relaciones (cómo varían conjuntamente).

El vector de media es el vector cuyos elementos son las medias de cada columna (variable) de X:

$$Media(x) = \vec{\mu} = [\mu_1, ..., \mu_m]^T \tag{2.5}$$

Denotamos a la varianza de la fila X_i como σ_i^2. Definimos la covarianza de dos variables X_i y X_j como:

$$\sigma_{ij} = Cov(X_i, X_j) = Media[(X_i - \mu_i)(X_j - \mu_j)] = Media[X_i, X_j] - \mu_i\mu_j \tag{2.6}$$

siendo $\sigma_{ij} = \sigma_{ji}$ y $\sigma_{ii} = \sigma_i^2$. Las covarianzas se suelen expresarse en la matriz simétrica de covarianza:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & ... & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & ... & \sigma_{2n} \\ ... & & & \\ \sigma_{m1} & \sigma_m^2 & ... & \sigma_n^2 \end{pmatrix} \tag{2.7}$$

$$\Sigma = Cov(X) = Media[(X - \mu)(X - \mu)^T] = Media[XX^T] - \mu\mu^T \tag{2.8}$$

Si dos variables tienen covarianza postiva, están directamente relacionadas (cuando una aumenta, la otra también) y, si tienen covarianza negativa, la relación es inversa (cuando una aumenta, la otra disminuye). Como este parámetro depende de las unidades en que cada variable esté medida, no es muy útil a la hora de interpretar la magnitud de la relación. Un parámetro mejor para comparar la relación de dos variables es la correlación, que está normalizada entre -1 y 1. Se define como:

$$Corr(X_i, X_j) = \rho_{ij} = \frac{\sigma_{ij}}{\sigma_i\sigma_j} \tag{2.9}$$

Dos variables cuya correlación es cero son linealmente independientes, aunque pueden darse relaciones no lineales entre ellas. La matriz de correlación coincide con la de covarianza para datos con media cero y desviación estándar de uno.

La matriz de correlación nos interesa principalmente para realizar su descomposicion en valores y vectores propios, lo que nos permite obtener información útil acerca de las direcciones de mayor variación de los datos.

$$\Sigma = V\lambda V^T \tag{2.10}$$

siendo V la matriz $n \times n$ cuyas columnas son los vectores propios de Σ y λ la matriz diagonal cuyos elementos son sus valores propios.

El t-cuadrado de Hotelling y el error cuadrático residual (o error Q) son dos estadísticos que cuantifican lo bien que un modelo describe una muestra dada. Los usaremos para decidir si una muestra de los datos del proceso se corresponde o no con el funcionamiento normal, midiendo la discrepancia entre el t-cuadrado y el Q de esta y los datos de funcionamiento sin fallos.

Se describen para datos generados por métodos basados en factores, que reducen la dimensionalidad de los datos proyectándolos a un espacio de menos dimensiones, tales como el análisis de componentes principales o los autoencoders.[8]

Dado un modelo que representa unos datos X de forma reducida, con cargas P y puntuaciones T, la relación entre los datos X y el modelo es:

$$X = TP^T + E = \hat{X} + E \tag{2.11}$$

donde $\hat{X}$ representa el modelo reducido de los datos, mientras que E contiene la información que hemos dejado fuera.

2.3.1. T^2 de Hotelling

El estadístico T^2 cuantifica la distancia de una observación respecto al centro del modelo. Es equivalente a la distancia de Mahalanobis[9], que mide la distancia entre un punto y una distribución. Supone calcular la distancia euclídea al cuadrado entre un punto y la media y escalarla por la inversa de la matriz de covarianza. El uso de la matriz de covarianza lo hace mejor que la distancia euclidea para datos multivariables, ya que añade la medida de cómo las variables varían juntas (fig: 2.7).

Como tanto las muestras como los datos del modelo están en el espacio reducido, T^2 mide las variaciones de los componentes principales (o, en el caso de redes neuronales, las variables del espacio latente). Q, en cambio, opera en el espacio de los datos, como ya hemos visto. Así, T^2 está más cercano a la métrica de los datos observados; valores fuera del rango ordinario generan valores altos de T^2.

El valor de T^2 para cada observación x_i se define como:

$$T_i^2 = xP\Lambda_k^{-1}P^T x^T \tag{2.12}$$

donde x es una de las m observaciones de X, P es la matriz de vectores propios reducida y Λ_k es la matriz diagonal de valores propios reducida, ambas de dimensiones $k \times k$, siendo k el número de dimensiones en el espacio reducido.

2.3.2. Error Q residual

El error cuadrático residual, o Q residual, es la suma de los cuadrados de cada fila de E:

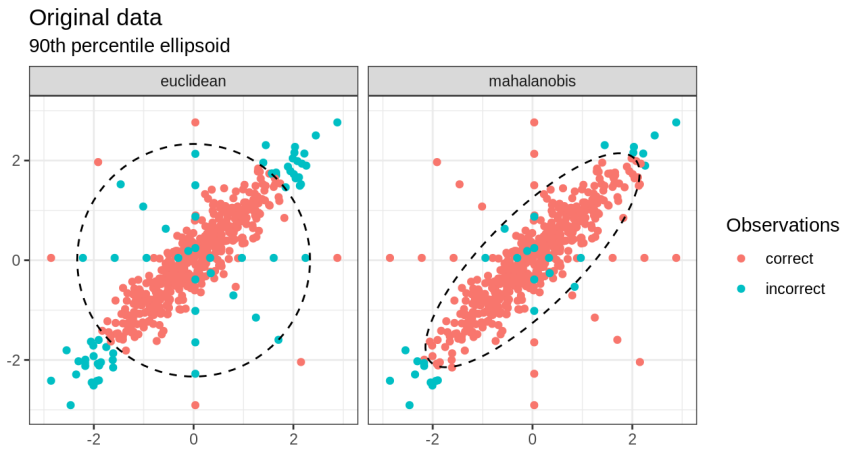


Figura 2.7: Distancia de Mahalanobis frente a la distancia euclídea [10]

$$Q_i = e_i e_i^T \tag{2.13}$$

siendo

$$e_i = (I - PP^T)x \tag{2.14}$$

donde e_i es la fila i-ésima de la matriz de residuo y P_k es la matriz de los k vectores de carga (cada vector es una columna de k). I es la matriz identidad $n \times n$. Q, por tanto mide la diferencia entre una muestra y su proyección al espacio reducido (si fueran iguales la matriz E sería cero).

El error de predicción cuadrado Q se genera porque, al realizar la reducción dimensional, hemos descartado parte de la información. La detección de una anomalía de tipo Q en un sistema de control, indica que la correlación entre las variables ha cambiado significativamente, es decir, las variables ya no varían las unas con las otras de la misma manera que hacían en los datos de funcionamiento normal.[11]

Controlando que estos dos parámetros no se salgan de sus límites, estaremos exigiendo a los datos de prueba que se mantengan en ciertos rangos (T^2) y que las variables varién entre sí de manera consistente (Q), todo ello referido a los datos con que se han entrenado los modelos. Tenemos entonces dos variables para crear gráficos de control con los que monitorizar el estado de todo el proceso.

2.4. Aprendizaje automático

Para resolver problemas con una computadora, necesitamos que ejecute un algoritmo. Para ciertos problemas, no conocemos los algoritmos, aunque sabemos cuáles han de ser las entradas y las salidas porque tenemos datos históricos, p.ej: no tenemos una aplicación que nos diga si un correo electrónico nuevo es spam o no, pero tenemos millones de correos guardados, y sabemos cuáles de ellos son spam y cuáles no. Los métodos de aprendizaje automático tratan de obtener aproximaciones útiles para este tipo de problemas (fig: 2.8). Asumiendo que podemos extraer información y generalizar a partir de los datos que ya tenemos, podremos hacer predicciones sobre nuevos datos.

El aprendizaje automático es la programación de computadoras para optimizar criterios de rendimiento usando datos o experiencias pasadas.[12] El aprendizaje consiste en ejecutar el programa para optimizar los parámetros del modelo en base a los datos que tenemos, haciendo que su funcionamiento se vaya ajustando más a lo que queremos que haga según aprende de los datos que le damos. Estos modelos pueden ser predictivos (para hacer predicciones) o descriptivos (obtener información). Dos criterios para evaluar modelos son la capacidad predictiva y la eficiencia computacional. Dado que se trata de hacer inferencias a partir de muestras, y no de obtener una solución única y perfecta, la base matemática de estos modelos es la estadística. Se puede considerar el aprendizaje automático como formado por modelos estadísticos (base teórica) y programación (implementación práctica).

Casi toda la ciencia consiste en crear modelos que se ajusten a los datos que tenemos, lo que se conoce como inducción, que extrae reglas generales de casos particulares. Para muchos problemas, los datos no pueden ser analizados por personas, y hay que automatizar el proceso de aprendizaje.

El uso de los modelos de aprendizaje automático ha crecido mucho en los últimos años, debido a que tenemos muchos más datos y capacidad de computación que nunca, aunque muchas de sus técnicas y algoritmos ya existían desde mediados del siglo XX[2].

Los modelos de aprendizaje automático trabajan con datos que ya conocemos, para extraer conocimiento de ellos. P.ej.: un modelo podría tomar los datos históricos de clientes de una empresa (edad, estado civil, salario etc.) y los datos de si esos clientes pagaron sus deudas o no, para entrenar a un modelo de aprendizje automático que tratase de encontrar una relación entre los datos y la salida que nos interesa. Una vez entrenado, este modelo debería ser capaz de predecir si un nuevo cliente pagará o no sus deudas, basándose en sus datos de edad, estado civil

[2]ver [14] para una breve introducción historica a la inteligencia artifical

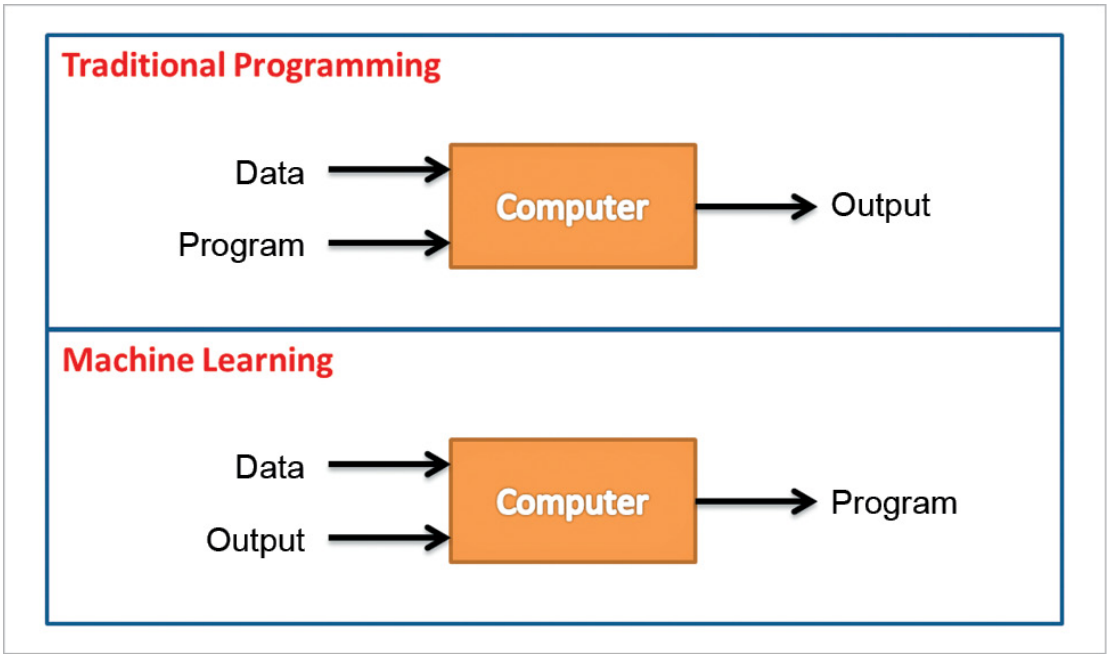


Figura 2.8: Dos paradigmas de programación [13]

etc. Los datos que queremos predecir se conocen como etiquetas, y no todos los modelos las requieren.

Los métodos de aprendizaje automático se pueden clasificar, según si sus entradas están etiquetadas o no, en:

- Aprendizaje superivsado
- Aprendizaje no supervisado

Los métodos que desarrollaremos en este trabajo serán métodos de reducción de dimensionalidad no supervisados[3].

En conjuntos de datos con muchas características, las distancias entre los datos son cada vez más complicadas de determinar, siendo difícil distinguir puntos típicos de atípicos. Esto hace también que los modelos se entrenen peor, ya que es más fácil que aprendan el ruido además de los patrones de los datos.

La reducción de dimensionalidad es un tipo de proceso de aprendizaje automático que busca reducir las dimensiones de un conjunto de datos, conservando la mayor cantidad de información posible. Suele usarse como etapa previa en muchas aplicaciones, en las que se reducen los datos de entrada antes de trabajar con ellos.

Algunas ventajas de realizar este proceso son:

[3] o semi-supervisados, según algunos criterios, ya que usaremos las entradas como objetivos.

1. - Usar entradas de menor dimensionalidad disminuye el uso de memoria y el tiempo de computación de los algoritmos.

2. - Los modelos más simples tienden a ser más robustos, teniendo menor varianza.

3. - Quedarnos con las variables que contienen mayor información nos permite entender mejor los datos e identificar más claramente las causas principales de las variaciones en los productos.

Los métodos de reducción dimensional se dividen en dos grupos:

- Selección de características, que buscan seleccionar las k dimensiones de las d que tenemos que contengan la mayor cantidad de información, descartando las $d - k$ demás

- Extracción de características, que buscan encontrar un nuevo conjunto de k dimensiones, que sean combinaciones de las d iniciales, siendo $k < d$, y que contengan su información esencial.

A continuación, veremos los métodos de extracción de características que usaremos para reducir las dimensiones de los datos de la planta, preservando la información esencial de las variables y de sus correlaciones.

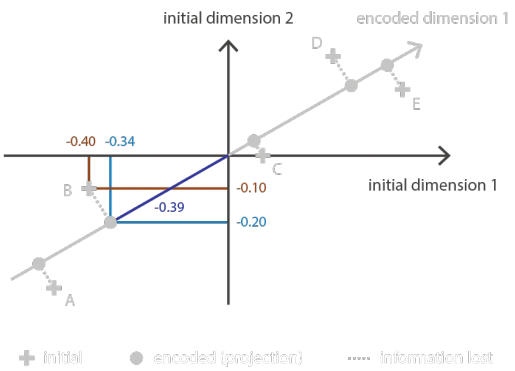
2.5. Análisis de componentes principales

El análisis de componentes principales, o PCA, es una técnica de reducción dimensional no supervisada que, tomando un conjunto de variables, o características, busca crear un conjunto más pequeño de nuevas características de forma que los datos proyectados en el nuevo espacio se parezcan todo lo posible a los del espacio inicial (fig: 2.9). Busca un subespacio lineal del espacio inicial descrito por una base ortogonal de nuevas características tal que el error de aproximar los datos por sus proyecciones sea mínimo.[15]

Como las variables interactúan unas con otras, es mejor usar métodos de extracción de características como este, en lugar de usar métodos de selección, ya que la información puede no estar solamente contenida en los valores de las variables concretas, sino también en cómo interactúan. PCA presupone que los datos están centrados en la media, lo que exige una etapa de preprocesamiento en la que se haga eso.

Este método realiza la reducción en dos etapas:

1. Analiza los datos para buscar qué partes son importantes



Point	Initial	Encoded	Decoded
A	(-0.50, -0.40)	-0.63	(-0.54, -0.33)
B	(-0.40, -0.10)	-0.39	(-0.34, -0.20)
C	(0.10, 0.00)	0.09	(0.07 0.04)
D	(0.30, 0.30)	0.41	(0.35, 0.21)
E	(0.50, 0.20)	0.53	(0.46, 0.27)

Figura 2.9: Proyección al espacio reducido en 2 dimensiones [15]

2. Los reduce, conservando los datos importantes.

La idea fundamental del PCA es que la varianza de una serie de datos mide la cantidad de información que contienen. Se calcula como sigue:

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n-1} \tag{2.15}$$

No es evidente que esto sea así, pero si pensamos en identificar personas conociendo estos datos (tabla: 2.1)

Persona	**Altura (m)**	**Peso (kg)**
David	1.60	70
Alejandro	1.90	68
Pablo	1.75	72

Tabla 2.1: Relación entre varianza e información

podemos ver que los datos de mayor varianza (las alturas) nos permitirían identificar mejor qué persona es quién, por lo que contienen mayor información.

Matemáticamente, el PCA es una transformación ortogonal lineal que traslada los datos a unas nuevas coordenadas llamadas componentes principales, cuyo primer eje (primer componente principal) se corresponde con la dirección de mayor varianza, el segundo con la segunda dirección de mayor varianza etc.

En la imagen (fig: 2.10), correspondiente a la etapa 1 del PCA, se han encontrado dos componentes principales, ordenados en el orden de mayor a menor varianza (en este ejemplo idealizado, toda la varianza está contenida en el primer componente). La segunda etapa consistiría en escoger aquellos componentes que

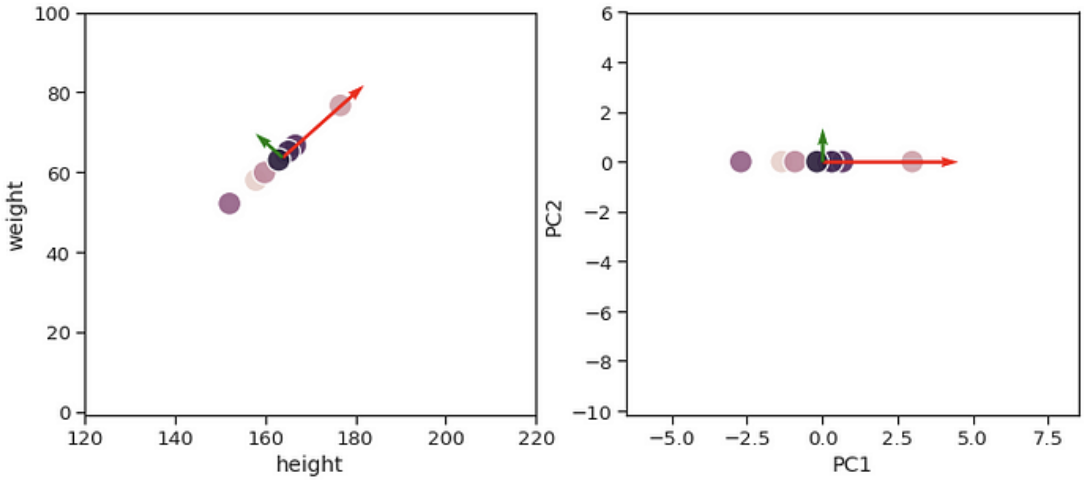


Figura 2.10: Dirección principal [15]

expliquen casi toda la varianza, en este caso eliminando el segundo componente principal que no nos da ninguna información.

En casos reales, tendremos N componentes principales y seleccionaremos aquellos que contengan la mayor varianza (fig: 2.11). Lo ideal es tener al menos 90 % de varianza con pocos componentes (con 2 o 3 podemos representar los datos gráficamente, aunque no todas las aplicaciones permiten esta reducción). Idealmente, los primeros componentes principales contienen la señal, y los últimos el ruido, por lo que, descartando los últimos componentes, que contienen poca información, separaríamos la señal del ruido.

Como estamos dejando componentes fuera, perdemos parte de la información contenida en los datos. Los datos con menor varianza son aquellos cuya distancia entre sí es más distorsionada al eliminar componentes en PCA, ya que la dirección de los ejes principales es la de mayor varianza.

Los pasos para implementar el PCA son los siguientes:

1. Preprocesar los datos.

   Restar la media a cada dato para que estén centrados en el 0 y dividir por su varianza, para que todos los datos tengan varianza 1.

2. Calcular la matriz de covarianza de los datos.

   Para ver cómo cada pareja de datos varía conjuntamente, calculamos la matriz de covarianza, que, para n características, tendrá dimensiones $n \times n$.

3. Realizar la descomposición en valores y vectores propios de la matriz de covarianza.

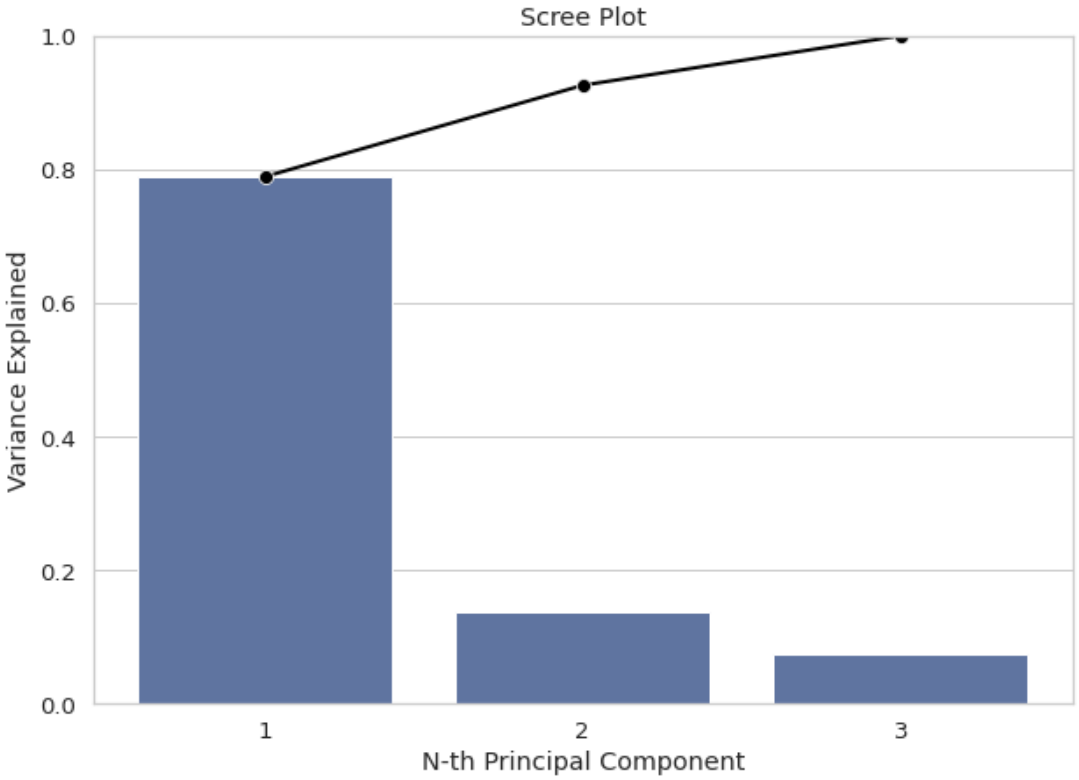


Figura 2.11: Varianza de cada componente

La descomposición de la matriz de covarianza en valores y vectores propios consiste en encontrar un conjunto de escalares (valores propios) y vectores (vectores propios) que cumplan la siguiente ecuación:

$$\Sigma\, v = \lambda\, v \tag{2.16}$$

donde v es el vector propio de longitud n y λ es el valor propio asociado al vector propio v.

Los vectores propios indican las direcciones de mayor varianza en los datos, mientras que los valores propios cuantifican la magnitud de la varianza que contiene cada vector propio.

Si una matriz puede ser descompuesta en valores y vectores propios, puede ser representada como:

$$X = V\, \Lambda\, V^{-1} \tag{2.17}$$

siendo V una matriz cuyas columnas son los vectores propios de X y Λ una matriz diagonal cuyos elementos son los valores propios de X.

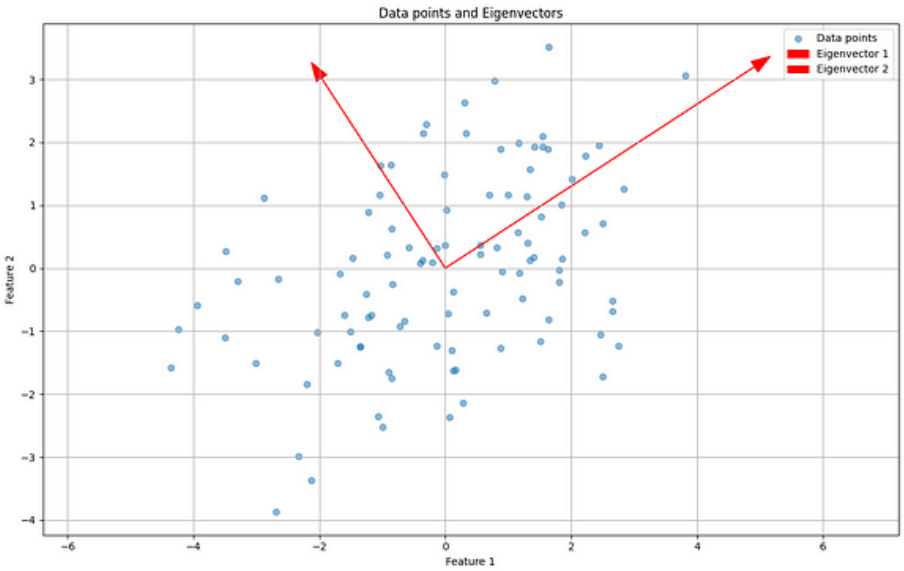


Figura 2.12: Direcciones de varianza [16]

En la imagen (fig: 2.12) vemos que los vectores propios apuntan en las direcciones de mayor varianza, y que los valores propios nos dan sus magnitudes. Por lo tanto, al calcular estos ya tenemos los componentes principales.

4. Ordenar los vectores propios en orden descendente según la magnitud de sus valores propios correspondientes.

   Ordenando los valores propios (cuya magnitud, como dijimos, nos dice cuánta varianza explica cada uno) en orden descendente, podemos seleccionar aquellos que contengan la mayor cantidad de información, descartando los demás. Se suelen escoger los k primeros componentes principales hasta llegar a un cierto porcentaje de varianza que queremos tener en el modelo simplificado. Ordenamos la matriz V de vectores propios según la magnitud de sus valores propios asociados, y hacemos cero las columnas correspondientes a los valores propios que hemos descartado, obteniendo la matriz de proyección V^T.

5. Determinar los k componentes principales a seleccionar.

6. Construir la matriz de proyección con los componentes principales.

Multiplicando la matriz de datos por la matriz de proyección, obtenemos la matriz de datos proyectados de dimensión $k \times k$ en el espacio reducido k-dimensional:

$$X_{proy} = X\,V^T \tag{2.18}$$

2.6. Redes neuronales

Las redes neuronales son un método de aprendizaje automático de tipo conexionista, inspirado por la neurociencia, cuyo elemento básico es la neurona artificial. Las primeras redes, creadas en los años 50, se aproximaban más a la biología, aunque fueron superadas por redes con arquitectura Von Neumann basadas en reglas.

Son de lejos el método de inteligencia artificial/aprendizaje automático más usado en la actualidad, dentro del paradigma del aprendizaje profundo, que se refiere a la aplicación de redes neuronales con más de una capa oculta para tareas de aprendizaje. Este paradigma se caracteriza por modelos con muchos parámetros, capaces de aprender reglas muy sutiles, tales como los de reconocimiento visual usados para coches autónomos o de generación de texto como el conocido ChatGPT.

Pese a su nombre, es más útil pensar en las redes neuronales como estructuras matemáticas que como algo cercano a la biología. La neurona artificial o perceptrón[4] es el bloque básico que forma las redes neuronales (fig: 2.13).

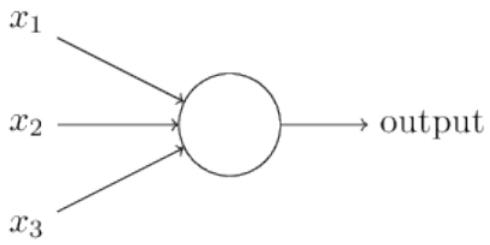


Figura 2.13: Perceptrón [17]

Un perceptrón se puede considerar como una operación matemática que recibe una serie de entradas (números que representan cierta información) y, en función de estas y de sus parámetros internos, produce una salida determinada. Los parámetros internos de un perceptrón son los pesos $\vec{w}$, que cuantifican el "peso"de

[4]Este nombre se usa a menudo por razones históricas, aunque en sentido estricto el perceptrón es un tipo de neurona que ya no se usa

cada entrada para influir sobre la salida y el sesgo b, que cuantifica lo fácil o difícil que es que la neurona se dispare.

$$salida = f(\vec{w} * \vec{x} - b) \tag{2.19}$$

La función $f()$ que relaciona las entradas y los parámetros internos de una neurona con su salida, llamada función de activación, depende del tipo de neurona, aunque lo más habitual es que sea uno de dos tipos: sigmoide o relu (fig: 2.14). La salida de una neurona sigmoide viene dada por la función:

$$\frac{1}{1 + exp(\vec{w}\,\vec{x} - b)} \tag{2.20}$$

La salida de una neurona RElU viene dada por la función:

$$max(0, \vec{x}\,\vec{w} - b) \tag{2.21}$$

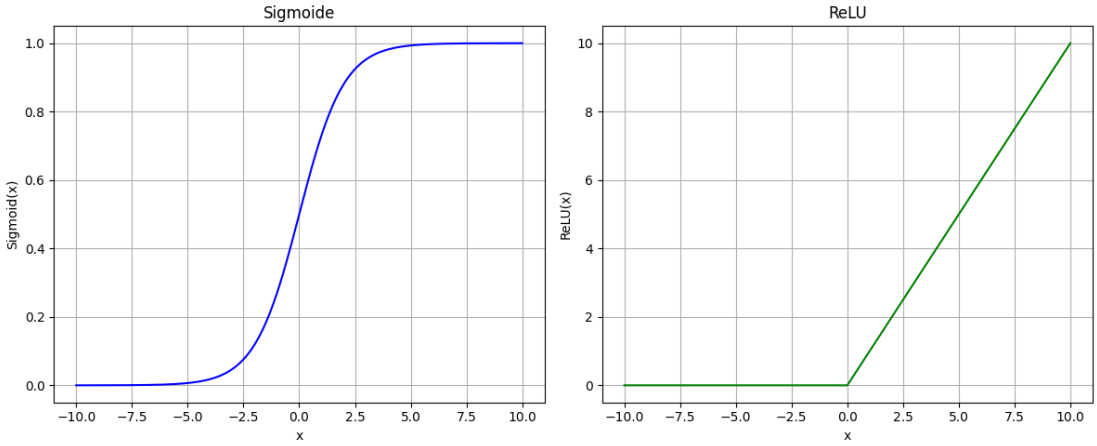


Figura 2.14: Funciones de activación sigmoide y ReLU

Usar este tipo de funciones de activación (en lugar de la función escalón, que caracterizaba la salida del perceptrón original) hace que un cambio pequeño en los pesos produzca un cambio pequeño en la salida, lo que permite entrenar la red de forma progresiva. Además, al usar funciones de activación no lineales, las redes neuronales pueden aprender relaciones no lineales entre los datos, pudiendo funcionar en muchos más contextos que otros métodos lineales. En nuestro caso, podrán aprender posibles correlaciones no lineales entre las variables de los procesos.

El siguiente elemento -a un nivel estructural más elevado- de una red neuronal es la capa, que funciona como una especie de filtro para los datos. Cada capa es un conjunto de neuronas colocadas en paralelo, que normalmente procesan las

mismas entradas. El número de entradas de la capa k ha de ser igual al número de salidas de la capa $k-1$. Se intenta que las salidas de las capas contengan representaciones útiles de la información que les damos. La siguiente red neuronal (fig: 2.15) produce una única salida a partir de cinco entradas, teniendo dos capas intermedias u ocultas:

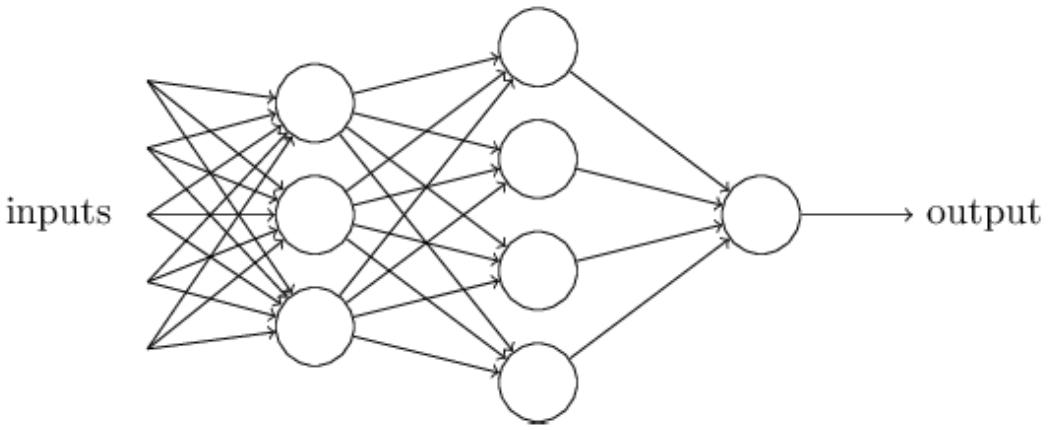


Figura 2.15: Red de 3 capas [17]

Una red neuronal es, entonces, una serie ordenada de capas, que recibe unas entradas y entrega unas salidas. Para que una red sea capaz de producir las salidas correctas para un tipo de entradas determinado hay que someterla a un proceso de entrenamiento, que consiste en ir introduciendo entradas a la red y modificando progresivamente los pesos de sus neuronas para que las salidas de la red coincidan todo lo posible con las salidas deseadas.

Este entrenamiento se puede realizar para casi cualquier tarea cuya información pueda ser codificada numéricamente, pudiendo realizar tareas antaño relegadas a la inteligencia humana, o incluso tareas que nosotros no podemos realizar. Algunas aplicaciones interesantes de las redes neuronales incluyen:

- Reconocimiento de rostros humanos. Se entrena la red con caras de la persona a identificar y de otras personas como ejemplos negativos.
- Detección de tinta para descifrar unos papiros calcinados por la erupción del Vesubio.[18] Se entrena la red con fragmentos anotados a mano.
- Generación de texto tipo ChatGPT, entrenado con un gran corpus de textos de todo tipo.
- Predicción de terremotos, entrenado con datos sismográficos.[19]

Para poder entrenar a una red, además de los datos de las entradas y las salidas, necesitamos tres elementos:

1. Un optimizador: Mecanismo mediante el cual el sistema modificará sus parámetros durante el entrenamiento.

2. Una función de pérdida: Función que el modelo buscará minimizar durante el entrenamiento, para saber en qué dirección llevar los parámetros.

3. Métricas: para comprobar si el funcionamiento de la red es correcto durante las fases de entrenamiento y de prueba. P.ej.: el porcentaje de datos correctamente clasificados.

El entrenamiento de una red, que se divide en etapas de entrenamiento llamadas épocas, consiste en introducir unas entradas en la red, calcular la función de pérdida, que mide la distancia entre las salidas reales de la red y las que queremos, y, en función de la función de pérdida, usar el optimizador para modificar los parámetros de la red. Si la función de pérdida nos da una pérdida pequeña, eso significa que la salida de nuestra red está cerca de la deseada, y el cambio que haremos a los parámetros será pequeño; en cambio, si estamos lejos del objetivo, haremos cambios más grandes. Para saber cómo cambiar los parámetros de forma que la función de pérdidas se reduzca y no aumente, el optimizador usa un método conocido como descenso de gradiente.

El gradiente de una función es una función que mide el ratio de cambio en cada punto de la función. Se puede ver como una generalización del concepto de derivada a funciones de más variables. Como el gradiente apunta a la dirección en que una función aumenta más, siguiendo la dirección opuesta tendremos la dirección en la que la función disminuye más. Dando un salto en esa dirección (cuyo tamaño depende, como ya hemos mencionado, del valor de la función de pérdidas), podemos ir disminuyendo esa función y, por tanto, haciendo que las salidas de nuestra red sean las que queremos.

Un problema que hay que tener en cuenta a la hora de entrenar redes neuronales es el sobreajuste (overfitting). Esto consiste en que, además de la información que queremos, los datos de cualquier proceso real contienen ruido, de naturaleza aleatoria y que no nos interesa que el modelo aprenda. Sin embargo, el proceso de aprendizaje que hemos descrito no distingue entre la información útil y el ruido, por lo que corremos peligro de que nuestra red aprenda "demasiado bien"los datos con los que la hemos entrenado, resultando en que funcione muy bien con esos datos y mucho peor con datos que nunca ha visto (que son precisamente con los que queremos que trabaje).

De los dos modelos de (fig: 2.16 y fig: 2.17), ¿cuál encaja mejor con los datos? ¿Cuál encajaría mejor con posibles datos nuevos? Cierto grado de sobreajuste es

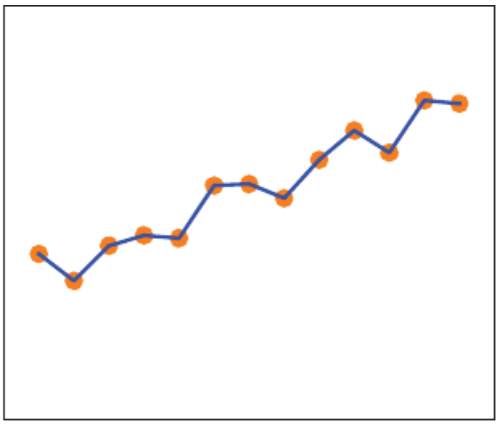


Figura 2.16: Modelo ajustado a los datos [14]

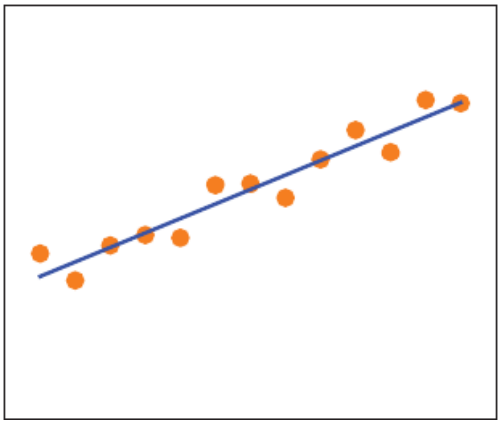


Figura 2.17: Regresión lineal sobre los mismos datos[14]

inevitable en estos métodos, pero es algo que ha de ser tenido en cuenta y reducido todo lo posible. Una forma de medir el posible sobreajuste es segregar un conjunto de los datos, llamados datos de validación, que no se usarán para entrenar al modelo sino que, una vez esté entrenado, se le introducirán para comprobar si la red trabaja bien con entradas que nunca ha visto antes. En la sección de autoencoders variacionales veremos una estructura de red neuronal más sofisticada, que permite mitigar el sobreajuste.

2.6.1. Autoencoder

El autocodificador o autoencoder (fig: 2.18) es una arquitectura de red neuronal, cuya salida se busca que sea igual a su entrada. Están compuestos de tres partes: un codificador (encoder), una capa latente y un decodificador (decoder). El autoencoder toma unos datos (a menudo imágenes), genera una representación comprimida, o latente, de los mismos y la decodifica. Como se les entrena para que las salidas se aproximen todo lo posible a las entradas, el espacio latente tendrá representaciones de menos dimensiones de las entradas, lo que permite usarlo para la reducción dimensional y la detección de anomalías.

Lo que le proporciona al autoencoder su función particular de compresión de datos es el hecho de que su función de pérdidas es el módulo de la diferencia entre su entrada x y su salida $\hat{x}$:

$$perdidas = ||x - \hat{x}|| \tag{2.22}$$

Como la capa latente tiene menos neuronas que la entrada, esta aprenderá sólo la estructura esencial de los datos, quedándose con la señal y no el ruido. Al

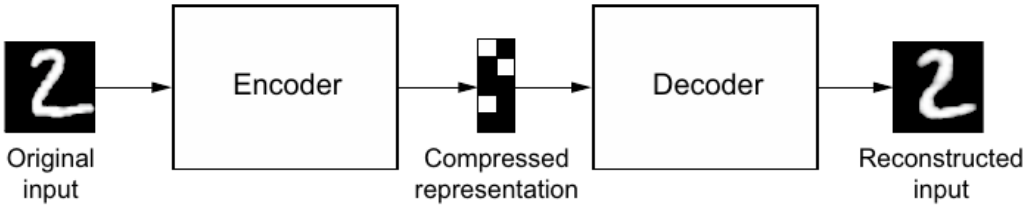


Figura 2.18: Autoencoder [20]

quedar forzada a priorizar qué aspectos de la entrada copiar, a menudo aprende propiedades útiles de los datos. Si las características de los datos fuesen independientes, la reconstrucción no sería posible; sin embargo, si en los datos existe cierta estructura (correlaciones entre las características), esta puede ser aprendida por la red.

La representación codificada de los datos generada por la capa latente se conoce como representación en el espacio latente, haciendo estas redes algo similar (aunque no linealmente) a lo que hace el PCA de pasar del espacio de datos al espacio de componentes principales. En el espacio latente se codifican las características de los datos que queremos aprender. La dimensión latente: el número de neuronas de la capa intermedia, será el número de características latentes que aprenda nuestra red, y deberá ser fijado antes del entrenamiento. P. ej.: un autoencoder entrenado para codificar rostros podría almacenar en su espacio latente características de: sonrisa, apertura de ojos, pelo etc.

Algunas características de los autoencoders son las siguientes:

- Especificidad: Sólo son capaces de comprimir datos similares a aquellos en los que han sido entrenados, a diferencia de otros métodos de compresión de información que tienen validez general para un tipo de archivo. P.ej.: si sólo ha sido entrenado con fotos de dígitos, un autoencoder no será capaz de comprimir fotos de árboles. Esta propiedad es fundamental para la detección de anomalías, como veremos abajo.

- Pérdidas: Inevitablemente, en la compresión de los datos, se pierde cierta información, por lo que sus salidas descomprimidas estarán degradadas respecto a las entradas originales.

- No supervisado: Se considera que un autoencoder es un método no supervisado, ya que los datos con los que trabaja no han de estar etiquetados. También se puede considerar como auto-supervisado, ya que la red genera sus propias etiquetas a partir de los datos de entrenamiento.

La arquitectura del autoencoder (fig: 2.19) sólo requiere que la dimensión de entrada sea igual a la de salida y que la dimensión latente sea menor que estas. Además de eso, lo más común es que el codificador y el decodificador tengan el mismo número de capas y de neuronas por capa, colocadas de manera inversa. El número de neuronas por capa tiende a disminuir según nos acercamos a la capa latente.

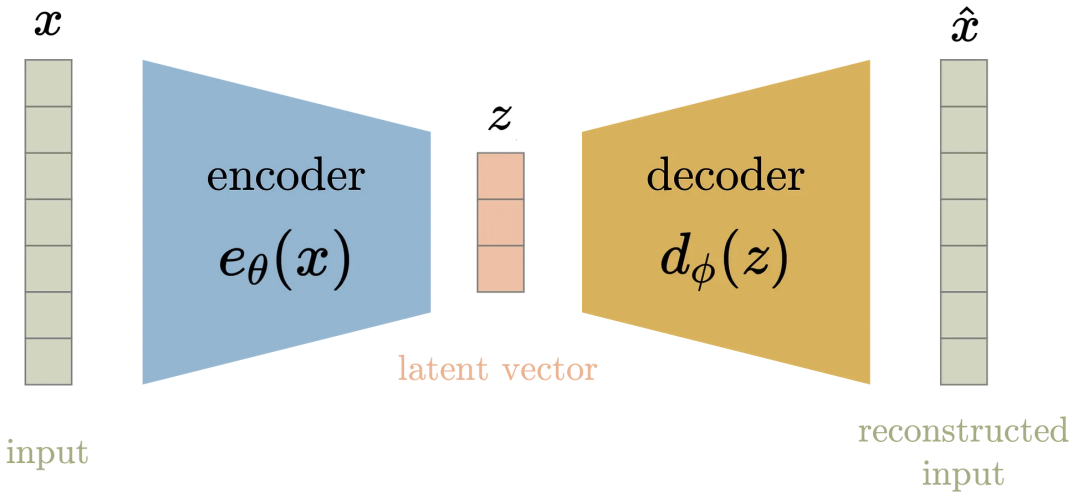


Figura 2.19: Estructura de un autoencoder [21]

Aunque podríamos hacer un autoencoder que copiase perfectamente las entradas, simplemente metiendo muchas capas y muchas neuronas por capa, esto no sería una buena idea en la práctica, pues lo que normalmente queremos es que la red sea capaz de aprender información útil de los datos en vez de ser una versión sofisticada de la función de identidad. Para ello debemos reducir el tamaño de las capas intermedias, obligando así a la red a cribar la información no esencial.

Los usos principales de estas redes están en la eliminación de ruido y la detección de anomalías. Para la eliminación de ruido no se alimenta a la red con las salidas que queremos que tenga, sino que la entrenamos con imágenes que contienen ruido como entradas y con las imágenes originales (sin ruido) como objetivos. Así aprenderá a eliminar el ruido, conservando la información útil (fig: 2.20).

En este trabajo usaremos los autoencoders para la detección de anomalías. Un autoencoder entrenado con datos de funcionamiento normal producirá salidas similares cuando se le alimente con estos datos, y (debido a su especificidad) salidas dispares cuando se le introduzcan datos de funcionamiento anómalo. Según esto, si entrenamos un autoencoder con los datos de funcionamiento normal de la planta, y le presentamos a la red entrenada datos de funcionamiento anómalo, su salida deberá presentar una distribución lo bastante diferente de la de los datos normales

como para detectarla con los estadísticos multivariables vistos en la sección 2.3. Veremos la implementación práctica de esto en el capítulo 4.

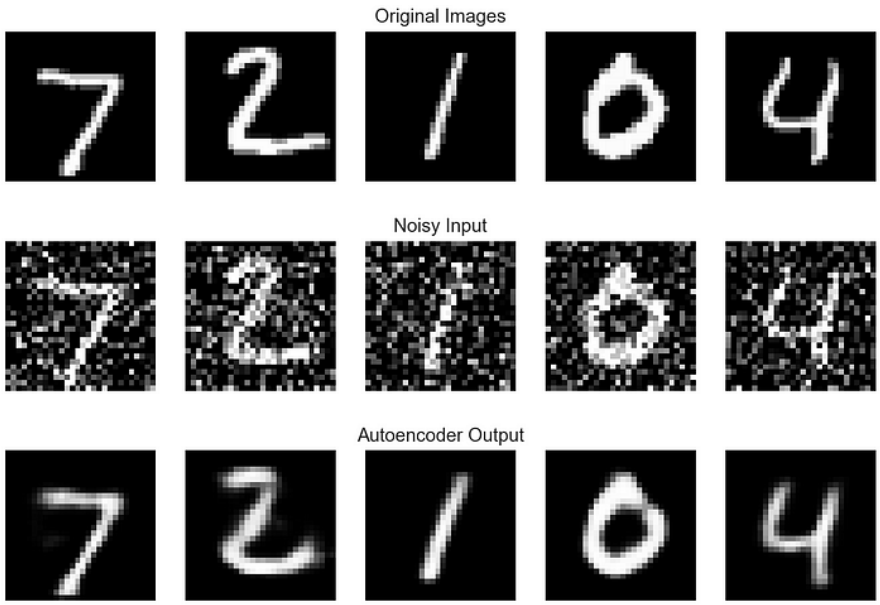


Figura 2.20: Autoencoder para eliminar ruido [22]

Por último, hay una arquitectura basada en el autoencoder que es más utilizada en la práctica: el autoencoder variacional o VAE, que veremos en la siguiente sección.

2.6.2. Autoencoder variacional

Los autoencoders variacionales o VAEs son redes neuronales basadas en los autoencoders tradicionales, a las que les añaden una regularización del espacio latente. Fueron definidos por Kingma et al.[23] y Rzende et al.[24]

El concepto fundamental de los autoencoders variacionales es el de espacio latente. Recordemos que un codificador transforma las entradas de un espacio de datos a un espacio latente, y que un decodificador toma puntos del espacio latente y los transforma en representaciones similares a las originales. Si el decodificador tomase puntos del espacio latente cercanos a las codificaciones originales pero distintos, podríamos pensar que generaría representaciones similares las originales, pero con ciertas variaciones. Esto no es cierto en el caso de los autoencoders tradicionales, ya que durante su entrenamiento no adquieren espacios latentes continuos ni altamente estructurados. Como consecuencia, sólo pueden reconstruir imágenes muy similares a los datos para los cuales han sido entrenados. Esto

puede dar lugar a un sobreentrenamiento. Los VAE modifican la estructura del autoencoder original para solucionar este problema.

En vez de transformar cada entrada en un punto del espacio latente, el VAE transforma cada entrada en una distribución estadística multivariable, caracterizada por la media y varianza de cada componente del espacio latente (fig: 2.21). Para generar la salida, el VAE muestrea un elemento de esta distribución y lo decodifica. Esto fuerza a que el espacio latente (que ahora está formado por distribuciones continuas y no por puntos) ofrezca salidas con sentido en todos sus puntos, quedando así continuo y estructurado.

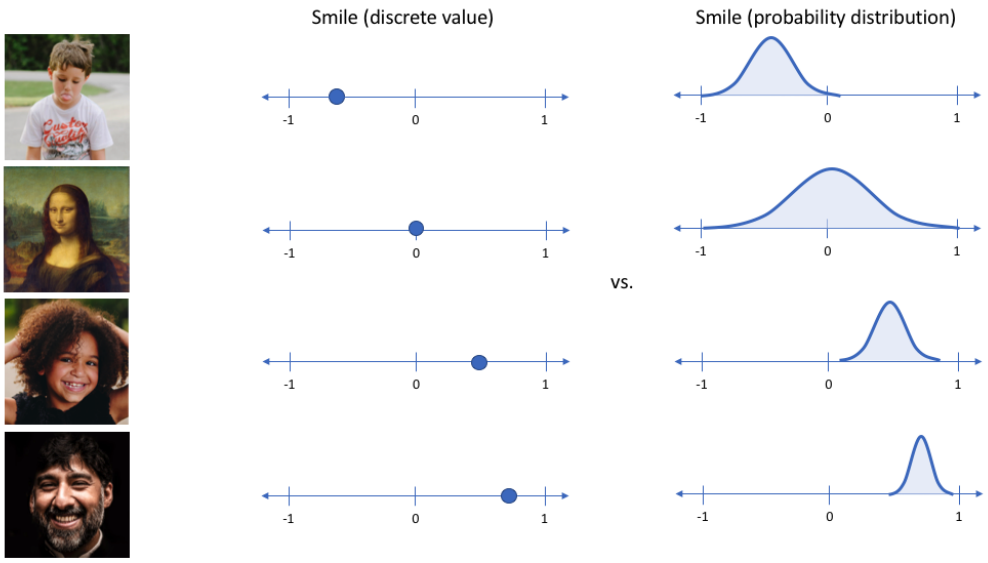


Figura 2.21: Representación de los atributos latentes en el VAE frente al autoencoder [25]

El proceso de regularización del espacio latente es el siguiente (fig: 2.22):

1. El codificador transforma un dato de entrada en los parámetros del espacio latente: z_media y z_var (con tantas distribuciones como parámetros tenga el espacio latente).

2. Se muestrea aleatoriamente un punto z de la distribución normal latente dada por las media y varianzas via

$$z = z\_media + exp(z\_var) * \epsilon \tag{2.23}$$

donde ϵ es un tensor aleatorio de valores pequeños.

3. El decodificador transforma dicho punto del espacio latente de vuelta al dato original.

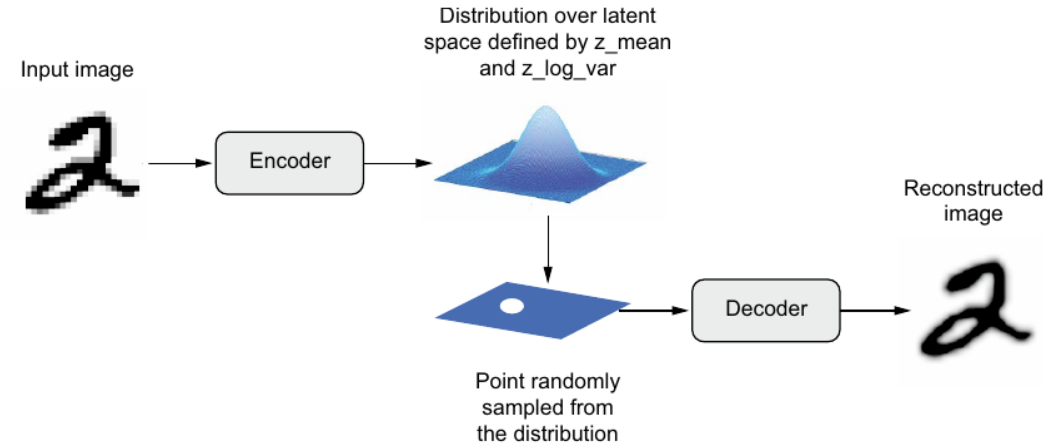


Figura 2.22: Muestreo [20]

Como ϵ es aleatorio, tras el entrenamiento, todos los puntos del espacio latente cercanos a la media reconstruyen datos similares a los originales. Así el espacio latente será continuo.[20] Los puntos cercanos del espacio latente codifican representaciones similares. Esto es lo que buscábamos. La continuidad, junto con la baja dimensionalidad, nos dan un espacio latente muy estructurado y que captura bien los ejes de variación de los datos.

La estructura de un VAE es la siguiente (fig: 2.23)

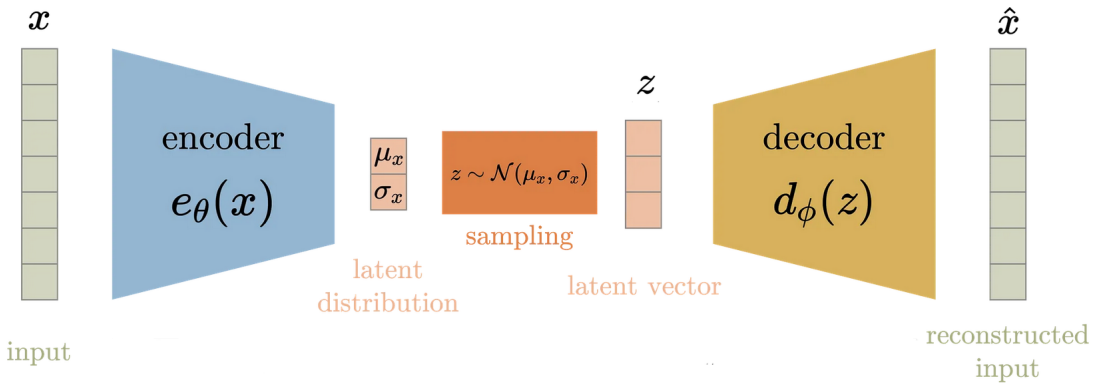


Figura 2.23: Estructura de un VAE [21]

en la que la capa latente ha sido sustituida por dos capas de media μ_x y varianza σ_x y un muestreador, siendo la entrada al codificador la muestra z.

La función de pérdidas de los VAE es la suma de dos términos:

- Pérdida de reconstrucción: Igual que en el autoencoder tradicional, mide la diferencia entre las salidas y las entradas.
- Pérdida de regularización: Mide la diferencia entre la distribución latente y una distribución normal estándar, para asegurarse de que la distribución latente esté estructurada. Se calcula usando la divergencia de Kullback-Leibler (KL) entre la distribución del espacio latente y una normal.

La imagen de (fig: 2.24) muestra una visualización de los dígitos reconstruidos a partir de puntos del espacio latente de un VAE. Nótese que, en vez de contener sólo los 9 digitos, es continuo, existiendo también posiciones intermedias. Tomando estos puntos intermedios podríamos crear nuevos números, si eso quisiéramos.

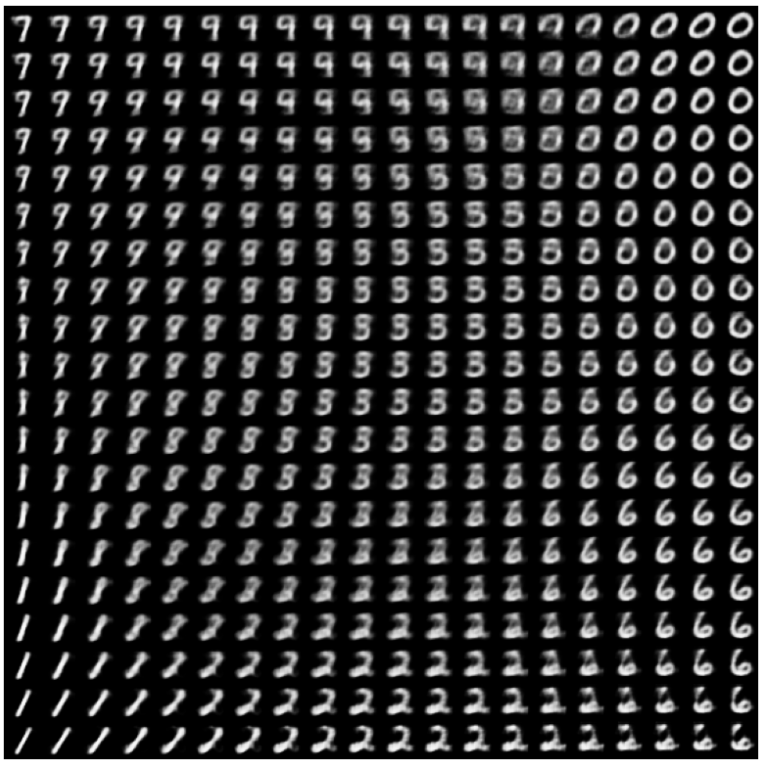


Figura 2.24: Visualización del espacio latente de un VAE

El espacio latente continuo, por tanto, permite generar información nueva, con estrucura similar a aquella para la que ha sido entrenado, lo que convierte a los

VAEs en útiles generadores de contenido, como vemos en (fig: 2.25), que muestra un espacio continuo de rostros de personas que no existen.

Figura 2.25: Rostros generados por un VAE [20]

Si bien nuestro objetivo no es la generación de contenido, los VAEs también se utilizan para la detección de anomalías, ya que tener un espacio latente continuo y estructurado permite evitar el sobreajuste, haciendo que funcionen mejor con datos nuevos que los autoencoders tradicionales.

2.7. Monitorización distribuida

En esta sección presentaremos un método de detección distribuida de fallos apropiado para procesos complejos al nivel de la planta, que permite reducir la complejidad de los mismos y monitorizar la redundancia de sus variables, siguiendo la metodología detallada en [26].

El proceso se compone de tres fases:

1. División de las variables del proceso en bloques según el método de mínima redundancia y máxima relevancia, para que se realize la detección de fallos en cada una de ellas por separado.

2. Cálculo de las estadísticas T^2 y Q de cada bloque aplicando un método de detección de fallos, en el caso de este trabajo, el VAE.

3. Combinación de las estadísticas de cada bloque según la inferencia bayesiana, para obtener las estadísticas globales del proceso.

2.7.1. División en bloques mediante mínima redundancia y máxima relevancia

Para dividir las variables del proceso, debemos hacerlo teniendo en cuenta las correlaciones entre ellas, tratándo de meter en el mismo bloque aquellas que estén más relacionadas entre sí. El método de mínima redundancia y máxima relevancia (o mRMR) nos proporciona un criterio para hacerlo.

Este método se basa en el concepto de información mutua, perteneciente a la teoría de la información. La información mutua es una estadística que mide la correlación entre dos variables, teniendo en cuenta relaciones lineales y no lineales. Es grande si las variables están muy relacionadas.

Para dos variables x_1 y x_2, se calcula como:

$$I(x_1, x_2) = \sum_{x_1} \sum_{x_2} p(x_1, x_2) log(\frac{p(x_1, x_2)}{p(x_1)p(x_2)}) \tag{2.24}$$

siendo $p(x_1)$ y $p(x_2)$ las funciones de densidad de probabilidad de x_1 y de x_2 y $p(x_1, x_2)$ la función de densidad de probabilidad conjunta.

Usando la fórmula de la entropía, se puede simplificar la expresión anterior. La entropía de una variable aleatoria cuantifica la incertidumbre asociada con sus posibles estados. Mide la cantidad de información esperada para describirla.

$$H(x) = -\sum_{x \in X} p(x) log\ p(x) \tag{2.25}$$

Siendo $H()$ la entropía y $H(x_1, x_2)$ la entropía conjunta de las variables x_1 y x_2:

$$H(x_1, x_2) = -\sum_{x_1} \sum_{x_2} log\ p(x_1, x_2) \tag{2.26}$$

entonces, la ecuación de I puede escribirse como:

$$I(x_1, x_2) = H(x_1) + H(x_2) - H(x_1, x_2) \tag{2.27}$$

El método de mRMR pretende obtener un conjunto S de variables que tengan la máxima dependencia de cierta variable c, maximizando la relevancia D de las variables de S respecto a c:

$$max\ D(S, c) \tag{2.28}$$

$$D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i, c) \tag{2.29}$$

siendo $|S|$ el número de variables del conjunto S. Como las variables redundantes tendrán necesariamente mucha relevancia, se añade el criterio adicional de minimizar la redundancia R:

$$min\ R(S) \tag{2.30}$$

$$R = \frac{1}{|S|} \sum_{x_i, x_j \in S} I(x_i, x_j) \tag{2.31}$$

Definiendo el operador $\Phi(D, R)$ como un operador que combina relevancia y rendundancia, la siguiente fórmula optimiza ambos parámetros:

$$max\ \Phi(D, R) \tag{2.32}$$

$$\Phi = D - R \tag{2.33}$$

En la parte práctica, usaremos el método de mRMR para dividir las variables del proceso en bloques automáticamente (sin conocimiento previo), considerando correlaciones tanto lineales como no lineales.

El proceso comenzará por escoger un conjunto de datos de funcionamiento normal $X = [x_1, x_2, ..., x_m] \in R_{n*m}$. Escogemos una variable arbitraria $c \in X$ para formar el primer bloque y, tomando un valor umbral I_c para la mRMR de c con cada una de las demás variables, incluiremos en el bloque de c aquellas variables cuya mRMR supere el umbral. Una vez construido el primer bloque, escogeremos como nueva c a la variable de las restantes cuya mRMR respecto a la c original sea mínima, y empezaremos de nuevo el proceso, hasta que todas las variables se hayan agrupado. Una vez hecha la ordenación, podemos agrupar las variables que hayan quedado en grupos demasiado pequeños, creando un bloque nuevo o juntando bloques pequeños.

El resultado de este proceso es la división de las variables en B sub-bloques:

$$X = [X1, X2, ..., X_B] \tag{2.34}$$

Aunque la elección de la primera variable sea arbitraria, las variables de alta relevancia entre sí quedarán en el mismo bloque al final del proceso.

Pasos del proceso de división automática de bloques:

1. Obtener un conjunto X de variables.

2. Escoger una variable arbitraria c de X y, para las $m - 1$ demás variables del proceso, comprobar si $\Phi > I_c$ para x_i. Si se cumple la desigualdad, incluiremos x_i en el bloque de c y eliminarla del conjunto X.

3. Escoger de entre las variables que han quedado en X la variable con menor Φ respecto de la c anterior como nueva c y repetir el paso 2 con la nueva c.

4. Repetir el paso 3 hasta que no queden variables o hasta que queden variables sueltas que no puedan formar bloque. En ese caso, juntar dichas variables en un bloque adicional.

2.7.2. Inferencia bayesiana

Los B bloques que hemos creado nos proporcionarán cada uno sus estadísticos, pero tendremos que combinar estos datos para poder medir el rendimiento global de la planta; para ello usamos la inferencia bayesiana.[27]

Según la regla de probabilidad condicional de Bayes, la probabilidad de fallo de T^2 en una muestra x_b perteneciente al bloque b es:

$$P_{T^2}(F|x_b) = \frac{P_{T^2}(x_b|F)P_{T^2}(F)}{P_{T^2}(x_b)} \tag{2.35}$$

$$P_{T^2}(F|x_b) = P_{T^2}(x_b|N)P_{T^2}(N) + P_{T^2}(x_b|F)P_{T^2}(F) \tag{2.36}$$

donde $P_{T^2}(x_b|N)$ y $P_{T^2}(x_b|F)$ se expresan como:

$$P_{T^2}(x_b|N) = e^{-T_b^2(x_b)/T_{b,lim}^2} \tag{2.37}$$

$$P_{T^2}(x_b|F) = e^{T_{b,lim}^2/-T_b^2(x_b)} \tag{2.38}$$

Los estadísticos finales que combinan la información de todos los bloques se calculan como:

$$BIC_{T^2} = \sum_{b=1}^{B}(\frac{P_{T^2}(x_b|F)P_{T^2}(F|x_b)}{\sum_{b=1}^{B} P_{T^2}(x_b|F)}) \tag{2.39}$$

$$BIC_Q = \sum_{b=1}^{B}(\frac{P_Q(x_b|F)P_Q(F|x_b)}{\sum_{b=1}^{B} P_Q(x_b|F)}) \tag{2.40}$$

Una vez calculados los dos BIC, podemos usarlos para la detección de anomalías del proceso general, tal como usabamos los otros, teniendo α como límite de confianza para establecer sus umbrales, tal como hacemos con los T^2 y Q.

Capítulo 3

Proceso Tennessee Eastman

3.1. Proceso Tennessee-Eastman

El proceso de Tennessee Eastman es un proceso industrial real que fue modelado computacionalmente en el lenguaje FORTRAN (con algunas modificaciones para proteger la propiedad comercial) en 1993 por Downs y Vogel [28], cuyos datos son usados muy frecuentemente para comparar el rendimiento de métodos de detección de fallos en ingeniería de procesos.

El proceso produce, a partir de cuatro reactantes (A, C, D y E), dos productos (G y H), además de un subproducto (F) y un componente interte (B) que son purgados (fig: 3.1). Las reacciones químicas en las que se generan los productos son:

$$A(gas) + C(gas) + D(gas) -> G(liq) \tag{3.1}$$

$$A(gas) + C(gas) + E(gas) -> H(liq) \tag{3.2}$$

De este proceso se pueden conocer más de 50 variables, como caudales, aperturas de válvulas, presiones, temperaturas, niveles de líquido, fracciones molares etc. Conocemos tanto variables manipuladas por el operador como no manipuladas. Las variables son muestreadas cada 3 minutos. La tabla 3.1 recoge las 52 variables del proceso.

Los aspectos particulares de este proceso químico no nos conciernen en este trabajo, ya que asumiremos que no conocemos la dinámica interna del proceso, analizando directamente las mediciones proporcionadas por los sensores. Nuestro único criterio para interpretar los datos, a la hora de desarrollar nuestros métodos de detección de fallos, será el conocimiento de que unos datos corresponden al comportamiento normal, sin fallos, y otros corresponden al comportamiento con posibles fallos.

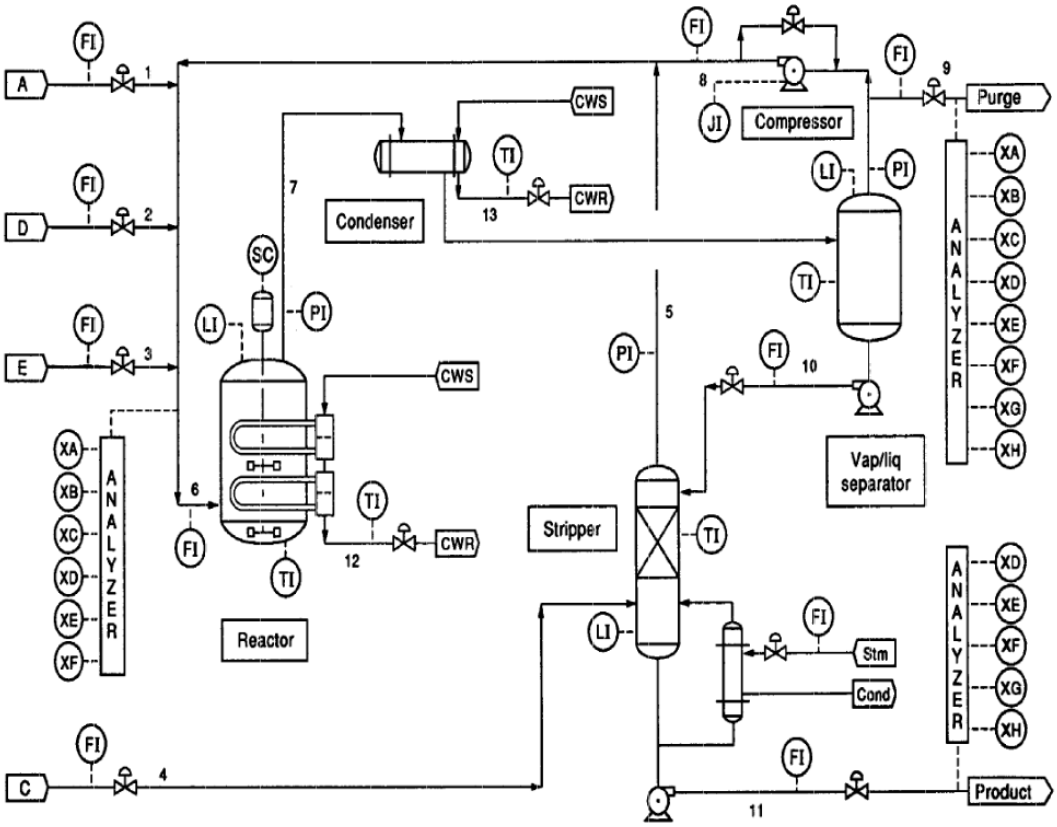


Figura 3.1: Proceso T-E [28]

Los fallos se corresponden con variaciones de diversas clases (escalón, aleatoria, desviación lenta etc.) de las variables del proceso. La tabla 3.2 recoge todos los fallos del proceso.

3.2. Datos

Los archivos con los que trabajaremos serán los siguientes:

1) Un archivo .csv de funcionamiento normal grande, que contiene mediciones de 480000 períodos de muestreo de las 52 variables del proceso. Estos datos se usarán para entrenar las redes neuronales. 2) 21 archivos .csv correspondientes al funcionamiento anómalo, de 960 períodos de muestreo. Se usarán para evaluar la capacidad de detección de fallos de los distintos programas. 3) Un archivo .csv de funcionamiento normal pequeño, de 960 períodos de muestreo. Se usará para entrenar el PCA, ya que esta técnica no requiere tantos datos.

Sabemos que, en los archivos de fallo, el fallo se produce a partir de la medición

Medida de Proceso	**Medida de Composición**	**Variable Manipulada**
Descripción de la Variable	Descripción de la Variable	Descripción de la Variable
1 Alimentación A	23 Comp. A de alimentación	42 Flujo de alimentación D
2 Alimentación D	24 Comp. B de alimentación	43 Flujo de alimentación E
3 Alimentación E	25 Comp. C de alimentación	44 Flujo de alimentación A
4 Alimentación Total	26 Comp. D de alimentación	45 Flujo total de alimentación
5 Flujo de Reciclaje	27 Comp. E de alimentación	46 Válvula de reciclaje compresor
6 Caudal de alimentación	28 Comp. F de alimentación	47 Válvula de purga
7 Presión del reactor	29 Comp. A de la purga	48 Flujo de producto separador
8 Nivel del reactor	30 Comp. B de la purga	49 Flujo de producto purgador
9 Temperatura del reactor	31 Comp. C de la purga	50 Válvula de vapor del purgador
10 Caudal de purga	32 Comp. D de la purga	51 Flujo enfriamiento de reactor
11 Temperatura separador	33 Comp. E de la purga	52 Flujo enfriamiento condensador
12 Nivel del separador	34 Comp. F de la purga	
13 Presión del separador	35 Comp. G de la purga	
14 Desbordamiento separador	36 Comp. H de la purga	
15 Nivel del purgador	37 Comp. D del producto	
16 Presión del purgador	38 Comp. E del producto	
17 Desbordamiento purgador	39 Comp. F del producto	
18 Temperatura del purgador	40 Comp. G del producto	
19 Caudal de vapor purgador	41 Comp. H del producto	
20 Trabajo del compresor		
21 T$^{\underline{a}}$ de salida del agua		
22 T$^{\underline{a}}$ de salida del agua		

Tabla 3.1: Descripción de las variables del proceso Tennessee Eastman [29]

160. Aunque no usaremos esta información para diseñar los modelos, sí que nos serviará para evualuar su funcionamiento, pues sabremos que no deberían detectar alarmas antes de ese instante de muestreo, y sí deberían hacerlo después.

ID de Fallo	Descripción	Tipo	Magnitud
IDV1	Relación de alimentación A/C, comp. B constante	Escalón	203 %
IDV2	Composición B, relación A/C constante	Escalón	105 %
IDV3	Temperatura de alimentación D	Escalón	5 %
IDV4	Temperatura de entrada del agua al reactor	Escalón	9 %
IDV5	Temperatura de entrada del agua al condensador	Escalón	15 %
IDV6	Pérdida de alimentación A	Escalón	342 %
IDV7	Pérdida de presión del cabezal C	Escalón	25 %
IDV8	Composición de alimentación A, B, C	Aleatorio	736 %
IDV9	Temperatura de alimentación D	Aleatorio	8 %
IDV10	Temperatura de alimentación C	Aleatorio	112 %
IDV11	Temperatura de entrada del agua al reactor	Aleatorio	567 %
IDV12	Temperatura de entrada del agua al condensador	Aleatorio	8 %
IDV13	Cinética de reacción	Desviación	16 %
IDV14	Válvula de agua al reactor	Bloqueo	1285 %
IDV15	Válvula de agua al condensador	Bloqueo	5 %
IDV16	Desconocido	Aleatorio	78 %
IDV17	Desconocido	Aleatorio	557 %
IDV18	Desconocido	Escalón	57 %
IDV19	Desconocido	Aleatorio	73 %
IDV20	Desconocido	Aleatorio	310 %
IDV21	Desconocido	Desconocido	? %

Tabla 3.2: Fallos del proceso T-E [30]

Los archivos de datos pertenecen al dominio público y pueden descargarse de [31] (datos de fallos) y [32] (datos de funcionamiento normal ampliado).

Capítulo 4

Aplicaciones y evaluación de métodos

4.1. Introducción

Habiendo visto en capítulos anteriores los métodos de aprendizaje automático y detección de fallos que emplearemos en este trabajo, así como los datos que usaremos para entrenar y evaluar el funcionamiento de nuestros modelos, en esta sección aplicaremos todo ello a la práctica, desarrollando cuatro aplicaciones de detección de fallos usando dichos conceptos.

Siguiendo la metodología de detección de anomalías en dos pasos que vimos en la sección 2.2, escribiremos dos programas para cada método: uno que aprenda el comportamiento normal de la planta, al que llamaremos entrenamiento (más apropiado para redes neuronales que para PCA), y otro que, a partir de los datos almacenados, detecte los posibles fallos en archivos de datos de la planta, al que llamaremos detección. En el caso de un proceso real, el entrenamiento se haría fuera de línea y la detección en línea.

Haremos esto para los siguientes métodos de aprendizaje automático:

1. PCA
2. Autoencoder
3. VAE
4. VAE distribuido.

Al comparar el rendimiento de los programas para cada uno de estos métodos, podremos hacernos una idea de su capacidad para la detección de anomalías.

Antes de pasar a la implementación, mencionaremos brevemente las principales librerías que hemos usado en los programas, programados todos en Python 3.12.3:

- Numpy (numpy.org/) para los cálculos con tensores a lo largo de todo el trabajo.

  Versión usada: 1.26.4.

- Pandas (pandas.pydata.org/)) para cargar los datos de la planta y para la división en bloques de la sección 4.5.

  Versión usada: 2.2.2.

- Matplotlib (matplotlib.org/) para dibujar todos los gráficos de control y muchos de las imágenes del capítulo 2.

  Versión usada: 3.8.4.

- Keras (keras.io/) para programar las redes neuronales en las secciones 4.3, 4.4 y 4.5.

  Version usada: 3.3.3.

4.2. PCA

El PCA empleado en la detección de anomalías se puede ver como un sistema de codificación-decodificación entrenado con datos de funcionamiento normal. El programa de entrenamiento se usará para seleccionar el número de componentes principales, establecer umbrales de los estadísticos T^2 y Q y guardar la media y varianza para normalizar los datos. El programa de detección obtendrá los componentes principales de los datos de fallo, calculará sus estadísticos T^2 y Q y realizará la detección de fallos, generando gráficas de control y estadísticos de rendimiento.

4.2.1. Entrenamiento

Empezamos cargando en memoria los datos de funcionamiento normal (para esta técnica usaremos el archivo de funcionamiento normal más pequeño, de dimensiones 960×52). Almacenamos los datos en la matriz X (tabla: 4.1).

Para tener en cuenta las diferentes escalas de cada variable, los normalizamos restando a cada dato la media de su columna y dividiéndolo por la desviación estándar de la misma, para que tengan media 0 y varianza 1, calculando así la matriz de datos normalizados X_n, siendo cada uno de sus elementos:

	1	2	3	...	50	51	52
0	0.24889	3702.3	4502.7	...	47.594	41.384	18.905
1	0.24904	3666.2	4526.0	...	47.508	41.658	18.976
2	0.25034	3673.3	4501.3	...	47.612	41.721	16.562
3	0.25109	3657.8	4497.8	...	47.459	40.836	20.094
4	0.24563	3698.0	4537.4	...	47.458	41.727	18.330
...	...	...	...	...	...	...	...
955	0.23955	3687.2	4581.0	...	47.510	41.466	16.998
956	0.23352	3625.4	4500.9	...	47.567	40.971	15.621
957	0.23440	3660.3	4535.7	...	47.338	41.891	21.744
958	0.23611	3645.0	4506.9	...	47.266	39.813	18.826
959	0.23729	3666.8	4511.1	...	47.165	40.500	18.353

Tabla 4.1: Datos de funcionamiento normal

$$X_{n,ij} = \frac{X_{i,j} - \mu_j}{\sigma_j} \tag{4.1}$$

Guardaremos en memoria las medias y las desviaciones típicas de los datos normales, ya que las usaremos para normalizar los datos de detección. Se deben normalizar con la misma media y desviación típica, en lugar de hacer que ambos conjuntos de datos tengan media 0 y desviación típica 1. Se debe hacer así para que ambos conjuntos tengan una escala consistente, pues escalar cada uno según su propia media distorsiona la información y hace imposible compararlos correctamente.

A continuación, calculamos la matriz de covarianza Σ, usando la función *cov()* de *numpy*. Véase la definición de covarianza en la sección 2.3.

A partir de la matriz de covarianza, podemos realizar la descomposición en valores y vectores propios, que podremos usar para obtener los componentes principales. Ordenando los vectores propios V en función de la magnitud de sus valores propios Λ asociados, y seleccionando hasta tener los que representan el 90 % de la variabilidad, descartando los demás, obtenemos la matriz T (no confudir con T^2), que contiene la información de los datos de funcionamiento normal proyectada en el espacio reducido de PCA y que usaremos para calcular el estadístico T^2:

$$T = X_n P \tag{4.2}$$

siendo P la matriz de vectores propios reducida.

Como lo que queremos hacer es modelar lo que consideramos como comportamiento normal, y después inferirlo para todos los puntos futuros, el cálculo de la covarianza y su descomposición sólo se hará para los datos de funcionamiento normal.

$$\hat{X_n} = TP^T \tag{4.3}$$

Calculamos la matriz de residuo E, que contiene la información que hemos descartado y que usaremos para calcular el estadístico Q:

$$E = X_n - \hat{X_n} \tag{4.4}$$

La tabla 4.2 recoge la varianza debida a cada componente principal. Vemos que el 90 % de la información está contenida en los 31 primeros componentes, que serán los que tomaremos para construir nuestro espacio reducido de dimensiones 960×31:

Componente	Varianza	Proporción	Acumulada
1	7.466214e+00	1.434315e-01	0.143431
2	4.564872e+00	8.769456e-02	0.231126
3	2.835147e+00	5.446527e-02	0.285591
4	2.156285e+00	4.142382e-02	0.327015
5	2.099565e+00	4.033419e-02	0.367349
6	1.994786e+00	3.832132e-02	0.405671
7	1.954597e+00	3.754925e-02	0.443220
8	1.682603e+00	3.232405e-02	0.475544
9	1.549010e+00	2.975762e-02	0.505302
10	1.377069e+00	2.645451e-02	0.531756
11	1.267539e+00	2.435036e-02	0.556106
12	1.226808e+00	2.356788e-02	0.579674
13	1.184271e+00	2.275071e-02	0.602425
14	1.141209e+00	2.192346e-02	0.624348
15	1.091112e+00	2.096107e-02	0.645310
16	1.045550e+00	2.008579e-02	0.665395
17	1.038536e+00	1.995103e-02	0.685346
18	1.014787e+00	1.949481e-02	0.704841
19	9.568096e-01	1.838102e-02	0.723222
20	9.362531e-01	1.798611e-02	0.741208
21	9.191049e-01	1.765668e-02	0.758865
22	8.976954e-01	1.724539e-02	0.776110
23	8.615052e-01	1.655015e-02	0.792661
24	8.361105e-01	1.606230e-02	0.808723
25	7.958499e-01	1.528886e-02	0.824012
26	7.837654e-01	1.505671e-02	0.839068
27	7.586519e-01	1.457426e-02	0.853643

28	7.361014e-01	1.414105e-02	0.867784
29	6.990761e-01	1.342977e-02	0.881213
30	6.960026e-01	1.337072e-02	0.894584
31	6.174918e-01	1.186247e-02	0.906447
32	6.093663e-01	1.170638e-02	0.918153
33	5.903873e-01	1.134177e-02	0.929495
34	5.402323e-01	1.037826e-02	0.939873
35	5.512830e-01	1.059055e-02	0.950464
36	5.090745e-01	9.779697e-03	0.960243
37	4.505631e-01	8.655649e-03	0.968899
38	4.259732e-01	8.183259e-03	0.977082
39	3.601984e-01	6.919677e-03	0.984002
40	3.005830e-01	5.774421e-03	0.989776
41	2.394066e-01	4.599177e-03	0.994376
42	1.667107e-01	3.202635e-03	0.997578
43	5.352975e-02	1.028346e-03	0.998607
44	4.012648e-02	7.708593e-04	0.999377
45	1.430575e-02	2.748240e-04	0.999652
46	7.469961e-03	1.435035e-04	0.999796
47	5.338860e-03	1.025634e-04	0.999898
48	3.290133e-03	6.320589e-05	0.999961
49	1.914315e-03	3.677540e-05	0.999998
50	9.178992e-05	1.763352e-06	1.000000
51	4.033111e-08	7.747903e-10	1.000000
52	4.261851e-08	8.187329e-10	1.000000

Tabla 4.2: Componentes principales

Calculamos T^2 aplicando la ecuación (2.12), obteniendo un vector de longitud n (el núm. de medidas). El límite superior de control de T^2 será el percentil 99 del T^2 de funcionamiento normal. Una vez calculemos el T^2 de los datos de detección, aquellas medidas que superen este umbral serán consideradas como anomalías:

$$UCL_{T^2} = percentil(T^2, 99) \tag{4.5}$$

Calculamos Q aplicando la ecuación (2.13) y su límite de control superior de la misma manera que el de T^2:

$$UCL_Q = percentil(Q, 99) \tag{4.6}$$

4.2.2. Detección

Con los datos de funcionamiento normal recogidos, nos falta calcular los estadísticos T^2 y Q de los archivos de datos de fallo y ver si superan o no sus umbrales.

Empezamos cargando los datos de fallo, los normalizamos mediante la media y desviación típica que calculamos antes según la ecuación (4.1), y calculamos T^2 y Q siguiendo el mismo proceso del entrenamiento, salvo que usando las matrices de valores propios reducidos Λ_a y de vectores propios reducidos P correspondientes a los datos normales.

Una vez calculados los T^2 y Q de fallo, se comprueba si superan el UCL_T y UCL_Q, considerando que se ha detectado una anomalía si se producen 10 mediciones seguidas por encima del umbral. Como sabemos de antemano que en los datos de fallo las anomalías se producen a partir de la medida 160, consideraremos como falsas alarmas aquellas medidas de T^2 y Q que estén por encima de los UCL antes de 160, y como alarmas no detectadas aquellas que a partir de dicha medida estén por debajo.

Los siguientes estadísticos serán nuestros criterios para evaluar el rendimiento de este método y de los demás:

1. Número de fallos detectados
2. Porcentaje de alarmas detectadas
3. Porcentaje de falsas alarmas
4. Tiempo de detección (dado a partir de la medida 160)

4.2.3. Resultados

A continuación presentaremos algunos gráficos de control para caracterizar la detección de anomalías con PCA y, al final de la sección, presentaremos los estadísticos completos para cada archivo de error.

Vemos (fig: 4.1) que el PCA detecta el fallo 2 en la medida 171 con T^2 y 174 con Q. El fallo tarda 11 mediciones en ser detectado, y continúa detectado hasta el final.

El fallo 3 (fig: 4.2) no es detectado en ningún momento por el PCA; aunque se observan ciertos picos en los gráficos de control, no duran lo suficiente como para que se llegue a activar la alarma. Este fallo no será detectado por ningún método.

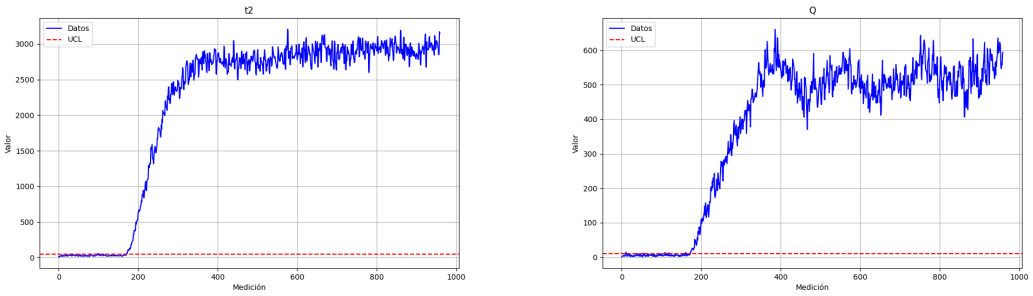


Figura 4.1: Detección del Fallo 2 mediante PCA. a) T^2, b) Q

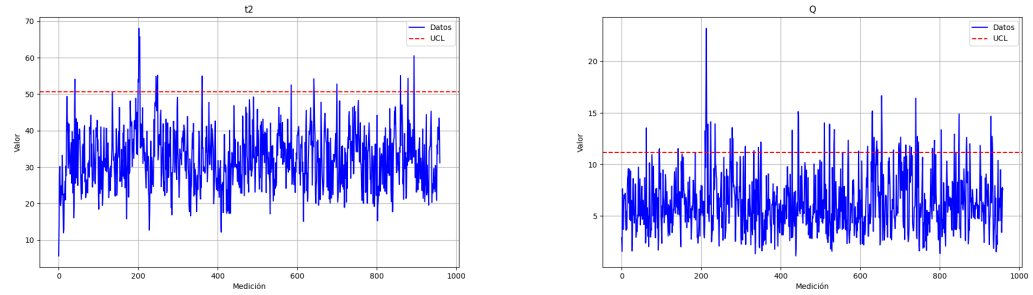


Figura 4.2: Detección del Fallo 3 mediante PCA. a) T^2, b) Q

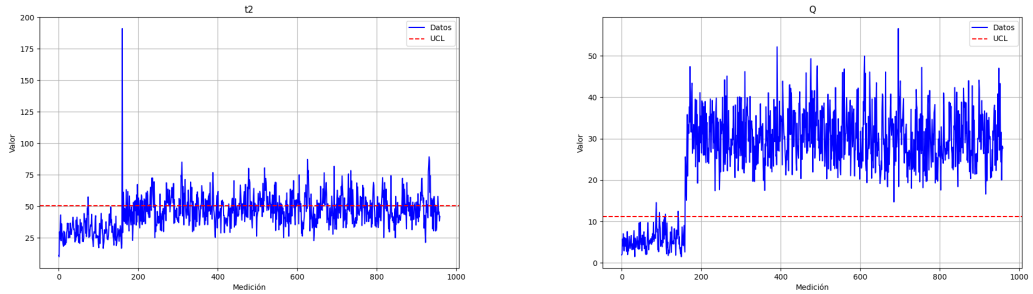


Figura 4.3: Detección del Fallo 4 mediante PCA. a) T^2, b) Q

El fallo 4 (fig: 4.3) se detecta mucho mejor usando el estadístico Q que T^2, lo que indica que la estructura de correlación de las variables ha cambiado más que lo que se ha desplazado de la media la distribución de los componentes principales.

El PCA no llega a detectar el fallo 19 (fig: 4.4), pese a tener muchos picos por encima del UCL de Q. Este fallo sí que será detectado por otros métodos.

A continuación presentamos los estadísticos de rendimiento de PCA para cada uno de los 21 archivos de fallo (tabla: 4.3). Los tiempos de alarma se miden a

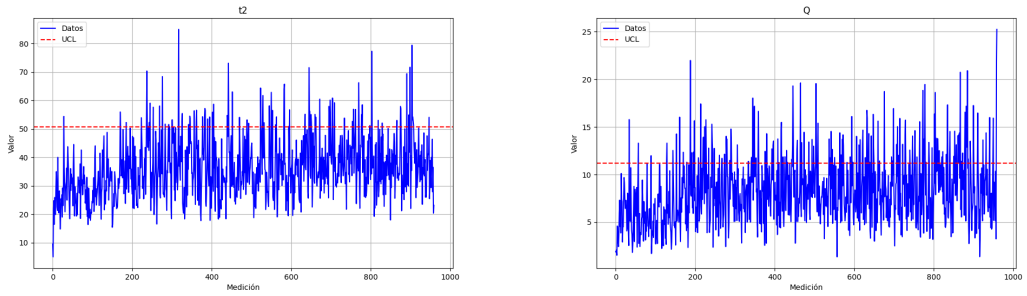


Figura 4.4: Detección del Fallo 19 mediante PCA. a) T^2, b) Q

partir de la medición 160 (si la alarma ha saltado). Además, hemos incluido las medias de los estadísticos de cada fallo (tabla: 4.5), con y sin los fallos 3, 9 y 15, que son fallos no detectables por ningún método de los estudiados en este trabajo:

Fallo	**Falsas** T^2	**Detect.** T^2	**Alarma** T^2	**Falsas** Q	**Detect.** Q	**Alarma** Q
01	1.258	99.25	6	3.774	99.75	2
02	1.887	98.625	11	4.403	98.625	14
03	1.258	1.875	NaN	2.516	7.5	NaN
04	0.629	41.875	504	2.516	100.0	0
05	0.629	25.75	10	2.516	33.875	0
06	0.629	98.875	9	2.516	100.0	0
07	0.0	100.0	0	2.516	100.0	0
08	0.0	97.25	22	4.403	96.875	17
09	3.145	3.375	NaN	5.031	4.5	NaN
10	1.887	30.75	103	0.0	46.0	47
11	0.0	51.25	50	5.031	69.375	6
12	1.887	98.625	21	6.289	95.125	22
13	1.887	94.625	45	1.258	95.125	40
14	0.629	99.5	0	5.031	99.875	1
15	1.258	2.5	NaN	3.145	6.125	NaN
16	3.774	15.125	310	3.145	43.5	193
17	0.629	78.625	28	3.145	95.625	21
18	1.258	89.375	92	5.66	90.125	83
19	0.629	12.25	NaN	4.403	21.875	NaN
20	2.516	31.25	86	3.774	55.875	84
21	1.258	41.375	504	3.774	50.125	249

Tabla 4.3: Estadísticos del PCA

	Fallos detectados
T^2	17
Q	17
Total	17

Tabla 4.4: Fallos detectados con PCA

Falsas T^2(%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
1,288	57,72	105,941
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
1,188	66,91	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo detección Q (mediciones)**
3,564	67,137	45,824
Falsas Q sin (%)	**Detectadas Q sin (%)**	
3,564	77,319	

Tabla 4.5: Estadísticos de resumen de PCA

Vemos que el PCA logra detectar fallos en la mayoría de archivos (tabla: 4.4), aunque tiene una media de alarmas detectadas relativamente baja. En las siguientes secciones, aplicaremos métodos de aprendizaje profundo para conseguir una mejor detección, ya que son capaces de aprender patrones más complejos que los del PCA.

4.3. Autoencoder

Siguiendo la metodología del apartado anterior, crearemos y entrenaremos una red neuronal de tipo autoencoder para detectar fallos en los datos de la planta.

4.3.1. Entrenamiento

Como las redes neuronales funcionan generalmente mejor cuantos más datos de entrenamiento tengan para estudiar, cargaremos el archivo grande de datos de funcionamiento normal, obteniendo así una matriz X de dimensiones 480000×52 (tabla: 4.6):

Tras cargar los datos, los normalizamos entre el máximo y el mínimo por columnas (variables), restando a cada valor el mínimo y dividiéndolo entre la diferencia entre el máximo y el mínimo, obteniendo los datos normalizados X_n, véase la tabla 4.7.

Después de haber probado diferentes números de capas y de neuronas por

	1	2	3	...	50	51	52
0	0.25171	3672.4	4466.3	...	47.300	42.100	15.345
1	0.25234	3642.2	4568.7	...	47.502	40.553	16.063
2	0.24840	3643.1	4507.5	...	47.479	41.341	20.452
3	0.25153	3628.3	4519.3	...	47.440	40.780	17.123
4	0.21763	3655.8	4571.0	...	47.530	41.089	18.681
...	...	...	...	...	...	...	...
479995	0.26428	3671.5	4510.8	...	52.721	40.927	18.854
479996	0.25132	3573.4	4483.9	...	52.908	41.412	18.847
479997	0.25105	3622.6	4433.3	...	53.099	40.150	16.791
479998	0.24521	3660.5	4485.2	...	53.157	41.065	17.097
479999	0.24373	3652.1	4471.8	...	53.407	41.596	16.697

Tabla 4.6: Datos de funcionamiento normal extendido

	1	2	3	...	50	51	52
0	0.482238	0.526136	0.364957	...	0.476661	0.702419	0.328431
1	0.484592	0.432172	0.656695	...	0.484882	0.395718	0.380963
2	0.469874	0.434972	0.482336	...	0.483946	0.551943	0.702078
3	0.481566	0.388923	0.515954	...	0.482359	0.440722	0.458516
4	0.354936	0.474487	0.663248	...	0.486021	0.501983	0.572505
...	...	...	...	...	...	...	...
479995	0.529192	0.523335	0.491738	...	0.697269	0.469865	0.585162
479996	0.480781	0.218108	0.415100	...	0.704879	0.566019	0.584650
479997	0.479773	0.371189	0.270940	...	0.712652	0.315821	0.434226
479998	0.457958	0.489110	0.418803	...	0.715012	0.497224	0.456614
479999	0.452430	0.462974	0.380627	...	0.725186	0.602498	0.427349

Tabla 4.7: Datos normalizados de funcionamiento normal extendido

capa, la estructura que mejores resultados ha producido para nuestros datos es la de la tabla 4.8, de 5 capas entrenables. La primera capa es la de entrada y no se entrena.

Usaremos con la función de activación Sigmoide en cada capa, ya que nuestro problema no es de clasificación y la función sigmoide es continua. Usaremos el optimizador Adam y como función de pérdidas el valor medio del error cuadrático.

Entrenando al modelo usando los datos X_n tanto como entrada como objetivo, tendremos una red capaz de detectar anomalías respecto a esos datos. Realizamos el entrenamiento durante 30 épocas, con un tamaño de batch de 1024, ya que tenemos muchos datos.

Capa (tipo)	Neuronas	Parámetros
capa de entrada (InputLayer)	(52)	0
x1 (Dense)	(52)	2756
x2 (Dense)	(42)	2226
h (Dense)	(32)	1376
x3 (Dense)	(42)	1386
x4 (Dense)	(52)	2236

Tabla 4.8: Capas del autoencoder

Una vez entrenada la red completa, la salida del codificador, porporcionada por la capa h, para los datos X nos dará una matriz H que contiene la información de X proyectada en el espacio latente del autoencoder. Con H podremos calcular los estadísticos T^2 y Q de los datos normales, según las ecuaciones 2.12 y 2.13, para así obtener sus umbrales, de nuevo usando un percentil de 99 %.

4.3.2. Detección

Cargando en el programa de detección la red entrenada con los datos de funcionamiento normal, sabemos que, si le metemos como entradas datos de funcionamiento anómalo, debería darnos salidas cuya distribución sea lo suficientemente distinta de la de los datos normales para poder ser detectada con los estadísticos T^2 y Q.

Igual que en el apartado anterior, calculamos los estadísticos T^2 y Q de cada fallo a partir de los datos proyectados obtenidos mediante la ecuación 2.18 y de su residuo, respectivamente, y detectamos los posibles fallos.

4.3.3. Resultados

Como hicimos antes, comentaremos algunos gráficos de control significativos y pasaremos a ver los estadísticos de cada fallo y los generales.

El estadístico Q detecta mejor el fallo 1 (fig: 4.5) que el T^2, que tiene unas oscilaciones mucho más grandes. Esto será algo muy frecuente en el autoencoder, cuyo estadístico Q tiende a ser mucho mejor que su T^2.

El fallo 4 (fig: 4.6) ha dejado de ser detectado con el estadístico T^2. El Q sigue detectándolo, aunque con una detección muy pobre, estando la mayor parte de los datos de fallo por debajo del UCL.

El fallo 5 (fig: 4.7) es detectado por ambos estadísticos, aunque el T^2 deja de detectarlo en torno a la medición 350, mientras que el Q lo detecta inmediatamente y no baja por debajo del UCL.

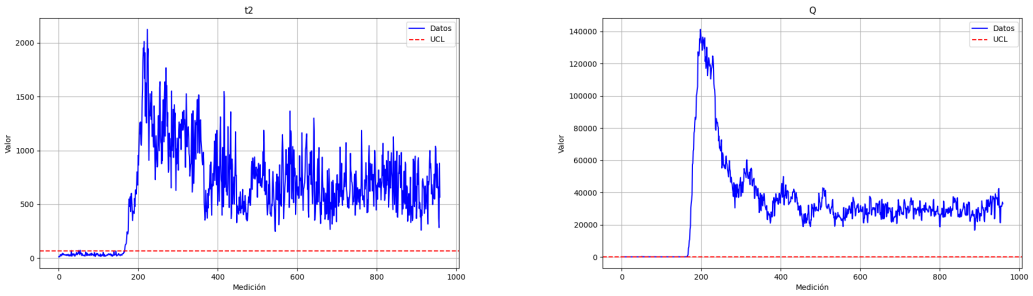


Figura 4.5: Detección del Fallo 1 mediante Autoencoder. a) T^2, b) Q

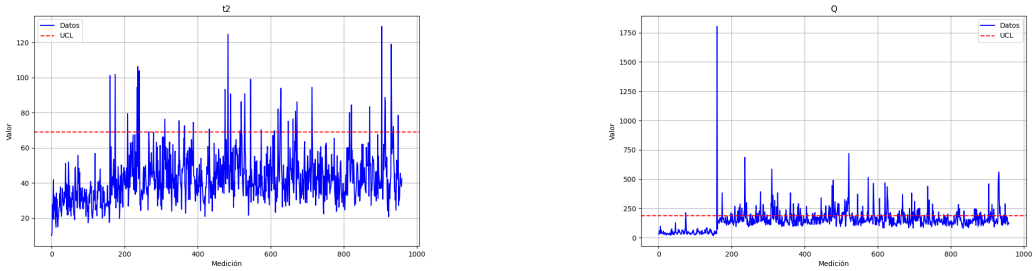


Figura 4.6: Detección del Fallo 4 mediante Autoencoder. a) T^2, b) Q

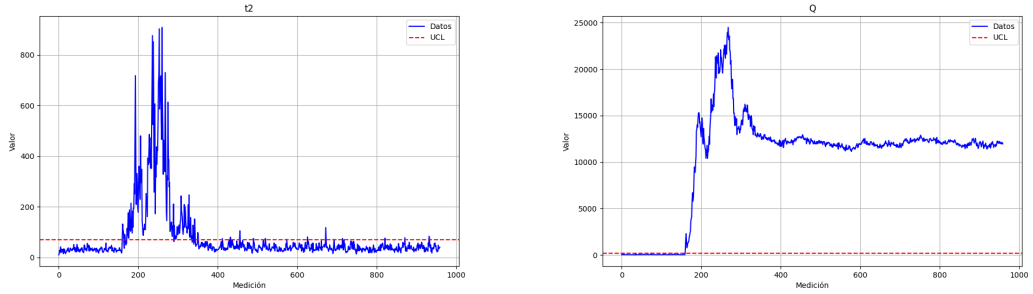


Figura 4.7: Detección del Fallo 5 mediante Autoencoder. a) T^2, b) Q

El autoencoder detecta a la perfección el fallo 7 (fig: 4.8) con ambos estadísticos: no tiene falsas alarmas y lo detecta inmediantamente sin que el valor caiga nunca por debajo del UCL.

El fallo 19 (fig: 4.9), que el PCA no llegaba a detectar, es detectable con el error Q del autoencoder, aunque la detección es mejorable.

Los resultados completos de detección con el autoencoder están recogidos en la tabla 4.9.

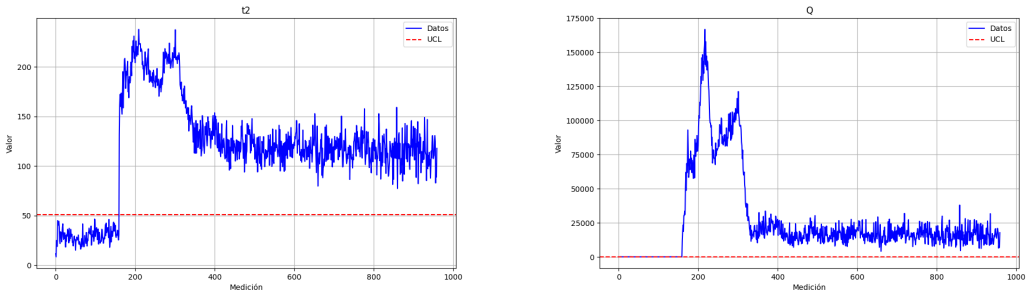


Figura 4.8: Detección del Fallo 7 mediante Autoencoder. a) T^2, b) Q

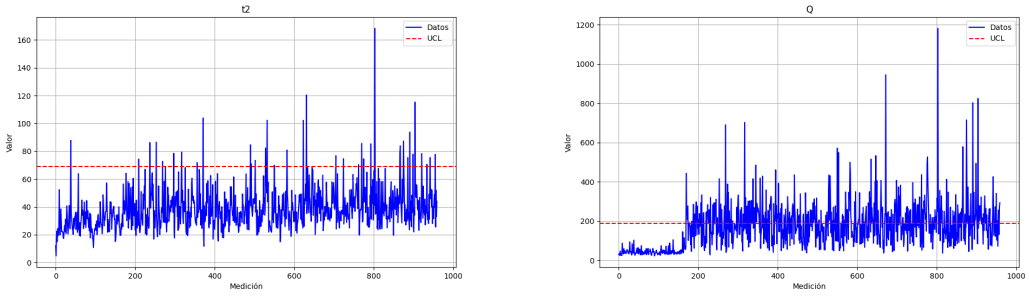


Figura 4.9: Detección del Fallo 19 mediante Autoencoder. a) T^2, b) Q

Usando el autoencoder, podemos detectar 18 fallos (tabla: 4.10), pudiéndose detectar ahora el fallo 19. Comparándolo sus datos con los de PCA, podemos ver que el porcentaje de falsas alarmas se ha reducido considerablemente en ambos estadísticos. Por el contrario, muchos parámetros han empeorado: ambos estadísticos son más lentos (tabla: 4.11) y el T^2 ahora tan solo detecta 15 fallos, habiendo dejado de detectar los fallos 4 y 15. Con las siguientes técnicas buscaremos mejorar la detección de fallos con T^2, aumentar los porcentajes de fallos detectados y reducir los tiempos de detección con T^2 y Q.

4.4. VAE

Para mejorar el rendimiento del autoencoder, entrenaremos un autoencoder variacional (VAE) para la detección de anomalías. Su espacio latente regularizado nos debería ayudar a reducir el sobreajuste y así poder aprender mejor los datos de funcionamiento normal.

Fallo	**Falsas T^2**	**Detect. T^2**	**Alarma T^2**	**Falsas Q**	**Detect. Q**	**Alarma Q**
01	1.887	99.125	7	0.629	99.5	4
02	0.0	98.375	14	1.258	98.75	10
03	0.0	1.25	NaN	0.0	0.875	NaN
04	0.0	5.0	NaN	0.629	24.375	354
05	0.0	24.125	12	0.629	100.0	0
06	0.0	99.0	9	0.629	100.0	0
07	0.0	100.0	0	0.0	100.0	0
08	1.258	97.0	22	0.629	97.375	21
09	0.629	1.0	NaN	0.0	0.75	NaN
10	0.0	16.875	148	0.0	70.625	30
11	1.258	21.0	NaN	0.0	43.5	143
12	0.0	97.0	21	0.0	99.75	2
13	0.629	94.625	46	0.0	94.75	45
14	1.258	88.125	31	0.629	99.875	1
15	0.0	1.375	NaN	0.0	2.125	NaN
16	1.258	7.5	310	1.887	67.875	11
17	0.629	67.25	30	0.629	83.375	25
18	0.629	88.75	92	0.0	89.625	84
19	0.629	4.375	NaN	0.0	49.375	184
20	0.629	20.625	300	0.0	67.125	74
21	1.258	30.125	723	0.0	34.5	706

Tabla 4.9: Estadísticos de Autoencoder

	Fallos detectados
T^2	15
Q	18
Total	18

Tabla 4.10: Fallos detectados con Autoencoder

4.4.1. Entrenamiento

La diferencia entre el entrenamiento del autoencoder y del VAE es que en este caso tenemos que crear una clase personalizada, para incluir el proceso de muestreo, así como la función de pérdidas personalizada (que, como vimos en el apartado 2.6,2, combina la pérdida de Kullbach-Leiber con la pérdida de reconstrucción).

Nuestro VAE estará formado por dos modelos de Keras, el encoder y el decoder. La salida del encoder estará unida con la del decoder por el muestreador.

Como probar diferentes estructuras, hemos obtenido la mejor detección de

Falsas T^2 (%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
0,569	51,429	137,75
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
0,489	59,826	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo detección Q (mediciones)**
0,389	68,78	69,353
Falsas Q sin (%)	**Detectadas Q sin (%)**	
0,419	79,938	

Tabla 4.11: Estadísticos de resumen de Autoencoder

fallos con la estructura recogica en las tablas 4.12 y 4.13.

Capa	**Neuronas**	**Parámetros**
Entrada (InputLayer)	(52)	0
x1 (Dense)	(48)	2,438
z media (Dense)	(28)	1,222
z log (Dense)	(28)	1,222

Tabla 4.12: Estructura del encoder del VAE

Capa	**Neuronas**	**Parámetros**
z (InputLayer)	(28)	0
x2 (Dense)	(48)	1,440
x3 (Dense)	(52)	2,548

Tabla 4.13: Estructura del decoder del VAE

Entrenaremos la red usando funciones de activación sigmoidal para cada capa, el optimizador *Adam* y la función de pérdida antes mencionada. Entrenaremos el modelo durante 50 épocas (pues este modelo se puede entrenar más sin peligro de sobreentrenarlo) y con un tamaño de batch de 1024.

4.4.2. Detección

La detección de fallos se realiza de la misma forma que la del el autoencoder, ya que la estructura del VAE y su función de pérdidas ya aseguran una mejor detección.

4.4.3. Resultados

Algunos gráficos de control que muestran la diferencia entre el rendimiento del VAE y el autoencoder son los siguientes:

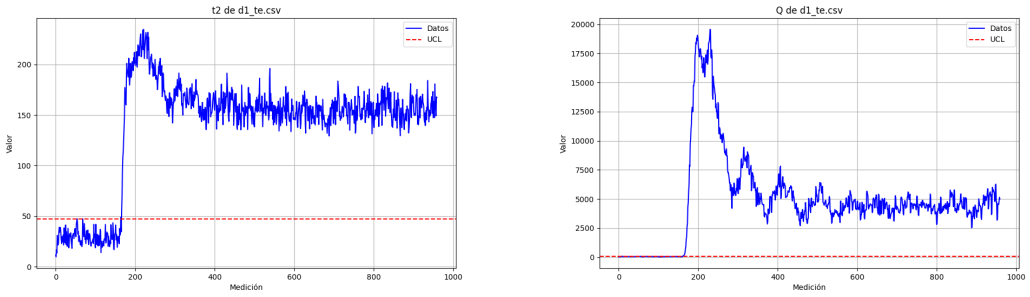


Figura 4.10: Detección del Fallo 1 mediante VAE. a) T^2, b) Q

Vemos que para el fallo 1 (fig: 4.10) el ruido en ambos estadísticos se ha reducido, siendo ahora la detección más precisa y rápida. Esto se cumple para casi todos los fallos.

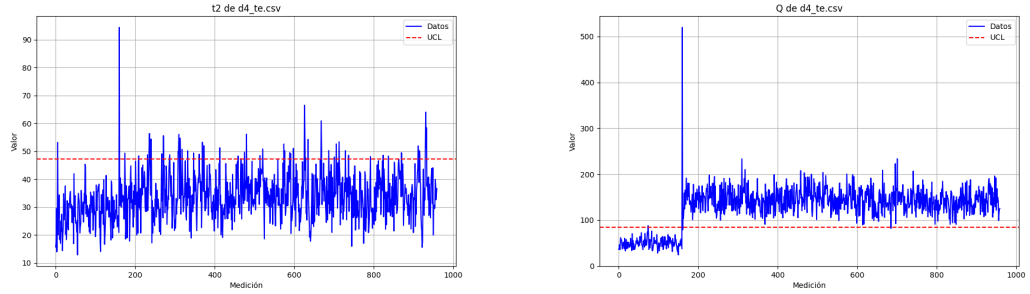


Figura 4.11: Detección del Fallo 4 mediante VAE. a) T^2, b) Q

El fallo 4 (fig: 4.11) sigue sin ser detectado por el T^2, aunque el Q ha mejorado mucho y detecta todo el error, mientras que el autoencoder sólo lo detectaba parcialmente.

Al igual que con el fallo 4, el VAE presenta una clara mejora en la detección del fallo 19 (fig: 4.12), que les resultaba difícil de detectar a los otros métodos. El VAE es la estructura que mejor rendimiento tiene con este error, de los vistos en este trabajo.

Los estadísticos generales del VAE quedan recogidos en la tabla 4.14:

Vemos que el rendimiento del VAE ha mejorado al del autoencoder simple. Detectan (tabla: 4.15 el mismo numero máximo de errores (18), pero $\mathbf{T^2}$ detecta

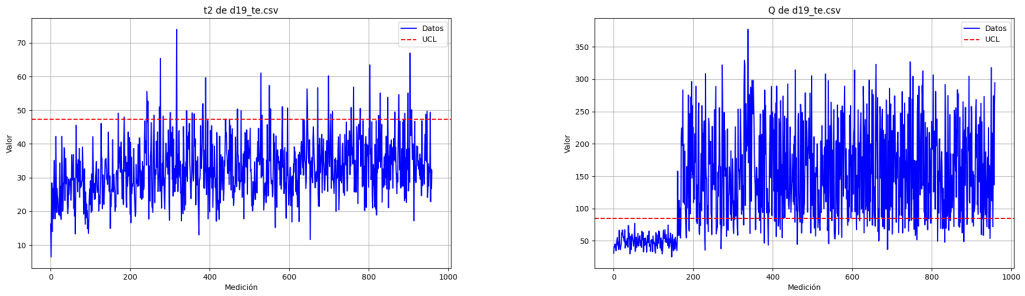


Figura 4.12: Detección del Fallo 19 mediante VAE. a) T^2, b) Q

Fallo	**Falsas** T^2	**Detect.** T^2	**Alarma** T^2	**Falsas** Q	**Detect.** Q	**Alarma** Q
01	1.258	99.375	6	0.629	99.75	2
02	1.258	98.625	11	1.258	98.625	11
03	0.0	1.5	NaN	0.629	0.875	NaN
04	0.629	7.125	NaN	0.0	100.0	0
05	0.629	26.5	14	0.629	100.0	0
06	0.0	98.625	11	0.0	100.0	0
07	0.0	100.0	0	0.0	100.0	0
08	0.629	97.375	22	0.629	97.875	19
09	0.629	0.625	NaN	2.516	0.875	NaN
10	0.629	19.75	148	0.0	83.75	23
11	0.629	26.5	462	0.0	69.375	6
12	0.0	98.0	21	0.0	99.875	1
13	0.0	94.5	47	0.0	95.375	37
14	0.629	93.75	1	0.0	100.0	0
15	0.629	1.375	NaN	0.629	3.375	NaN
16	1.887	8.375	311	0.629	85.375	8
17	0.0	70.25	30	0.629	95.375	21
18	1.258	89.125	92	0.629	90.0	83
19	0.0	7.0	NaN	0.0	82.5	170
20	0.629	21.75	96	0.629	87.75	66
21	1.258	35.25	527	1.258	41.75	484

Tabla 4.14: Estadísticos del VAE

un fallo más (el 11) y en menos tiempo (unos 20 mediciones menos). Q también ha mejorado reduciendo más de 25 mediciones su tiempo medio de detección (tabla: 4.16) y detectando el 90 % de alarmas (exceptuando los tres archivos cuyo fallo no detecta).

Por último, tomaremos el VAE y lo implementaremos de forma distribuida,

	Fallos detectados
T^2	16
Q	18
Total	18

Tabla 4.15: Fallos detectados con el VAE

Falsas T^2 (%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
0,599	52,161	112,438
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
0,629	60,66	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo detección Q (mediciones)**
0,509	77,738	42,833
Falsas Q sin (%)	**Detectadas Q sin (%)**	
0,384	90,41	

Tabla 4.16: Estadísticos de resumen del VAE

para tratar de adquirir cierto conocimiento de las causas de los fallos en el proceso, realizando una detección de fallos más localizada.

4.5. VAE distribuido

Usando las técnicas de división automática vistas en la sección 2.7 para dividir las variables del proceso en bloques, entrenaremos tantos VAEs como bloques hayamos creado, realizando con cada uno de ellos la detección de fallos en las variables pertenecientes a su bloque de forma independiente. Posteriormente, los estadísticos generados por cada VAE se combinarán mediante los coeficientes de Inferencia bayesiana para medir el rendimiento global del proceso.

Esta implementación, además de detectar anomalías como las demás, tiene la función adicional de proporcionarnos cierto conocimiento de las variables del proceso, ya que nos permite asignar la responsabilidad de los fallos a los bloques que estén produciendo resultados anómalos. Esto soluciona uno de los problemas de los métodos anteriores, a saber, que al proyectar las variables del proceso a un espacio latente (o de componentes principales en el caso del PCA), no es posible decir nada acerca de cuáles de ellas pueden estar comportándose de forma anómala. Este conocimiento podría ahorrar mucho tiempo en una planta industrial real, al permitir que los operarios se centren en solucionar los fallos de un subconjunto de las variables del proceso, en lugar de tener que revisarlas todas. Si sabemos que un bloque determinado es el que ha contribuido mayoritariamente a un fallo, conociendo las variables que lo componen, nos ocuparemos de ellas.

4.5.1. Entrenamiento

Lo primero que haremos será dividir las variables del proceso en bloques, siguiendo la metodología de mínima redundancia y máxima relevancia detallada en la sección 2.7. Tomaremos como umbral el valor de la media de Φ multiplicado por un parámetro de sesgo. Probaremos con tres sesgos distintos: 1, 1.5 y 0.5, para comprobar las divisiones en bloques que se forman. Como no conviene entrenar VAEs de dimensiones demasiado reducidas, tomaremos aquellos bloques de 3 variables o menos y los combinaremos en uno.

Los bloques creados por el programa para los distintos valores del sesgo son los siguientes (tablas 4.17, 4.18 y 4.19 respectivamente):

Sesgo 1	**Variables**
Bloque 1	3, 4, 7, 8, 10, 11, 13, 16, 18, 19, 22, 25, 31, 35, 43, 47, 50
Bloque 2	5, 17, 42, 46, 52
Bloque 3	2, 9, 21, 51
Bloque 4	20, 27, 28, 33, 34, 36
Bloque 5	6, 23, 24, 29, 38, 39, 41
Bloque 6	14, 26, 32, 40
Bloque 7	1, 12, 15, 30, 37, 44, 48, 49

Tabla 4.17: Bloques creados con sesgo 1

Sesgo 1.5	**Variables**
Bloque 1	7, 8, 10, 11, 13, 16, 18, 19, 22, 25, 31, 35, 43, 47, 50
Bloque 2	5, 17, 46, 52
Bloque 3	2, 9, 21, 42, 51
Bloque 4	20, 27, 28, 33, 36
Bloque 5	4, 6, 23, 24, 38, 39, 41
Bloque 6	3, 29, 30, 34, 37
Bloque 7	1, 12, 14, 15, 26, 32, 40, 44, 48, 49

Tabla 4.18: Bloques creados con sesgo 1,5

Vemos que, aunque hay ciertos cambios según el sesgo que elijamos, las correlaciones principales entre las variables del proceso se mantienen. Hay ciertas variables que siempre quedan juntas, tales como la 5 y la 17, la 2 y la 9 o la 47 y la 50.

Una vez creados los B bloques, pasamos al entrenamiento de los VAEs de cada uno de ellos. Dividimos la matriz de datos de entrenamiento X en B matrices, cada una con las variables correspondientes a un bloque, y pasamos a entrenar

Sesgo 0.5	**Variables**
Bloque 1	3, 4, 6, 7, 8, 10, 11, 13, 16, 18, 19, 20, 21, 22, 25, 29, 31, 33, 35, 37, 41, 43, 47, 50
Bloque 2	5, 17, 34, 42, 46, 52
Bloque 3	15, 23, 26, 27, 28, 30, 36, 38, 40, 45, 49
Bloque 4	12, 24, 32, 48
Bloque 5	1, 2, 9, 14, 39, 44, 48, 51

Tabla 4.19: Bloques creados con sesgo 0,5

un autoencoder con los datos de funcionamiento normal de su bloque. Una vez realizado el entrenamiento, almacenamos los estadísticos T^2 y Q y los UCL de cada bloque.

Una vez obtenidos todos los estadísticos de entrenamiento, los combinamos para calcular BIC_{T^2} y BIC_Q según las expresiones (2.39) y (2.40). Con el percentil 99 de esos dos parámetros obtenemos los UCL globales de T^2 y Q para la implementación distribuida.

4.5.2. Detección

Para cada archivo de fallo, cargamos los datos de fallo X_f y los distribuimos en B partes según los bloques creados. Después, realizamos la detección en cada parte de los datos de fallo con cada uno de los VAEs entrenados y recogemos sus estadísticos. Una vez completada la detección con cada VAE, calculamos los BIC_{T^2} y BIC_Q de fallo, usando los T^2 y Q de detección y los UCL_{T^2} y UCL_Q del entrenamiento de cada bloque. Con esos BIC_{T^2}, BIC_Q y los UCL globales obtenidos durante el entrenamiento, realizamos la detección de fallos global.

Los pasos a seguir son estos, para cada archivo de fallo:

1. Cargar datos del fallo X_f
2. Dividir X_f en B matrices $X_{f,b}$ según los bloques creados.
3. Realizar la detección sobre cada $X_{f,b}$ con su VAE correspondiente, obteniendo los T^2 y Q de cada bloque.
4. Combinar los T^2 y Q de cada bloque para obtener los BIC_{T^2} y BIC_Q globales.
5. Realizar la detección de fallos a nivel de bloque con los estadísticos y los UCL individuales.
6. Realizar la detección de fallos global con los BIC y los UCL globales.

Los VAEs que entrenaremos serán similares a los de la sección anterior, pero con un número de neuronas en cada capa dependiente de las variables en su bloque. En los bloques con más de 9 variables, usaremos un VAE con 4 capas latentes (a parte de las de z_{media}, z_{var} y z) y en aquellos con menos de 9 usaremos VAEs con 2 capas latentes y una dimensión latente más reducida.

4.5.3. Resultados

Presentaremos algunos ejemplos de la detección del mismo fallo en cada bloque (para un sesgo de 1), para ilustrar la técnica de detección distribuida. Después, presentaremos los datos de todos los fallos, para los distintos sesgos elegidos.

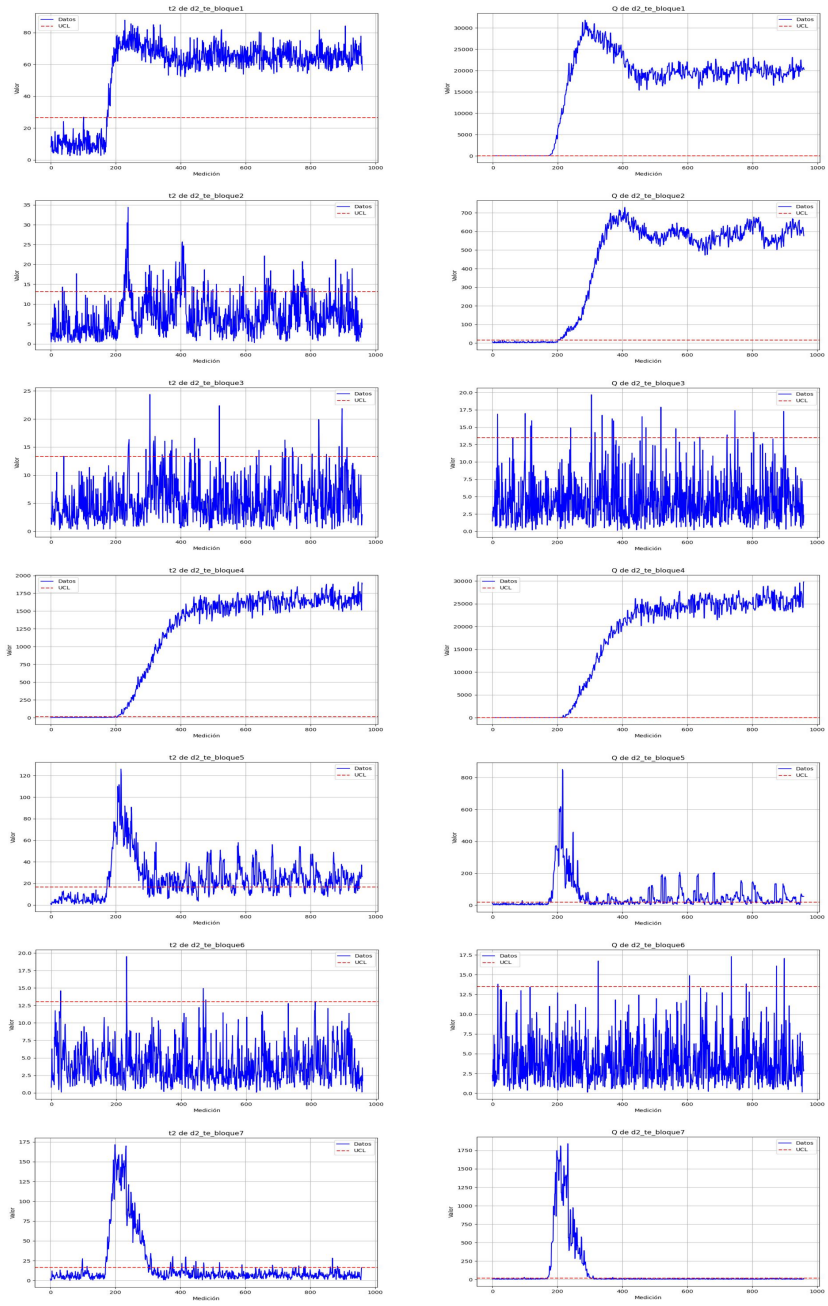


Figura 4.13: Detección del fallo 2 mediante autoencoder distribuido con los 7 bloques

El fallo 2 (fig: 4.13) lo detectan los bloques 1, 2, 4, 5 y 7. Con esto podemos ver que afecta a gran parte de las variables de la planta, por lo que su solución resultaría más difícil.

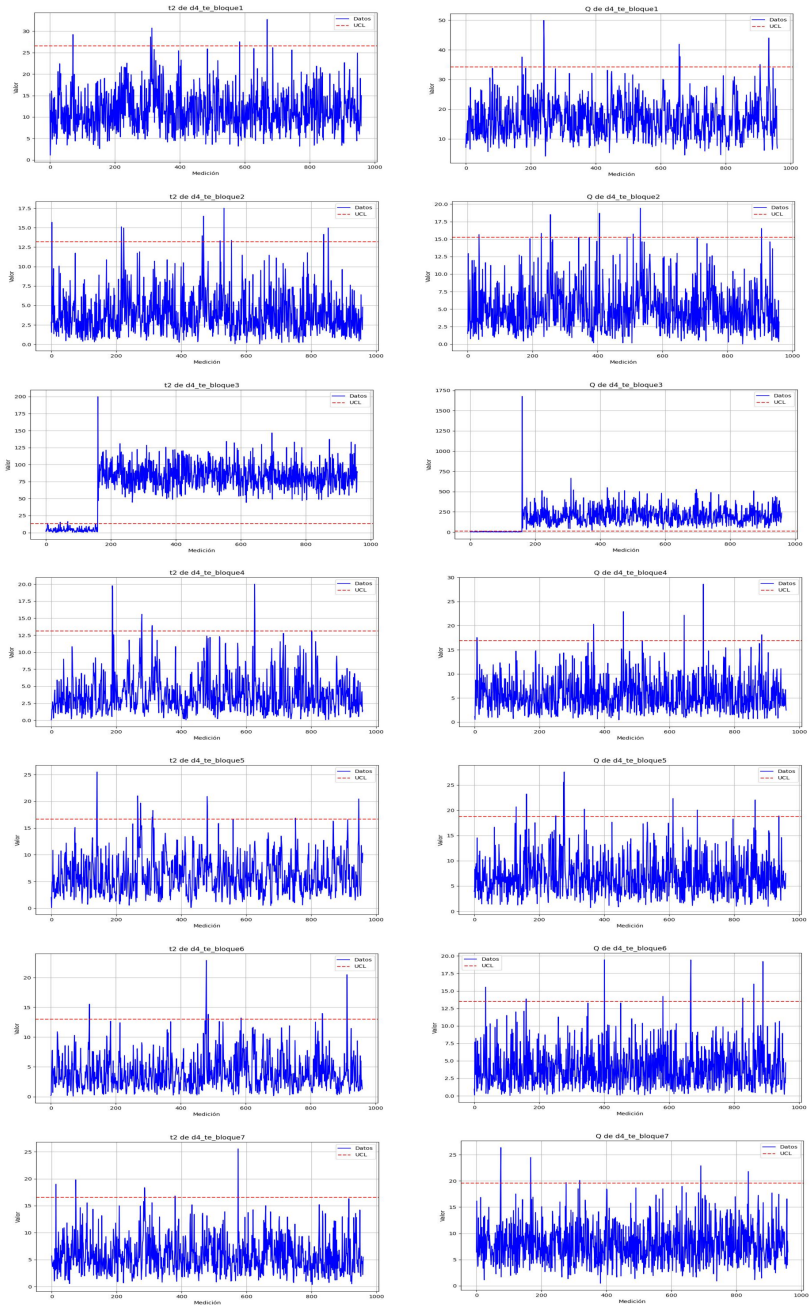


Figura 4.14: Detección del fallo 4 mediante autoencoder distribuido con los 7 bloques

Las variables del bloque 3 son las responsables del fallo 4 (fig: 4.14), mientras que las demás no presentan desviaciones estatísticamente significativas respecto al comportamiento normal. Solucionar este fallo, por tanto, implicaría comprobar

las partes del proceso relacionadas con dichas variables. Es en este tipo de fallos donde la detección distribuida presenta sus mayores ventajas.

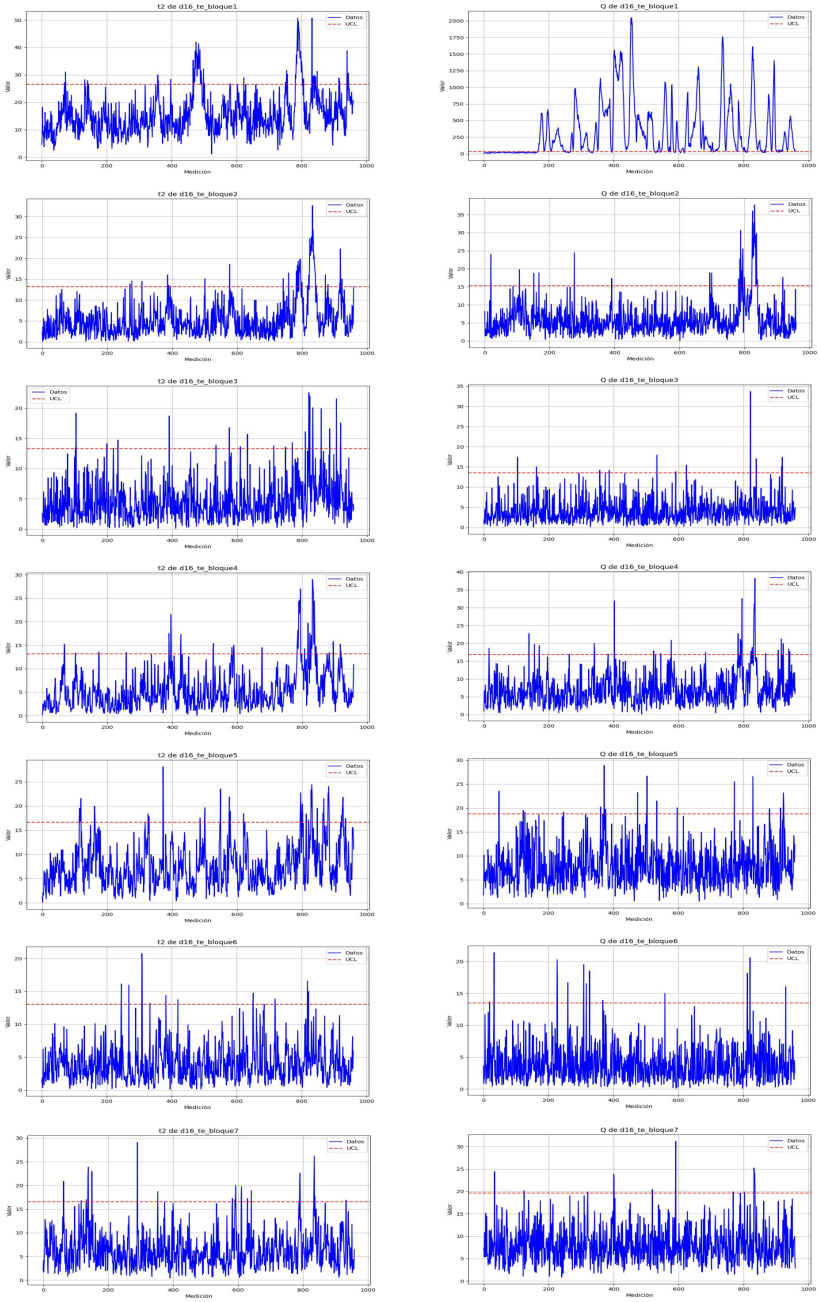


Figura 4.15: Detección del fallo 16 mediante autoencoder distribuido con los 7 bloques

El fallo 16 (fig: 4.15) es detectado principalmente por el bloque 1 y algo menos por el bloque 2; es también un fallo bastante localizado (si bien es cierto que el bloque 1 contiene muchas variables).

A continuación presentamos la lista de los fallos detectados por cada bloque con los distintos sesgos (tablas 4.20, 4.21 y 4.22), para poder observar qué bloques son los más relevantes, a la hora de la detección de fallos.

Fallo/Bloque	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	SÍ	SÍ	SÍ	SÍ	SÍ	NO	SÍ
2	SÍ	SÍ	NO	SÍ	SÍ	NO	SÍ
3	NO	NO	NO	NO	NO	NO	NO
4	NO	NO	SÍ	NO	NO	NO	NO
5	SÍ	SÍ	SÍ	SÍ	SÍ	NO	SÍ
6	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
7	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
8	SÍ	SÍ	SÍ	SÍ	SÍ	NO	SÍ
9	NO	NO	NO	NO	NO	NO	NO
10	SÍ	SÍ	NO	SÍ	SÍ	NO	SÍ
11	NO	NO	SÍ	NO	NO	NO	NO
12	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
13	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
14	NO	NO	SÍ	NO	NO	NO	NO
15	NO	NO	NO	NO	NO	NO	NO
16	SÍ	SÍ	NO	SÍ	NO	NO	NO
17	SÍ	SÍ	NO	NO	NO	NO	NO
18	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
19	SÍ	SÍ	NO	NO	NO	NO	NO
20	SÍ	SÍ	NO	SÍ	NO	NO	NO
21	SÍ	SÍ	SÍ	NO	SÍ	NO	NO

Tabla 4.20: Detección por bloque con sesgo 1

Vemos que hay fallos que sólo afectan a un conjunto de variables, y que tan solo son detectados por ciertos bloques, como los fallos 4, 11 o 17, mientras que otros como el 12 y el 13 afectan a las variables del proceso de forma más generalizada. También se observa que ciertos bloques detectan la mayoría de los fallos, en este caso los bloques 1 y 2, mientras que otros tienen una contribución menor, como el bloque 6, que tan sólo detecta 6 fallos.

Es muy significativo que el bloque 2 sea el que más fallos detecta (15 junto con el bloque 1), ya que contiene tan sólo 5 variables. Eso nos dice que las variables

del bloque 2 son muy importantes para la detección de fallos.

Fallo/Bloque	1	2	3	4	5	6	7
1	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
2	SÍ	SÍ	NO	SÍ	SÍ	SÍ	NO
3	NO	NO	NO	NO	NO	NO	NO
4	NO	NO	SÍ	NO	NO	NO	NO
5	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
6	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
7	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
8	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
9	NO	NO	NO	NO	NO	NO	NO
10	SÍ	SÍ	NO	SÍ	SÍ	SÍ	NO
11	NO	NO	SÍ	NO	NO	NO	NO
12	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
13	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
14	NO	NO	SÍ	NO	NO	NO	NO
15	NO	NO	NO	NO	NO	NO	NO
16	SÍ	SÍ	NO	SÍ	NO	NO	NO
17	SÍ	NO	SÍ	NO	NO	NO	NO
18	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
19	SÍ	SÍ	NO	NO	NO	NO	NO
20	SÍ	SÍ	NO	SÍ	NO	NO	NO
21	SÍ	SÍ	NO	NO	NO	NO	NO

Tabla 4.21: Detección por bloques con sesgo 1,5

Este sesgo ofrece resultados similares a los del anterior, aunque mejora la detección de fallos por parte del bloque 6. Cualquiera de los dos funcionaría bien.

Fallo/Bloque	1	2	3	4	5
1	SÍ	SÍ	SÍ	NO	SÍ
2	SÍ	SÍ	NO	SÍ	SÍ
3	NO	NO	NO	NO	NO
4	NO	NO	NO	NO	SÍ
5	SÍ	SÍ	SÍ	NO	SÍ
6	SÍ	SÍ	SÍ	NO	SÍ
7	SÍ	SÍ	SÍ	SÍ	SÍ
8	SÍ	SÍ	SÍ	SÍ	SÍ
9	NO	NO	NO	NO	NO
10	NO	SÍ	SÍ	NO	NO
11	NO	NO	NO	NO	SÍ
12	SÍ	SÍ	SÍ	SÍ	SÍ
13	SÍ	SÍ	SÍ	SÍ	SÍ
14	SÍ	NO	NO	NO	SÍ
15	NO	NO	NO	NO	NO
16	SÍ	SÍ	NO	NO	NO
17	SÍ	NO	NO	NO	SÍ
18	SÍ	SÍ	SÍ	SÍ	SÍ
19	NO	SÍ	NO	NO	NO
20	SÍ	SÍ	NO	NO	NO
21	SÍ	SÍ	NO	NO	NO

Tabla 4.22: Detección por bloques con sesgo 0,5

Con este sesgo el fallo 19 deja de detectarse, presentando la misma detección de fallos que el PCA, por lo que este es el peor sesgo de los tres.

A continuación, mostraremos algunos gráficos de fallo correspondientes a los estadísticos combinados BIC_{T^2} y BIC_Q, para compararlos con la detección que proporcionan los bloques individuales y el resto de métodos.

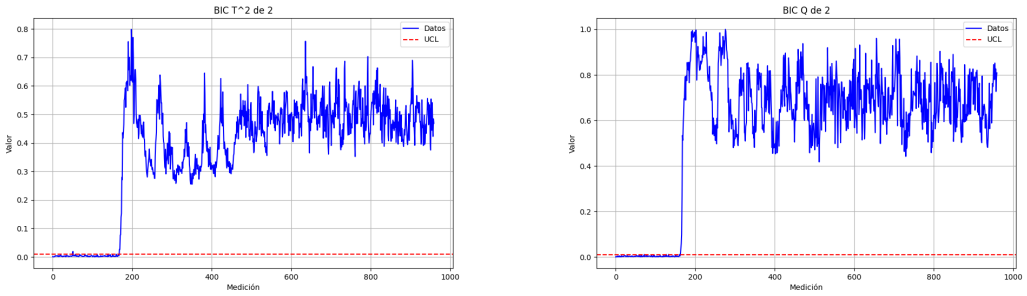


Figura 4.16: Detección del Fallo 1 mediante BIC. a) T^2, b) Q

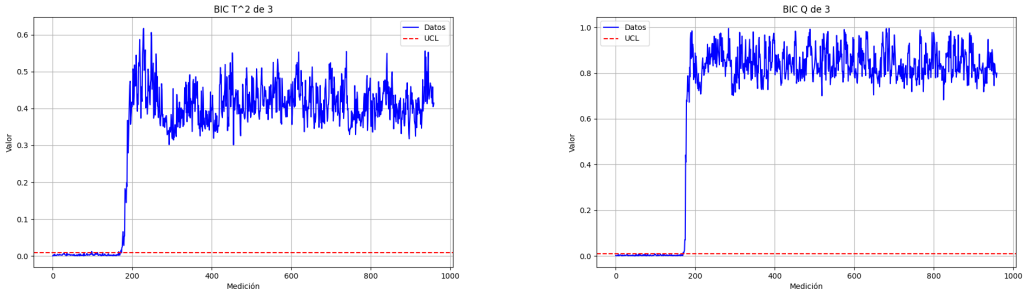


Figura 4.17: Detección del Fallo 2 mediante BIC. a) T^2, b) Q

El fallo 1 (fig: 4.16) y el fallo 2 (fig: 4.17) se detctan de forma similar a la detección que teníamos con el VAE, con algo más de ruido en el fallo 1, aunque no afecta a la detección, que sigue siendo completa.

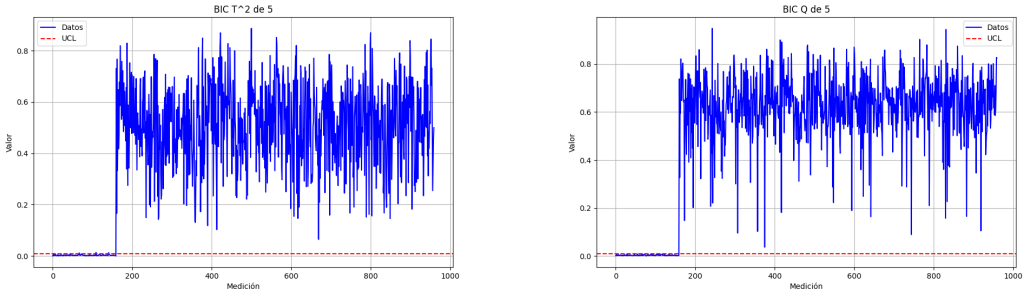


Figura 4.18: Detección del Fallo 4 mediante BIC. a) T^2, b) Q

La detección del fallo 4 (fig: 4.18) mejora mucho con este método en cuanto al estadístico T^2, que detecta todo el error y muy rápidamente.

Por último, presentamos los estadísticos globales obtenidos con el método de la inferencia bayesiana para cada sesgo.

Para un umbral de $\Phi * 1$, los estadísticos de detección de fallos pueden verse en la tabla 4.23.

Fallo	**Falsas** T^2	**Detect.** T^2	**Alarma** T^2	**Falsas** Q	**Detect.** Q	**Alarma** Q
01	3.145	99.0	8	1.258	99.625	3
02	1.887	98.625	14	3.145	98.75	10
03	1.258	1.625	NaN	2.516	2.5	NaN
04	2.516	100.0	0	3.145	100.0	0
05	1.887	25.25	13	3.145	100.0	0
06	1.258	99.5	5	3.145	100.0	0
07	0.0	38.5	0	0.0	37.375	0
08	0.0	96.25	23	1.258	98.0	17
09	2.516	1.875	NaN	2.516	2.875	NaN
10	0.629	25.25	95	1.258	81.5	25
11	2.516	82.875	5	1.887	74.5	5
12	0.629	96.125	24	1.887	99.75	2
13	0.0	94.5	48	1.258	94.875	41
14	1.258	100.0	0	0.629	100.0	0
15	0.0	2.25	NaN	1.258	2.625	NaN
16	2.516	12.375	623	1.887	85.875	7
17	1.258	95.625	21	1.258	93.125	21
18	0.629	87.875	99	0.629	90.375	83
19	0.0	19.125	NaN	1.258	25.625	415
20	1.258	44.0	80	0.629	80.25	65
21	1.887	25.125	669	3.145	44.375	483

Tabla 4.23: Estadísticos del VAE distribuido de sesgo 1

	Fallos detectados
T^2	17
Q	18
Total	18

Tabla 4.24: Fallos detectados con el VAE destribuido de sesgo 1

Falsas T^2 (%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
1,228	59,321	101,588
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
1,293	68,889	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo eteccion Q (mediciones)**
1,767	72	65,389
Falsas Q sin (%)	**Detectadas Q sin (%)**	
1,712	83,556	

Tabla 4.25: Estadísticos de resumen del VAE destribuido con sesgo 1

Respecto del VAE, el T^2 ha mejorado, pudiendo detectar ahora el fallo 4, que antes no detectaba, y disminuyendo su tiempo de detección medio unas 10 mediciones (tabla: 4.25). El estadístico Q, por el contrario, ha aumentado su tiempo de detección medio y disminuido su precisión, aunque sigue detectando 18 fallos (tabla: 4.24). La mala detección del fallo 19 con este método es la responsable de la detección media más lenta.

Para un umbral de $\Phi * 1,5$ los datos están recodigos en la tabla: 4.26.

Fallo	**Falsas T^2**	**Detect. T^2**	**Alarma T^2**	**Falsas Q**	**Detect. Q**	**Alarma Q**
01	4.403	99.125	8	1.258	99.625	3
02	3.145	98.5	12	1.887	98.75	10
03	2.516	3.625	NaN	1.258	2.125	NaN
04	2.516	97.25	0	1.258	100.0	0
05	1.258	24.375	15	1.258	100.0	0
06	0.0	99.375	5	0.629	100.0	0
07	0.629	35.5	0	1.258	37.875	0
08	0.0	96.75	21	1.258	97.875	17
09	1.258	2.75	NaN	1.887	2.875	NaN
10	1.887	22.75	100	0.629	81.125	25
11	1.258	73.375	5	2.516	77.125	5
12	1.258	95.0	22	3.774	99.75	2
13	0.629	94.25	46	0.0	95.125	42
14	0.0	100.0	0	0.629	100.0	0
15	1.258	2.0	NaN	1.258	3.25	NaN
16	0.629	11.875	623	0.0	85.25	7
17	3.145	95.125	21	1.887	91.25	21
18	1.887	87.875	100	1.258	90.125	83
19	0.0	21.0	NaN	0.629	28.25	415
20	0.629	44.375	80	1.258	82.0	65
21	1.887	26.875	624	3.774	44.25	471

Tabla 4.26: Estadísticos del VAE distribuido con sesgo 1,5

	Fallos detectados
T^2	17
Q	18
Total	18

Tabla 4.27: Fallos detectados con el VAE distribuido de sesgo 1,5

La detección de fallos con este sesgo es muy similar a la que obtuvimos con el anterior (tabla: 4.27), marginalmente más rápida (tablas: 4.28) con menos falsas alarmas en Q y con un procentaje muy parecido de fallos detectados.

Para un umbral de $\Phi * 0,5$ los estadísticos están en la tabla 4.29.

Falsas T^2 (%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
1,438	58,655	98,941
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
1,398	67,965	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo detección Q (mediciones)**
1,408	72,202	64,778
Falsas Q sin (%)	**Detectadas Q sin (%)**	
1,398	83,799	

Tabla 4.28: Estadísticos de resumen del VAE distribuido de sesgo 1,5

Fallo	**Falsas T^2**	**Detect. T^2**	**Alarma T^2**	**Falsas Q**	**Detect. Q**	**Alarma Q**
01	1.887	98.75	170	0.629	99.75	162
02	1.258	98.5	172	2.516	98.5	172
03	1.258	1.5	NaN	2.516	2.25	NaN
04	0.629	63.75	418	1.258	100.0	160
05	1.258	21.625	177	2.516	100.0	160
06	3.145	98.625	171	1.258	100.0	160
07	0.0	96.0	160	0.0	100.0	160
08	0.0	94.0	188	1.887	98.125	175
09	1.258	2.5	NaN	2.516	2.5	NaN
10	0.629	20.25	256	1.887	81.0	185
11	1.887	58.375	396	0.0	75.875	165
12	1.258	90.0	191	4.403	99.75	162
13	0.629	92.25	216	0.629	95.125	199
14	1.887	89.625	192	0.629	100.0	160
15	1.887	2.25	NaN	1.887	3.0	NaN
16	3.145	8.25	471	1.887	85.375	167
17	1.258	66.0	194	1.258	93.625	181
18	1.258	88.125	256	0.0	90.0	243
19	0.0	15.125	NaN	0.629	30.125	NaN
20	3.145	31.375	248	1.887	82.0	226
21	1.887	17.25	880	3.774	45.25	634

Tabla 4.29: Estadísticos del VAE distribuido con sesgo 0,5

	Fallos detectados
T^2	17
Q	17
Total	18

Tabla 4.30: Fallos detectados con el VAE distribuido de sesgo 0,5

Como con este sesgo el modelo detecta un fallos menos que con los anteriores (tabla: 4.30), es menos recomendable que los otros, aunque el estadístico Q parezca mejorar (tabla: 4.31) (en realidad, su menor tiempo de detección se debe principalmente a no detectar el fallo 19). El fallo 19 parece oscilar cerca del UCL, de ahí que con este sesgo se pueda detectar un 30 % del mismo y que al mismo tiempo no salte la alarma, ya que no hay 10 mediciones positivas consecutivas.

Falsas T^2 (%)	**Detectadas T^2 (%)**	**Tiempo detección T^2 (mediciones)**
1,408	54,958	119,765
Falsas T^2 sin (%)	**Detectadas T^2 sin (%)**	
1,398	63,771	
Falsas Q (%)	**Detectadas Q (%)**	**Tiempo detección Q (mediciones)**
1,617	75,345	44,716
Falsas Q sin (%)	**Detectadas Q sin (%)**	
1,502	87,472	

Tabla 4.31: Estadísticos de resumen del VAE distribuido de sesgo 0,5

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

Habiendo desarrollado los cuatro métodos de detección de fallos anteriores: PCA, autoencoder, VAE y VAE distribuido, y habiendo comparado su rendimiento para la detección de fallos (ver tablas 5.1 y 5.2), podemos sacar las siguientes conclusiones.

Método	Falsas alarmas T^2 (%)	Alarmas detectadas T^2 (%)
PCA	1,188	66,91
Autoencoder	0,489	59,826
VAE	0,629	60,66
DVAE (sesgo 1)	1,293	68,889
DVAE (sesgo 1.5)	1,398	67,965
DVAE (sesgo 0,5)	1,398	63,771

Tabla 5.1: Comparación de métodos de detección con T^2

Método	Falsas alarmas Q (%)	Alarmas detectadas Q (%)
PCA	3,564	77,319
Autoencoder	0,419	79,938
VAE	0,384	90,41
DVAE (sesgo 1)	1,712	83,556
DVAE (sesgo 1,5)	1,398	83,799
DVAE (sesgo 0,5)	1,502	87,472

Tabla 5.2: Comparación de métodos de detección con Q

Hemos visto que el PCA, si bien no captura las correlaciones no lineales entre las variables, teniendo por tanto un peor rendimiento que los otros métodos, sigue

Método	Fallos detectados T^2	Fallos detectados Q
PCA	17	17
Autoencoder	15	18
VAE	16	18
DVAE (sesgo 1)	17	18
DVAE (sesgo 1,5)	17	18
DVAE (sesgo 0,5)	17	17

Tabla 5.3: Comparación de los fallos detectados

siendo una buena opción si la eficiencia computacional/cronológica es un problema, ya que es un algoritmo muy ligero, fácil de implementar y que no requiere de muchos datos de entrenamiento.

Hemos comprobado cómo las técnicas de aprendizaje automático pueden extraer información útil de datos multivariables de una planta industrial, reduciendo su dimensionalidad y aprendiendo su funcionamiento normal. Una vez aprendido, hemos podido usarlos para la detección de anomalías en línea.

Además, hemos usado el control estadístico de procesos para generar estadísticos multivariables, con los que monitorizar la variabilidad del proceso de forma gráfica.

Hemos comprobado las mejoras que el VAE ofrece frente al autoencoder tradicional, que permiten un entrenamiento mucho mejor, proporcionando una detección de fallos más fiable y rápida.

Además, hemos aplicado un método para dividir automáticamente las variables y obtener información sobre ellas sin requerir conocimiento previo, permitiendo una monitorización de los fallos más localizada, que podría ahorrar tiempo y esfuerzo en una planta industrial.

Por ese motivo, el VAE distribuido es el método que resultaría preferible en una planta real de características similares a la estudiada en este trabajo, con muchas variables y correlaciones complejas entre las mismas, ya que, aunque su detección con Q sea algo más lenta (cosa que se debe exclusivamente al fallo 19), es el método que más fallos ha detectado con ambos estadísticos (tabla: 5.3), y genera datos de fallo mucho más interpretables que los otros métodos.

El VAE no distribuido tiene la ventaja de ser más rápido que el distribuido (tanto para entrenarlo como en su detección de fallos), y, aunque su detección con T^2 es peor que la de su versión distribuida, su detección con Q es algo mejor.

El autoencoder convencional no tiene muchas ventajas sobre los otros métodos, mas allá de generar pocas falsas alarmas. Sería recomendable usar el VAE en su lugar.

5.2. Trabajo futuro

Como trabajo futuro, uno de los objetivos por alcanzar sería llegar a una detección completa de todos los fallos de la planta, ya que hay tres que se le han escapado a las técnicas vistas en este trabajo. Para ello habría que usar redes con estructuras más sofisticadas, que sean capaces de un aprendizaje más complejo.

Otro objetivo importante sería mejorar la monitorización de las variables y su importancia hacia los fallos, buscando determinar de forma más precisa la contribución que cada una de ellas hace a los distintos fallos de la planta.

Bibliografía

[1] Warren T. Ha y Richard K. Morris. *The Book of Statistical Process Control*. Zontec Inc, 2002.

[2] W. A. Shewhart. *Economic Control of Quality of Manufactured Product*. Preface. New York: Van Nostrand, 1931.

[3] International Organization for Standardization (ISO). *Quality management systems – Requirements for quality management systems*. 2015. URL: https://iso.org/standard/61341.html (visitado 07-09-2024).

[4] British Deming Association. *Why SPC?* SPC Press, Inc., 1992.

[5] W. Rojas-Preciado et al. «Control Chart T2Qv for Statistical Control of Multivariate Processes with Qualitative Variables». En: *Mathematics* 11.12 (2023), pág. 2595. DOI: 10.3390/math11122595.

[6] Jesus Solana. *Una oveja negra entre un rebaño de ovejas blancas. Jesus Solana from Madrid, Spain*. Imagen de Flickr. 2023. URL: https://www.flickr.com/photos/65069067@N00/2561252664/ (visitado 07-09-2024).

[7] Ted Dunning y Ellen Friedman. *Practical Machine Learning*. O'Reilly Media, Inc., 2014.

[8] Jeremy. *T-Squared Q residuals and Contributions*. URL: https://wiki.eigenvector.com/index.php?title=T-Squared_Q_residuals_and_Contributions (visitado 07-09-2024).

[9] P. C. Mahalanobis. «On the Generalised Distance in Statistics.» En: *Sankhya: The Indian Journal of Statistics, Series A* 80.Suppl 1 (2018). Reprint of: Mahalanobis, P.C. (1936). Sankhya, 2, 128-149., págs. 1-7. DOI: https://doi.org/10.1007/s13171-019-00164-5.

[10] Charles Gauvin. *Distances and outlier detection*. 2021. URL: https://www.charlesgauvin.ca/post/distances-and-outlier-detection/ (visitado 07-09-2024).

[11] Davide Massidda. *Multivariate Process Control by Principal Component Analysis Using T^2 and Q errors*. 2023. URL: https://towardsdatascience.com/multivariate-process-control-by-principal-component-analysis-using-t%C2%B2-and-q-errors-c94908d14b04 (visitado 07-09-2024).

[12] Ethem Alpaydin. *Introduction to Machine Learning, 3rd Edition*. MIT Press, 2014.

[13] Sanam Malhotra. *Machine Learning for Android Applications*. 2020. URL: https://artificialintelligence.oodles.io/blogs/machine-learning-for-android-applications/ (visitado 07-09-2024).

[14] Stuart J. Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th. Pearson, 2021.

[15] Casey Cheng. *Principal Component Analysis (PCA) Explained Visually with Zero Math*. 2022. URL: https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d#0226 (visitado 07-09-2024).

[16] Marcus Sena. *Principal component analysis made easy: A step-by-step tutorial*. 2023. URL: https://towardsdatascience.com/principal-component-analysis-made-easy-a-step-by-step-tutorial-69015a9592f8 (visitado 07-09-2024).

[17] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] Adrien Debray y Johannes Lootens. *Recovering the ink of Herculaneum using neural networks*. dataroots. 2021. URL: https://dataroots.io/blog/vesuvius-challenge-predicting-ink-in-x-ray-data-of-papyrus (visitado 07-09-2024).

[19] Ruben Tous et al. «Deep Neural Networks for Earthquake Detection and Source Region Estimation in North Central Venezuela». En: *Journal of South American Earth Sciences* 144 (2024), pág. 107386.

[20] François Chollet. *Deep Learning with Python*. 2nd. Manning Publications, 2021.

[21] Aqeel Anwar y Towards Data Science. *Difference between AutoEncoder (AE) and Variational AutoEncoder (VAE)*. 2021. URL: https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2 (visitado 07-09-2024).

[22] Arden Dertat. *Applied Deep Learning - Part 3: Autoencoders*. 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 (visitado 07-09-2024).

[23] Diederik P Kingma y Max Welling. *Auto-Encoding Variational Bayes*. 2022. eprint: 1312.6114. URL: https://arxiv.org/abs/1312.6114 (visitado 07-09-2024).

[24] Danilo Jimenez Rezende, Shakir Mohamed y Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. eprint: 1401.4082. URL: https://arxiv.org/abs/1401.4082 (visitado 07-09-2024).

[25] Jeremy Jordan. *Variational autoencoders*. 2018. URL: https://www.jeremyjordan.me/variational-autoencoders/ (visitado 07-09-2024).

[26] Chen Xu, Shunyi Zhao y Fei Liu. «Distributed plant-wide process monitoring based on PCA with minimal redundancy maximal relevance». En: *Chemometrics and Intelligent Laboratory Systems* 169 (2017), págs. 53-63. ISSN: 0169-7439. DOI: https://doi.org/10.1016/j.chemolab.2017.08.004. URL: https://www.sciencedirect.com/science/article/pii/S0169743916305299.

[27] Z. Ge, M. Zhang y Z. Song. «Nonlinear process monitoring based on linear subspace and Bayesian inference». En: *Journal of Process Control* 20.5 (2010), págs. 676-688.

[28] J.J. Downs y E.F. Vogel. «A plant-wide industrial process control problem». En: *Computers & Chemical Engineering* 17.3 (1993). Industrial challenge problems in process control, págs. 245-255. ISSN: 0098-1354. DOI: https://doi.org/10.1016/0098-1354(93)80018-I. URL: https://www.sciencedirect.com/science/article/pii/009813549380018I.

[29] Chris Aldrich. *Process Fault Diagnosis for Continuous Dynamic Systems Over Multivariate Time Series*. Ed. por Chun-Kit Ngan. Rijeka: IntechOpen, 2019. Cap. 1. DOI: 10.5772/intechopen.85456. URL: https://doi.org/10.5772/intechopen.85456.

[30] Huanhuan Chen, Peter Tiňo y Xin Yao. «Cognitive fault diagnosis in Tennessee Eastman Process using learning in the model space». En: *Computers & Chemical Engineering* 67 (2014), págs. 33-42. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2014.03.015. URL: https://www.sciencedirect.com/science/article/pii/S0098135414000969.

[31] Massachusetts Institute of Technology - Braatz Group. *Braatz Group Data*. URL: http://web.mit.edu/braatzgroup/links.html (visitado 07-09-2024).

[32] Harvard University. *Harvard Dataverse Dataset*. DOI: 10.7910/DVN/6C3JR1. URL: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1 (visitado 07-09-2024).