TRABAJO FIN DE MÁSTER

Máster Oficial en Ciencia de Datos e Ingeniería de Computadores

Detección de Anomalías en Series Temporales basada en técnicas Deep Learning

Biblioteca de algoritmos

Autor
Ignacio Aguilera Martos

Director
Francisco Herrera Triguero

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Facultad de Ciencias

—

Granada, 10 de Septiembre de 2020

Detección de Anomalías en Series Temporales basada en técnicas Deep Learning

Biblioteca de algoritmos

Autor
Ignacio Aguilera Martos

Directores
Francisco Herrera Triguero

Detección de Anomalías en Series Temporales basada en técnicas Deep Learning: Biblioteca de algoritmos

Ignacio Aguilera Martos

Palabras clave:

Resumen

Outlier Detection in Time Series using Deep Learning: Library implementation

Ignacio Aguilera Martos

Keywords:

Abstract

Agradecimientos

Índice general

Capítulo 1

Introducción

Antes de comenzar con el desarrollo en sí del estudio acometido en este trabajo, vamos a contextualizar el mismo y vamos a establecer un marco de trabajo teórico previo a la experimentación, que nos otorgará de rigurosidad para la parte práctica del mismo.

El estudio realizado en este trabajo versa sobre la aplicación de estructuras de Aprendizaje Profundo (Deep Learning) para obtención y detección de anomalías, en concreto, en series temporales. Dentro de este trabajo se van a desarrollar las técnicas conocidas como Autoencoders y Redes Neuronales para predicción de series temporales.

Lo primero que atacaremos en este estudio es la definición de anomalía, para luego pasar a una introducción teórica de Estadística Multivariante y Machine Learning en general. Estas dos secciones nos van a aportar el rigor que necesitamos para adentrarnos teóricamente dentro del Deep Learning y entender los fundamentos de las arquitecturas de redes neuronales que aplicaremos en la práctica.

Tras esto se realizará una descripción de la experimentación realizada, los datos que se emplearán en dicha experimentación y los resultados de la misma.

1.1. Contextualización

Lo primero que debemos de hacer antes de empezar, es establecer el problema u objetivo a resolver de este estudio. Para ello vamos a hacer una breve introducción a los datos (o al problema propuesto que es lo mismo

en este ámbito) y a explicar por qué precisamos de un trabajo arduo y prolongado, es decir, por qué no es un problema trivial.

1.1.1. Definición del problema

El ámbito de trabajo va a ser el de las series temporales, pues el conjunto de datos que nos define el problema es una serie temporal. Esta serie temporal mide la sensórica de una máquina de la empresa ArcelorMittal, que no podemos especificar por motivos de privacidad. En este sentido tenemos 106 variables de tipo numérico con las que vamos a trabajar y 468 días de datos con una granularidad de una medida por segundo. Esto hace que el volumen de datos del que disponemos sea inmenso, haciendo que tenga sentido el uso del Deep Learning por la enorme cantidad de datos de entrenamiento de los que vamos a disponer.

Como hemos comentado los datos son medidas de sensores de una cierta máquina. Esta máquina experimenta errores graves de vez en cuando, que hacen que se deba detener completamente para labores de mantenimiento. Nuestro objetivo es ser capaces de detectar estas labores de mantenimiento mediante técnicas de detección de anomalías. El principio subyacente es sencillo: esperamos un comportamiento normal de la máquina en la mayoría del tiempo salvo cuando haya necesidad de un mantenimiento, momento en el cual la sensórica arrojará medidas alteradas que nos den pie a pensar en un posible fallo.

Este tipo de problemas son conocidos como mantenimiento predictivo, pues lo que pretenden precisamente es anticipar la necesidad de dichas labores.

Con esto dicho nuestro objetivo será tomar los datos de entrada (la sensórica) para nuestros modelos Deep Learning y, de alguna manera, saber diferenciar lo que son datos normales y datos anómalos.

1.2. Contenido básico y fuentes

El trabajo contiene una primera sección en la que se incluye una introducción de Aprendizaje Automático orientado específicamente a nuestro problema. Para ello primero se hace una contextualización del concepto de aprendizaje así como los principios inductivos que guían el mismo hacia un buen resultado como por ejemplo el ERM o minimización del error empírico. Se aportan también algunas reflexiones y conceptos en cuanto a la aproxima-

ción de funciones, que no es más que el objetivo del aprendizaje automático.

Todos estos conocimientos están basados en la teoría estadística de Vapnik y Chervonenkis que es brevemente repasada y en la que se dan cotas sobre el aprendizaje y su rendimiento. Esta introducción ha sido escrita basándose en tres libros: Learning from Data de Yaser Abu-Mostafa [1], Learning from Data de Cherkassky y Mulier [2] y Outlier Ensembles de Aggarwal y Sathe [3].

Este marco nos dirige hacia la primera definición del concepto de anomalía que está basada en distancias y rangos intercuartil que se describen en el libro Outlier Analysis [**?**].

Para dar una definición alternativa y una buena introducción para los modelos debemos hacer una breve introducción estadística. En esta introducción se define un vector aleatorio así como su función de densidad, su función característica y su función de distribución. Se refieren los conceptos de independencia y probabilidad y esperanza condicionada. Por último y aprovechando este contexto se enuncian y demuestran algunas desigualdades y fórmulas famosas. Este contenido viene dado por los apuntes de la asignatura Estadística Multivariante del grado en Matemáticas, los apuntes de la asignatura Procesos Estocásticos del grado en Matemáticas y el libro Probability Theory de M. Loève [**?**].

Tras esto puede ser introducido el concepto probabilístico y basado en densidad de una anomalía. Este concepto viene apoyado en el paper [**?**] que describe el algoritmo HICS.

Con los dos conceptos de probabilidad y el marco teórico ya planteado se introducen los modelos implementados y el concepto de algoritmos de ensamblaje. Estos conceptos sobre los algoritmos vienen de los libros Outlier Analysis [**?**] y Outlier Ensembles [3]. Se aporta en esta sección la explicación teórica de cada modelo así como la implementación desarrollada por mí mismo en Python. Los artículos en los que se basa cada algoritmo son [**?**], [3], [**?**] y [**?**].

Finalmente se analiza el comportamiento de todos los modelos en la sección de resultados frente a los algoritmos considerados como clásicos. Se aportan conclusiones tras todo el trabajo y, al haber margen de mejora, se aportan algunas ideas que podrían aplicarse en un futuro para desarrollar un modelo propio.

1.3. Objetivos

Por todo lo descrito anteriormente el trabajo tiene los siguientes objetivos claros:

- Desarrollar un marco teórico sobre el Machine Learning.
- Desarrollar un marco teórico sobre el Deep Learning.
- Estudiar el estado del arte de los algoritmos de detección de anomalías que emplean Deep Learning.
- Estudiar la teoría estadística que rodea el Machine Learning y el Deep Learning.
- Entender los fundamentos teóricos y el funcionamiento de los modelos implementados.
- Desarrollar una implementación de los modelos.
- Obtener una comparativa entre los modelos clásicos y los Deep Learning.

Todos estos objetivos han sido alcanzados en el desarrollo de este estudio, obteniendo además algunas ideas nuevas que pudieran ser la base de un modelo propio.

Parte I

Machine Learning, Deep Learning y el concepto de anomalía

Capítulo 2

Concepto de Anomalía y Series Temporales

2.1. Concepto de Anomalía

Debemos de tener en cuenta que el concepto de anomalía no es algo fácil de definir. Tanto es así que, por ligeros cambios o matices en la definición, podemos estar cayendo en un concepto completamente distinto.

Antes de comenzar debemos aclarar el objetivo que vamos persiguiendo, es decir, el concepto de anomalía que nos va a interesar. Podemos encontrar muchas definiciones de anomalías, pero en nuestro caso nos vamos a centrar en la dada por Carreño, Inza y Lozano en [4].

Según estos autores podemos definir el contexto de la detección de anomalías en 4 subtipos: eventos raros, anomalías, novedades y outliers.

En primer lugar, tenemos los eventos raros. Tenemos un problema en el que hay un tipo de datos que aparecen con muy poca frecuencia en el contexto de las series temporales y queremos detectar dicho tipo de eventos. Estamos en una perspectiva supervisada, por lo que esto no es más que un problema de clasificación altamente desbalanceado en el contexto de las series temporales. En este escenario el problema se resuelve aplicando distintas técnicas que favorezcan que los clasificadores aprendan bien esta clase rara y se detecte. Claramente no es nuestro caso pues no disponemos de etiquetas claras y no es un problema de clasificación.

En segundo lugar tenemos las anomalías. Según Carreño, las anomalías están enmarcadas en conjuntos de datos estáticos. Este simple hecho ya saca

el subtipo de nuestro marco de trabajo, pero aún así es bueno ver su definición. Las anomalías son, para Carreño et al., un problema de clasificación altamente desbalanceado en el contexto de datos estáticos. Esto es análogo al caso anterior, salvando el paso de datos estáticos a dinámicos. Claramente no es nuestro caso pues el problema no es de clasificación ni tenemos etiquetas claras ni son datos estáticos de los que disponemos.

En tercer lugar tenemos las novedades. Este apartado puede ser aplicado tanto a datos estáticos como a datos dinámicos. En los dos casos anteriores tenemos problemas de clasificación, pero disponemos de las etiquetas y de ejemplos de todas las clases para la fase de entrenamiento. En las novedades tenemos datos normales de una sola clase en la fase de entrenamiento, por lo que en el momento de entrenar tendremos que definir las fronteras de la única clase que tenemos. El objetivo en este problema es detectar la novedad, es decir, los nuevos ejemplos que no cuadran dentro de la frontera de decisión de la única clase que tenemos en la fase de entrenamiento en el problema. De nuevo esto no es nuestro caso, porque no tenemos etiquetas de los datos normales y por tanto no podemos definir claramente ese marco de trabajo "one class".

Por último tenemos los outliers. Este término no es de fácil traducción al español, por lo que es preferible dejar el original en inglés. Este punto engloba la clasificación no supervisada, es decir, tenemos ciertas nociones del conjunto de datos pero ninguna etiqueta precisa y aun así queremos saber qué datos son normales y cuáles anómalos basándonos en alguna técnica que no emplee más que los propios datos sin etiquetar. Este es nuestro marco de trabajo, pues no disponemos de etiquetas claras ni "ground truth", oráculo o verdad absoluta a la que recurrir para aprender de ella. Tenemos que elaborar un sistema capaz de detectar los mantenimientos de nuestra máquina sin poder aprender a priori lo que es normal y lo que es anómalo.

Dentro de este esquema de posibilidades ya hemos localizado la que más se acerca al objetivo que queremos cumplir. Como hemos podido ver, es la única opción completamente no supervisada que Carreño contempla en el artículo, lo que nos deja con el escenario más complejo de todos.

Pensemos un momento todas las posibles definiciones de anomalías que tenemos mediante varios ejemplos.

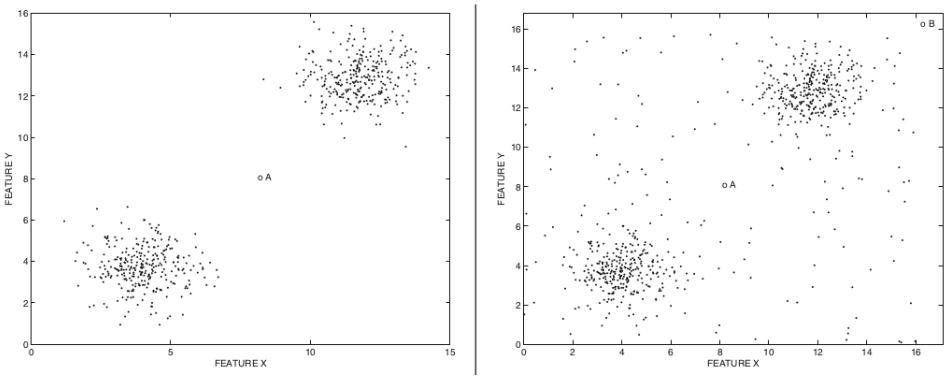

Figura 2.1: Ejemplo sin ruido y con ruido de una anomalía. [3, p21]

Como podemos ver, el caso de la izquierda es relativamente fácil de detectar, por ejemplo con un algoritmo de clústering. El ejemplo de la derecha es muchísimo más complejo. En el ejemplo tenemos identificado el mismo punto como anomalía, pero ahora está rodeado de ruido que hace muy complicado detectarlo. Además surge la pregunta de cuándo tenemos ruido y cuándo son puntos anómalos ya que la diferencia en algunos casos es inapreciable.

Para todas estas cuestiones no hay una respuesta que siempre sea la adecuada, pues dependen del contexto en el que estemos y de lo que queramos detectar y hacer con las anomalías. Los algoritmos de detección de anomalías, al no ser fácil la tarea de clasificación por ser no supervisado, nos suelen arrojar un número que puntúa cada instancia. Cuanto mayor sea el número asignado a una instancia mayor es su grado de anomalía. Este sistema nos permite asignar algún tipo de regla que detecte los puntos más anómalos y deje fuera los puntos de los que no estemos seguros. Por tanto, un algoritmo de detección de anomalías por si solo para este problema no es de utilidad. Hay que acompañarlo de un sistema que decida sobre los scores qué puntos son anómalos y cuales no, además de que en nuestro problema no tenemos datos estáticos, si no series temporales, por lo que debemos también dotar de esa temporalidad a las anomalías.

Todos estos aspectos los discutiremos más a fondo cuando nos acerquemos a la sección de experimentación, donde podremos ver mejor la forma final de detectar anomalías y mantenimientos en nuestro caso.

2.1.1. Concepto clásico de anomalía

Vamos a dar una breve definición del concepto clásico de anomalía basado en distancias. Este apartado está basado en el libro Outlier Analysis [3].

La definición clásica comienza en espacios de una única dimensión para poder entender bien el concepto antes de generalizarlo. Si tenemos un espacio de valores reales, solemos aplicar lo que se conoce como el criterio de Tukey. Este critero nos dice que, si el valor de un punto se sale del intervalo $[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$, donde Q_i representan los cuartiles y k es un número arbitrario (normalmente 1.5, 3 o 5), entonces dicho punto es una anomalía.

Este criterio es una primera aproximación simple y únicamente interesante en una dimensión. Podemos intentar extender este mismo criterio haciendo esta comprobación sobre todas las dimensiones o características de nuestros datos, pero esta es una extensión demasiado simple. Esto es fácilmente comprobable con un ejemplo en dos dimensiones:

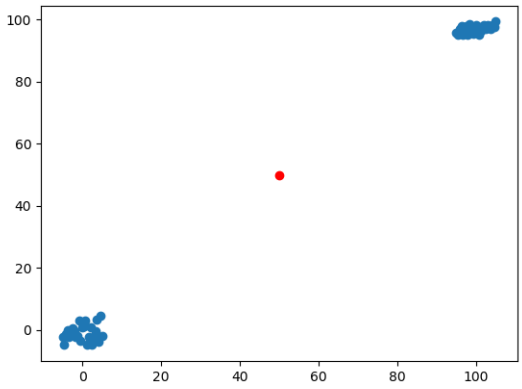

Figura 2.2: Ejemplo de anomalía entre clusters.

Tenemos un punto entre dos cluster en dos dimensiones. Este comportamiento es claramente anómalo pues no entra dentro de ninguno de los clusters ni está cerca aunque esté a medio camino entre los dos, por lo que tenemos una anomalía escondida entre los dos grupos que consideramos normales. Si aplicamos el criterio de Tukey en las dos dimensiones el punto anómalo cae dentro del intervalo normal con mucha holgura. Es por esto que el criterio de Tukey extendiéndolo a todas las dimensiones no es un criterio útil.

Podemos dar un criterio mejor para más dimensiones, por ejemplo podemos agrupar los datos por clústeres y calcular la mayor distancia dentro de cada uno de ellos. Si tenemos un punto que se distancie más de k veces dicha distancia máxima de todos los clústeres entonces estamos ante una anomalía, pues se distancia de todos los comportamientos normales.

Recordemos que esto no es más que una serie de criterios, que pueden no ser de utilidad para todos los problemas. Por ello debemos de estudiar siempre las peculiaridades de nuestros problemas antes de decidir el algoritmo o criterio que nos puede ayudar más en la detección de anomalías.

2.2. Series Temporales

Ya hemos comentado que vamos a trabajar con series temporales, por lo que antes de empezar cabe definir formalmente lo que consideramos una serie temporal.

Una serie temporal podemos definirla como un par $(t, x) \in \mathbb{R} \times \mathbb{R}$ donde t es un sello temporal, es decir, una cuantificación numérica para el tiempo desde un punto de referencia y x es un valor numérico asociado al valor temporal. De esta forma lo que tenemos son valores numéricos para cada valor temporal.

Si extendemos este concepto podemos definirlo como $(t, x) \in \mathbb{R} \times \mathbb{R}^n$ donde n es la dimensionalidad de la serie temporal, es decir, ahora no toma valores reales si no vectores de valores reales.

Las series temporales se pueden dividir en varias componentes. Esta división tiene la intención de poder entender mejor el comportamiento de la serie, tanto para su estudio como el posterior modelado y predicción si nos interesase.

Las componentes que podemos extraer de una serie temporal son:

- Tendencia: nos indica la componente que se mantiene estable a lo largo de toda la serie temporal, por ejemplo podemos tener tendencias crecientes, decrecientes o no tener tendencia en nuestra serie.
- Componente estacional: es la componente cíclica que se repite con periodos menores a un año, por ejemplo puede ser por estaciones, meses, semanas u otra fracción de tiempo significativa para el problema.
- Componente cíclica: es la componente que recoge los fenómenos periódicos de frecuencia mayor a un año, normalmente de periodo irregular influida por movimientos a menudo dependientes de la tendencia.
- Componente residual: es la componente que depende del ambiente del sistema. No tiene ningún tipo de regularidad y podemos decir que depende de las anomalías que se presenten en la serie de forma más común o frecuente.

- Componente accidental: esta componente recoge las variaciones que se producen por fenómenos muy raros y aislados.

Podemos poner un ejemplo para ilustrar las componentes de una serie. Por ejemplo si tenemos una serie temporal con las temperatura de la tierra actualmente podemos apreciar una tendencia creciente por el cambio climático, por lo que la componente de la tendencia sería creciente. Tenemos una componente estacional que podemos apreciar por las estaciones del año, momentos en los cuales las temperaturas bajan y suben siempre de la misma forma o muy similar. La componente cíclica puede ser por ejemplo las edades de la Tierra, momentos en los cuales las temperaturas bajan y suben dependiendo de las glaciaciones y la flora y fauna de la Tierra. En la componente residual podemos tener por ejemplo fenómenos como tormentas, inundaciones o fenómenos producidos por el hombre. En la componente accidental tendríamos fenómenos mucho más súbitos y repentinos, por ejemplo podríamos poner en esta componente la caída de un meteorito, fenómeno que raramente puede producirse más de una vez o dos.

Visto esto, ya tenemos una idea de lo que vamos a ir buscando y el tipo de datos con los que vamos a tratar. En nuestra serie temporal nosotros nos vamos a preocupar por la componente residual y la accidental, ya que son las componentes en las que se esconden los fenómenos que no son modelables y por tanto escapan al comportamiento normal de las variables.

Capítulo 3

Machine Learning

En este capítulo vamos a hacer un repaso sobre los conceptos asociados al Machine Learning, el aprendizaje y la teoría matemática que involucra. Estas herramientas y conceptos los utilizaremos posteriormente para resolver el problema de detección de anomalías. El contenido de esta sección se ha basado en varios libros y artículos, pero principalmente el contenido ha sido extraído de los libros: Learning from Data de Yaser Abu-Mostafa [1], Learning from Data de Cherkassky y Mulier [2] y Outlier Ensembles de Aggarwal y Sathe [3].

3.1. Contextualización del aprendizaje

Para comenzar tenemos que empezar definiendo en que consiste el proceso de aprender sobre unos datos. Supongamos que tenemos un problema en el que tenemos una entrada y una salida, por ejemplo una entrada válida podría ser un vector $x \in \mathbb{R}^d$ y una salida un valor real o un número natural. El problema de aprendizaje intenta estimar una estructura de tipo entrada-salida como la descrita usando únicamente un número finito de observaciones.

Podemos definirlo de forma más general empleando tres conceptos:

- Generador: El generador se encarga de obtener las entradas $x \in \mathbb{R}^d$ mediante una distribución de probabilidad $p(x)$ desconocida y fijada de antemano.
- Sistema: El sistema es el que produce la salida "y" (correcta) para cada entrada $x \in \mathbb{R}^d$ mediante la distribución de probabilidad $p(x|y)$

desconocida y fijada de antemano.

- Máquina de aprendizaje: esta es la que va a obtener información de las entradas y salidas conocidas para intentar predecir la salida correcta para una entrada nueva que se nos de. De forma abstracta esta máquina lo que hace es tomar una serie de funciones de un conjunto general de forma que para una entrada dada x la función $f(x, \omega)$ con $\omega \in \Omega$ nos de la salida que corresponde para x donde ω es una forma de indexar las funciones tomadas para generalizar la salida del conjunto más general de funciones que hemos indicado.

El único cabo que hemos dejado sin atar en las definiciones que acabamos de ver es el conjunto de funciones del cual tomaremos algunas para adaptar la máquina de aprendizaje a los datos recibidos. Este conjunto de funciones, que notaremos como $\mathcal{H}$, es de momento la única forma que tenemos de aplicar un conocimiento a priori en la máquina de aprendizaje.

Para finalizar esta breve introducción y poder continuar profundizando vamos a exponer algunos ejemplos de clases de funciones para que podamos visualizar el contexto.

- Funciones lineales: En este caso la clase de funciones $\mathcal{H}$ está formada por funciones de la forma $h(x) = w_0 + \sum_{i=1}^{d} x_i w_i$ donde $w \in \mathbb{R}^{d+1}$. Este es el modelo de funciones más clásico.

- Funciones trigonométricas: Un ejemplo de una clase de funciones trigonométricas podría ser $f_m(x, v_m, w_m) = \sum_{j=1}^{m-1}(v_j \sin(jx) + w_j \cos(jx)) + w_0$ donde en este caso la entrada es un único valor real. Este tipo de clases de funciones serán útiles en problemas de regresión que luego explicaremos con algo más de detalle.

3.1.1. Objetivo del aprendizaje

Cuando hablamos de aprendizaje nos referimos a que queremos obtener una cierta información a partir de los datos de que disponemos. Como ya se ha mencionado, intentamos obtener una función de una familia de funciones que aproxime o modele de buena manera la salida del sistema. Por tanto, ese es nuestro objetivo: obtener una función de la familia de funciones que minimice el error.

El problema que enfrentamos es que sólo disponemos de un número finito, por ejemplo n, de observaciones de datos y su correspondiente salida. Esto

nos va a hacer que no podamos tener una garantía de optimalidad a no ser que hagamos tender n a infinito.

Sin embargo si que podemos cuantificar cómo de buena es una aproximación con respecto a otra mediante la función pérdida o error que denotaremos como $L(y, f(x, \omega))$. Esta función nos va a medir la diferencia entre la salida real del sistema y la salida dada por la función f para la entrada x siendo siempre $L(y, f(x, \omega)) \geq 0$.

Recordemos además que el Generador obtiene datos mediante una distribución desconocida pero fijada de antemano y que son independientes e idénticamente distribuidos con respecto a la distribución conjunta, es decir:

$$p(x, y) = p(x)p(y|x)$$

Una vez definido todo esto podemos obtener el valor esperado de pérdida o error mediante el funcional

$$R(\omega) = \int L(y, f(x, \omega))p(x, y)dxdy$$

Ahora podemos concretar un poco más lo que entendemos como objetivo del aprendizaje. El objetivo será encontrar una función $f \in \mathcal{H}$ que nos minimice el valor del funcional $R(\omega)$. Pero recordemos que $p(x, y)$ es desconocida para nosotros, por lo que no podemos saber cómo se distribuyen los datos y por tanto el valor del funcional no es calculable para nosotros y por tanto la solución puramente de cálculo no es accesible.

Por tanto, la única forma realmente potente y útil de encontrar una buena aproximación será incorporar el conocimiento a priori que tenemos del sistema. En la sección anterior hemos visto que una forma de incorporar dicho conocimiento es mediante la selección de la clase de funciones, pero además será muy relevante el hecho de cómo los datos son empleados en el proceso de aprendizaje. En este apartado de decisión tendremos que resolver primero la codificación de los datos, el algoritmo empleado y el uso de técnicas como la regularización que veremos después para incorporar nuestro conocimiento en el camino que nos lleve a la solución.

3.1.2. Clases de aprendizaje

El problema de aprendizaje puede ser subdividido a su vez en cuatro clases distintas y que se suelen abordar de forma independiente. Estos tipos

de problemas de aprendizaje son:

- Clasificación: El problema de clasificación consiste en identificar y separar instancias de datos según su clase. Por ejemplo podemos dividir a la población mundial en dos clases: sanos y enfermos. Un problema de clasificación podría ser saber identificar estas clases para un conjunto de personas. Los problemas de clasificación más sencillos son aquellos en los que se usan dos únicas clases aunque se puede generalizar la definición del problema a k-clases.

- Regresión: El problema de regresión consiste en estimar una función $f : \mathbb{R}^n \to \mathbb{R}$ a partir de una serie de muestras previas con los valores de f. Un problema de regresión podría ser determinar la función que, dados los datos de altura y dimensiones corporales sea capaz de darnos el peso aproximado de la persona.

- Estimación de la función de densidad: en este caso no nos interesa la salida que proporciona el sistema, ya sea el valor de una clase o una función real como en el caso de la regresión. En este caso el objetivo del aprendizaje es conseguir la función de densidad $f(x, \omega)$, con $\omega \in \Omega$ los parámetros necesarios de la función de densidad, con la que se distribuyen los datos de entrada del sistema.

- Agrupamiento y cuantificación vectorial: El problema de cuantificación vectorial consiste en intentar explicar la distribución de los vectores de entrada mediante puntos clave llamados centroides. De esta forma se podría reducir la complejidad de los datos expresándolos en función de un sistema de generadores menor. El problema de agrupamiento tiene también relación por utilizar la idea de centroide, pero el objetivo es completamente distinto. El objetivo del problema de agrupamiento es intentar conseguir agrupar los datos en clústeres, es decir, regiones del espacio en las que se concentran un conjunto de datos. De esta forma intentamos agrupar los datos que mantienen una relación entre sí. Un ejemplo de un problema de cuantificación vectorial podría ser un problema de reducción de dimensionalidad y un ejemplo de problema de agrupamiento podría ser identificar instancias de datos con características comunes.

3.2. Principios y adaptación del aprendizaje

Según Vapnik [5] la predicción mediante el aprendizaje se puede dividir en dos fases:

1. Aprendizaje o estimación a partir de una muestra.
2. Predicción a partir de las estimaciones obtenidas.

Estas dos fases se corresponden con los dos tipos de inferencia clásica que conocemos, esto es, inducción y deducción. Traído a este caso el proceso de inducción es aquel que a partir de los datos de aprendizaje o los datos de la muestra que tenemos con la salida que corresponde podemos estimar un modelo. Es decir, estamos sacando el conocimiento de los datos para generar el modelo. El proceso de deducción es aquel que, una vez obtenido el modelo estimado (la generalización) obtenemos una predicción de la salida sobre un conjunto de datos.

Por contra, Vapnik propone un paso que resuelve estas dos fases directamente y que él denomina transducción. Este paso consiste en, dados los datos de entrenamiento obtenemos directamente los valores de salida sin tener que hacer la generalización a un modelo. De esta forma, según Vapnik, podríamos reducir el error que cometemos en la predicción. Este razonamiento tiene sentido, pues estamos omitiendo el paso más complejo del proceso de inducción-deducción.

En resumen esta idea se puede resumir en la siguiente figura:

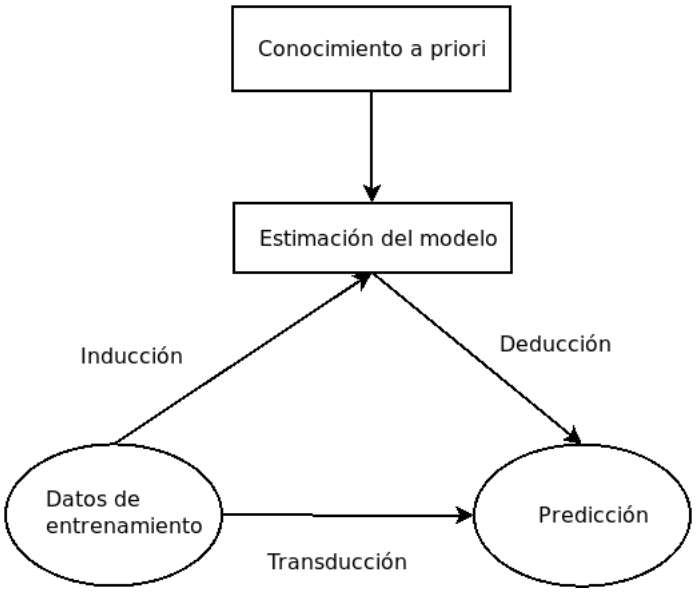

Figura 3.1: Tipos de inferencia y transducción [2, p. 41]

Podemos ver que el conocimiento a priori que tenemos del problema se manifiesta una vez se crea el modelo general, de forma que se emplearía en

el paso de la inducción. Ya hemos hablado previamente del conocimiento a priori y cómo incorporarlo al modelo, pero por concretar un poco más podemos añadirlo básicamente de dos formas:

- Escogiendo un conjunto de funciones para aproximar la salida del sistema
- Añadiendo restricciones o penalizaciones adicionales a dicho conjunto de funciones.

En resumen, para poder crear la generalización del modelo de forma única necesitamos:

1. Un conjunto de funciones para aproximar la salida.
2. Conocimiento a priori.
3. Un principio inductivo, que no es más que una indicación de cómo emplear los datos para llegar a la generalización del modelo.
4. Un método de aprendizaje, es decir, una implementación del principio inductivo.

En secciones posteriores revisaremos algunos de los principios inductivos más usados pero es importante reseñar la diferencia entre principio inductivo y método de aprendizaje. Para un mismo principio inductivo podemos tener varios métodos de aprendizaje, pues podemos escoger diferentes formas de llevarlo a la práctica. Por ejemplo, uno de los principios inductivos más empleados es el ERM o Empirical Risk Minimization, es decir, minimización del error empírico. Podríamos pensar en diferentes formas de utilizar este principio, por ejemplo sólo avanzamos en la creación del modelo si a cada paso que demos minimizamos el error, o por ejemplo vamos avanzando varios modelos a la vez hasta obtener un número de modelos finales de entre los cuales escogeremos aquel que mejor minimice dicho error.

3.2.1. Principios inductivos

Una vez introducido el concepto como hemos hecho en la sección anterior vamos a hacer un breve repaso de los principios más usados y en qué consiste cada uno de ellos.

Penalización o Regularización

Imaginemos que tenemos una clase de funciones muy flexible, esto es con un gran número de parámetros libres $f(x, \omega)$ con $\omega \in \Omega$. Vamos a partir de la base del ERM, es decir, minimizar el error empírico. La penalización lo que va a hacer es añadir un factor a la función a minimizar:

$$R_{pen}(\omega) = R_{emp}(\omega) + \lambda\phi[f(x, \omega)]$$

Donde $R_{emp}(\omega)$ es el error empírico con los parámetros ω y $\phi[f(x, \omega)]$ es un funcional no negativo asociado a cada estimación $f(x, \omega)$. El parámetro $\lambda > 0$ es un escalar que controla el peso de la penalización.

El funcional $\phi[f(x, \omega)]$ puede medir lo que creamos conveniente que debemos añadir, es decir, aquí podemos añadir a la minimización algún tipo de medida que nos diga cómo de bien funciona el ajuste de los datos y cómo de bien funciona la información a priori que hemos incluido en el modelo. Pensemos por ejemplo que λ fuera un parámetro con un valor muy alto. En este caso la penalización por un mal ajuste de los datos no sería de gran importancia pues lo más conveniente sería minimizar el valor del funcional para no obtener una gran penalización. De esta forma podemos ajustar y dar un poco más de información al error empírico. Por ejemplo, en función del problema, es posible medir la complejidad de la solución mediante el funcional ϕ y de esta forma no sólo vamos a obtener una función que ajuste bien los datos, si no que también mantenga una cierta simplicidad para evitar por ejemplo el sobreajuste.

Reglas de parada anticipada

Pensemos en un método que vaya aprendiendo de los datos de forma iterativa intentando a cada iteración reducir el error cometido, por ejemplo el ERM. Los métodos o reglas de parada anticipada pueden verse como penalizaciones sobre el algoritmo conforme se va ejecutando. Las reglas de parada anticipada, como su nombre indica lo que prevén es la parada del algoritmo antes de obtener su objetivo teórico. Por ejemplo un algoritmo intenta que el error sea menor que 10^{-6} pero para reducirlo desde 10^{-4} hasta 10^{-5} está consumiendo millones de iteraciones. Si queremos que el tiempo de cómputo penalice lo que podemos hacer es fijar por ejemplo un número máximo de iteraciones que detenga el método aunque no se haya alcanzado esa barrera de error que se preveía.

Minimización del riesgo estructural o SRM

Para entender esta filosofía nos ponemos en la situación de que ya sabemos la clase de funciones con la que vamos a aproximar la salida del sistema, por ejemplo hemos escogido la clase de funciones polinómicas. Bajo esta clase de funciones podemos ordenar las funciones por complejidad, entendiendo por complejidad el número de parámetros de la función. Por ejemplo los polinomios de grado m son de menor complejidad que los de grado $m+1$. De esta forma podemos pensar en una estructura de la clase de funciones de la forma:

$$S_0 \subset S_1 \subset S_2 \subset \cdots$$

Este parámetro de complejidad también puede ser un principio a minimizar para intentar conseguir una solución adecuada pero también simple. La generalización de la medida de complejidad para las clases de funciones es la conocida como dimensión VC o dimensión de Vapnik-Chervonenkis.

Inferencia Bayesiana

Este principio inductivo se utiliza en el problema de estimación de la función de densidad. El principio es utilizar la conocida fórmula de Bayes para hacer una estimación de la función de densidad empleando el conocimiento a priori que disponemos del problema. La forma en la que se emplea esta fórmula es de la siguiente:

$$P[modelo|datos] = \frac{P[datos|modelo] \cdot P[modelo]}{P[datos]}$$

, donde $P[modelo]$ es la probabilidad a priori, $P[datos]$ es la probabilidad de los datos de entrenamiento y $P[datos|modelo]$ es la probabilidad de que los datos estén generados por el modelo.

Descripción de mínima longitud

La idea de este principio es la minimización de la longitud que se necesita emplear para describir un modelo y la correspondiente salida. Llamamos l a la longitud total:

$$l = L(modelo) + L(datos|modelo).$$

Esta medida puede ser vista como una medida de complejidad conjunta de todo el modelo.

3.3. Regularización

Por la importancia de este principio inductivo vamos a desarrollarlo un poco más, junto con el concepto de penalización, la selección de los modelos y la relación entre sesgo y varianza. Este último es un concepto muy relevante en cuanto al aprendizaje y que en nuestro caso, al no poseer la clasificación real tendremos que tenerlo en cuenta.

3.3.1. Problema de la alta dimensionalidad

Sabemos que cuando estamos ante un problema de aprendizaje nuestro objetivo es conseguir estimar una función con un número finito de instancias de una muestra ya con la salida. Al tener un número finito de elementos en la muestra ya sabemos que no podemos garantizar que la respuesta sea la óptima o correcta, pero además debemos pensar que a mayor regularidad del conjunto de funciones empleado debemos tener una densidad suficiente de puntos para compensar dicha regularidad. Este problema es conocido como la maldición de la dimensionalidad (curse of dimensionality). El problema es que cuanto mayor sea la dimensionalidad considerada más difícil es poder tener esa alta densidad de datos que se requieren para funciones muy regulares.

Este problema que conlleva la alta dimensionalidad proviene de la geometría de los espacio con alta dimensionalidad. A medida que incrementamos la dimensionalidad el espacio se ve cada vez con más aristas o picos. Podemos pensar en un cubo para el espacio tridimensional y a medida que aumentamos la dimensión incorporamos más aristas y vértices. Podemos resumir en 4 propiedades de los espacio con alta dimensionalidad que causan este problema:

1. La densidad disminuye exponencialmente al aumentar el número de dimensiones. Supongamos que tenemos una muestra de n puntos en $\mathbb{R}$. Para poder tener la misma densidad en un espacio d-dimensional $\mathbb{R}^d$ necesitamos n^d puntos.

2. Cuanto mayor dimensionalidad tenga el conjunto de datos mayor lado se necesita para que un hipercubo contenga el mismo porcentaje del conjunto que con una menor dimensionalidad. Imaginemos que tenemos un conjunto d-dimensional en el que tenemos la muestra dentro de un hipercubo unidad. Si quisiéramos abarcar un porcentaje $p \in [0, 1]$ necesitaríamos un cubo de lado $e_d(p) = p^{\frac{1}{d}}$. Como se puede observar a mayor dimensionalidad y p constante el lado es cada vez mayor. Esta idea es fácilmente entendible si observamos la siguiente figura:

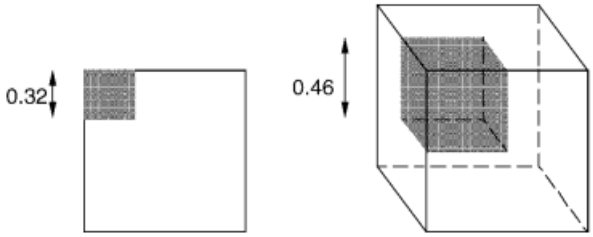

Figura 3.2: Para 2 dimensiones necesitamos menor lado que para 3 dimensiones. [2, p. 64]

3. Casi todo punto está más cerca de un borde que de otro punto. Pensemos en un conjunto de datos con n puntos distribuidos de forma uniforme en una bola d-dimensional de radio unidad. Para este conjunto de datos, según Hastie [**?**], la distancia media entre el centro de la distribución y los puntos más cercanos a dicho centro se mide bajo la fórmula:

$$D(d, n) = (1 - \frac{1}{2}^{1/n})^{1/d}$$

Si en esta fórmula tomamos por ejemplo $n = 200$ y $d = 10$ el resultado es $D(10, 200) \approx 0.57$. Esto significa que los puntos más cercanos al centro de la distribución están más cerca de los bordes que del centro.

4. Casi todo punto es una anomalía sobre su propia proyección. Si pensamos de nuevo en la idea de los vértices y aristas en espacio de alta dimensionalidad y pensamos en que, según el punto anterior, cada vez que aumenta la dimensionalidad los puntos están más cerca de los bordes entonces no es extraño pensar que los puntos a medida que aumenta la dimensionalidad están más distantes del resto de puntos. Esto intuitivamente (ya que aún no hemos visto la definición formal de anomalía) nos guía a pensar que vistos los puntos en sus propios entornos éstos serán anomalías comparados con el resto.

Figura 3.3: Forma conceptual de un espacio de alta dimensionalidad.[2, p. 64]

> Conceptualmente podemos imaginarlo con esta forma de picos, con lo que si tenemos los datos apiñados en dichos picos o extremos el resto de datos que estén en picos diferentes distan tanto del que estamos considerando que no podemos afirmar que tengan ninguna relación entre sí.

Estos puntos hemos de recordar que van referidos al conjunto de datos y no a las funciones que estamos considerando para representar la salida del sistema. Si estamos considerando la complejidad de las funciones la dimensionalidad no es una buena medida. Sabemos de la existencia de teoremas de aproximación de funciones como por ejemplo el Teorema de Superposición de Kolmogorov-Arnold.

Teorema 3.1 (Teorema de Superposición de Kolmogorov-Arnold) *Sea f una función continua de varias variables $f : X_1 \times ... \times X_n \to \mathbb{R}$, entonces existen funciones $\Phi_q : \mathbb{R} \to \mathbb{R}$ y $\phi_{q,p} : X_p \to [0, 1]$ tales que f se puede expresar como:*

$$f(x) = f(x_1, ..., x_n) = \sum_{q=0}^{2n} \Phi_q(\sum_{p=1}^{n} \phi_{q,p}(x_p))$$

Este Teorema argumenta perfectamente que la complejidad que le damos a los datos por tener una alta dimensionalidad no es transferible a las funciones pues podemos expresar funciones de varias variables como combinación de funciones de una sola variables. En otras palabras no podemos argumentar que la complejidad de funciones univariantes sea mayor o menor que la de funciones multivariantes.

3.3.2. Aproximación de funciones

Como ya hemos dicho en la introducción queremos aproximar una función salida del sistema dentro de una familia de funciones. Este campo no es nuevo, tenemos como herramientas una serie de Teoremas relacionados con la aproximación de funciones como el Teorema de Kolmogorov enunciado anteriormente o el Teorema de aproximación de Weierstrass.

La versión más simple del Teorema de Weierstrass es la de funciones reales definidas en intervalos cerrados, veamos un repaso de estos Teoremas para hacer un esquema de la aproximación de funciones.

Teorema 3.2 (Teorema de aproximación de Weierstrass) *Supongamos que $f : [a, b] \to \mathbb{R}$ es una función continua. Entonces $\forall \epsilon > 0$, $\exists p$ un polinomio tal que $\forall x \in [a, b]$ tenemos que $|f(x) - p(x)| < \epsilon$.*

En otras palabras, podemos aproximar las funciones continuas reales definidas en un intervalo cerrado con el error que queramos en un punto mediante polinomios. Además tenemos versiones más generales aún como el Teorema de Stone-Weierstrass para funciones reales, para espacios localmente compactos y para el espacio de los complejos.

Estas aproximaciones son más sencillas en términos de la complejidad de la clase de funciones, pero tenemos aproximaciones muy famosas, como por ejemplo la serie de Fourier.

Definición 3.1 (Serie de Fourier) *Si tenemos una función $f : \mathbb{R} \to \mathbb{R}$ integrable en el intervalo $[t_0 - \frac{T}{2}, t_0 + \frac{T}{2}]$ entonces se puede obtener el desarrollo en serie de Fourier de f en dicho intervalo. Si f es periódica en toda la recta real la aproximación será válida en todos los valores en los que esté definida.*

$$f(t) \approx \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(\frac{2n\pi}{T} t) + b_n \sin(\frac{2n\pi}{T} t)]$$

Donde a_0, a_n y b_n son los coeficientes de la serie de Fourier que tienen la forma:

$$a_0 = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) dt$$

$$a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)cos(\frac{2n\pi}{T}t)dt$$

$$b_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\sin(\frac{2n\pi}{T}t)dt$$

Como podemos ver hemos introducido dos conocidas formas de aproximar funciones, una con funciones polinómicas y otra con funciones trigonométricas. Vamos a dividir en dos los tipos de aproximación que podemos tener para el problema de aprendizaje.

1. Aproximaciones universales: son aquellas en las que se establece que cualquier función continua puede ser aproximada por otra función de otra clase con el error que queramos. En este grupo podríamos meter a los dos teoremas que hemos dado previamente. Dentro de este grupo podemos tener diferentes tipos de aproximaciones en función de la familia de funciones que escojamos como aproximaciones. Por ejemplo en los dos teoremas previos hemos cogido las clases de funciones polinómicas y trigonométricas pero podríamos haber tomado otras clases diferentes.

2. Aproximaciones inexactas: son aquellas en las que no podemos tener una aproximación como las que hemos dado en los teoremas previos, si no que proveen de una aproximación de peor calidad.

3.3.3. Penalización o control de la complejidad

Ya hemos discutido brevemente en la sección de principios inductivos la complejidad y cómo penalizarla. Vamos a ver qué elementos queremos controlar con la penalización:

1. La clase de funciones con la que vamos a hacer la aproximación. Tenemos que decidir si escoger una clase tan amplia que nos aseguremos que abarque la solución seguro pero penalicemos la complejidad de la elección o queremos una clase de funciones más ajustada.

2. Tipo de funcional de penalización. Tenemos que escoger entre los distintos tipos de penalización que queremos. Esto se reduce a escoger entre dos tipos de penalización: paramétrica y no paramétrica. La primera de ellas se basa en estudiar la suavidad del ajuste junto con el

número de parámetros que requiere la aproximación mientras que la segunda intenta estudiar lo mismo, es decir la suavidad del ajuste, sin medir los parámetros de la clase de funciones. En este punto se puede incorporar el conocimiento a priori del problema.

3. Método con el que queremos minimizar la penalización. Este apartado está relacionado con los métodos que tenemos de aprender de los datos y el objetivo será intentar hallar una forma eficiente de minimizar tanto el error de la aproximación como la propia penalización.

4. Control de la complejidad. Como hemos dicho antes el control de la complejidad no es algo sencillo y habrá que escoger la mejor manera de medir dicha complejidad. En secciones posteriores veremos medidas de complejidad como la dimensión de Vapnik-Chervonenkis.

Veamos brevemente la distinción que hemos hecho entre la penalización paramétrica y no paramétrica.

Penalización paramétrica

Supongamos que tenemos un conjunto de funciones $f(x, \omega)$ con $\omega \in \Omega$ donde Ω es el conjunto de parámetros de la forma $\omega = (\omega_0, ..., \omega_m)$. Como la aproximación viene definida por el parámetros ω entonces podemos definir también la penalización asociada a dicha selección de parámetros.

Vamos a ver los ejemplos de las penalizaciones más empleadas de este tipo.

- Ridge: $\phi_r(\omega_m) = \sum_{i=0}^{m} \omega_i^2$
- Selección de subconjunto: $\phi_s(\omega_m) = \sum_{i=0}^{m} \chi(\omega_i \neq 0)$
- Bridge: $\phi_p(\omega_m) = \sum_{i=0}^{m} |\omega_i|^p$
- Decaimiento de peso: $\phi_q(\omega_m) = \sum_{i=0}^{m} \frac{(\omega_i/q)^2}{1+(\omega_i/q)^2}$

Penalización no paramétrica

En primer lugar vamos a definir la transformada de Fourier de una función para poder definir el funcional de penalización.

Definición 3.2 (Transformada de Fourier) *Sea f una función integrable Lebesgue, $f \in L(\mathbb{R})$. Se define la transformada de Fourier de f como la función:*

$$\mathcal{F}\{f\} : \xi \to \hat{f}(\xi) := \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} dx$$

Recordemos brevemente las propiedades de la transformada de Fourier.

- La transformada de Fourier es un operador lineal: $\mathcal{F}\{a \cdot f + b \cdot g\} = a\mathcal{F}\{f\} + b \cdot \mathcal{F}\{g\}$
- $\mathcal{F}\{f(at)\}(\xi) = \frac{1}{|a|} \cdot \mathcal{F}\{f\}(\frac{\xi}{a})$
- $\mathcal{F}\{f(t-a)\}(\xi) = e^{-\pi i \xi a} \cdot \mathcal{F}\{f\}(\xi)$
- $\mathcal{F}\{f\}(\xi - a) = \mathcal{F}\{e^{\pi i a t} f(t)\}(\xi)$
- $\mathcal{F}\{f'\}(\xi) = 2\pi i \xi \mathcal{F}\{f\}(\xi)$
- $\mathcal{F}\{f\}'(\xi) = \mathcal{F}\{(-it) \cdot f(t)\}(\xi)$

Habiendo recordado esto podemos definir el funcional de penalización no paramétrica. Este funcional mide la suavidad del ajuste de la función gracias a que se puede medir, mediante la transformada de Fourier, la ondulación de la función. Por tanto el funcional no paramétrico que se propone es:

$$\phi[f] = \int_{\mathbb{R}^d} \frac{|\hat{f}(s)|^2}{\hat{G}(s)} ds$$

Donde $\hat{f}$ indica la transformada de Fourier de la función f y $\frac{1}{\hat{G}}$ es la transformada de Fourier de una función de filtro de paso alto. Es en esta proposición de filtro donde se añade el conocimiento a priori del problema. Por ejemplo pudiera ser interesante en alguna aplicación práctica tener un funcional invariante frente a rotaciones de funciones.

3.3.4. Equilibrio entre el sesgo y la varianza

Este enfoque es muy utilizado en el estudio del error, dividiéndolo en sesgo y varianza para hacer un mejor estudio del mismo y poder enfrentar ambos con varios métodos. Este estudio del caso clásico no es válido (o al

menos no del todo) para problemas no supervisados como es nuestro caso. Vamos a hacer una adaptación de esta teoría para que pueda encajar en nuestro caso de estudio.

Tenemos que tener en cuenta que no conocemos la salida real del sistema en el caso de detección de anomalías, es decir, no sabemos estimar con certeza el sesgo y la varianza y por tanto el error que cometemos. En primer lugar vamos a ver una pequeña adaptación de la notación al caso de detección de anomalías para poder hacer un estudio enfocado en nuestro problema.

Vamos a notar por $X_1, ..., X_n$ los datos de test y $\mathcal{D}$ como conjunto de datos de entrenamiento. Además vamos a considerar que existe una función f que nos da la etiqueta real de un dato, esto es, si es o no una anomalía. Por tanto podemos decir que la auténtica etiqueta de un dato es $y_i = f(X_i)$. Además nosotros estaremos usando un modelo ya escogido por nosotros para predecir la etiqueta de un dato de test, esto es $g(X_i, \mathcal{D}) \approx y_i + \beta$ donde β es un cierto error.

Una vez conocida esta notación podemos definir el error medio al cuadrado como:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\{y_i - g(X_i, \mathcal{D})\}^2$$

Y podemos definir también el valor esperado del error medio al cuadrado como:

$$E[MSE] = \frac{1}{n}\sum_{i=1}^{n}E[\{y_i - g(X_i, \mathcal{D})\}^2]$$

Una vez definido el MSE esperado podemos desarrollar un poco el cálculo para poder obtener el error y la varianza que esperamos.

En primer lugar podemos escribirlo como:

$$E[MSE] = \frac{1}{n}\sum_{i=1}^{n}E[\{(y_i - f(X_i)) + (f(X_i) - g(X_i, \mathcal{D}))\}^2]$$

Aquí solo hemos restado y sumado $f(X_i)$, ahora si recordamos que $y_i = f(X_i)$ entonces podemos igualar el primero de los paréntesis a 0 y por tanto nos queda:

$$E[MSE] = \frac{1}{n}\sum_{i=1}^{n} E[\{f(X_i) - g(X_i, \mathcal{D})\}^2]$$

Si seguimos descomponiendo podemos sumar y restar $E[g(X_i, \mathcal{D})]$ con lo que nos quedaría:

$$E[MSE] = \frac{1}{n}\sum_{i=1}^{n} E[\{f(X_i) - E[g(X_i, \mathcal{D})]\}^2]$$

$$+\frac{2}{n}\sum_{i=1}^{n}\{f(X_i) - E[g(X_i, \mathcal{D})]\} \cdot \{E[g(X_i, \mathcal{D})] - E[g(X_i, \mathcal{D})]\}$$

$$+\frac{1}{n}E[\{E[g(X_i, \mathcal{D}) - g(X_i, \mathcal{D})]\}^2]$$

Como es claro, el segundo término da cero por lo que nos queda al final:

$$E[MSE] = \frac{1}{n}\sum_{i=1}^{n} E[\{f(X_i) - E[g(X_i, \mathcal{D})]\}^2] + \frac{1}{n}\sum_{i=1}^{n} E[\{E[g(X_i, \mathcal{D})] - g(X_i, \mathcal{D})\}^2]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\{f(X_i) - E[g(X_i, \mathcal{D})]\}^2 + \frac{1}{n}\sum_{i=1}^{n} E[\{E[g(X_i, \mathcal{D})] - g(X_i, \mathcal{D})\}^2]$$

Si reconocemos cada uno de los términos, en primer lugar el primero de ellos es el sesgo al cuadrado y el segundo la varianza, por lo que finalmente lo que hemos obtenido es:

$$E[MSE] = sesgo^2 + varianza$$

El dilema que se nos plantea es el siguiente: si tomamos modelos con un bajo sesgo en la estimación de los parámetros entonces tendremos una alta varianza y viceversa. Esto significa que no podemos con el conocimiento del que disponemos disminuir tanto el sesgo como la varianza a la vez. Es esta propiedad la que se conoce como la compensación entre sesgo y varianza.

3.4. Teoría estadística del aprendizaje

En esta sección vamos a hacer un repaso por la teoría del aprendizaje, en concreto la teoría desarrollada por Vapnik-Chervonenkis. Esta teoría se basa o tiene como pilares cuatro puntos:

1. Condiciones para la consistencia del principio ERM o minimización del error empírico.
2. Cotas en la capacidad de generalización de las máquinas de aprendizaje.
3. Principios de inferencia sobre muestras finitas.
4. Métodos constructivos para implementar los principios inductivos ya expuestos.

Durante el desarrollo de esta sección haremos un repaso de estos cuatro puntos para dar el broche final a esta sección y poder realizar la primera de las definiciones de anomalía.

3.4.1. Condiciones para la convergencia y consistencia del ERM

En el problema de aprendizaje disponemos de una muestra en la que tenemos los propios datos de entrada y la salida del sistema. Denotemos a estos elementos por $z = (x, y)$ donde x son los datos de entrada e y la salida del sistema. Por tanto la muestra que se nos da es un conjunto $Z_n = \{z_1, ..., z_n\}$. Estos datos están generados como ya sabemos mediante una función de densidad desconocida $p(z)$. Sobre este esquema tenemos una serie de funciones de pérdida y un funcional de pérdida. El objetivo es encontrar dicha función de pérdida $Q(z, \omega)$ que minimice dicho funcional:

$$R(\omega) = \int Q(z, \omega)p(z)dz$$

Por tanto si tenemos la función de pérdida podemos definir el error empírico como:

$$R_{emp}(\omega) = \sum_{i=1}^{n} Q(z_i, \omega)$$

Donde ω son los parámetros escogidos para el modelo.

Para poder estudiar la consistencia del ERM primero debemos definir formalmente dicha propiedad. Denotamos por $R_{emp}(\omega_n^*)$ el valor del error empírico con la función de pérdida $Q(z, \omega_n^*)$ que minimiza el error empírico para el conjunto de entrenamiento Z_n. Denotemos además por $R(\omega_n^*)$ el verdadero valor (desconocido) del error para la función de pérdida. Como se puede ver estos valores dependen del tamaño del conjunto de entrenamiento n, podemos por tanto estudiar cómo se comportan estos errores cuando aumentamos el tamaño del conjunto de entrenamiento. Es aquí donde entra la definición de consistencia del ERM. Decimos que es consistente si la sucesión de errores reales y empíricos convergen en probabilidad al mismo límite $R(\omega_0) = \text{mín}_\omega R(\omega)$. Es decir:

$$R(\omega_n^*) \to R(\omega_0) \textit{ cuando } n \to \infty$$
$$R_{emp}(\omega_n^*) \to R(\omega_0) \textit{ cuando } n \to \infty$$

Para poder asegurar esta propiedad sobre el ERM tenemos el conocido como Teorema Clave de la Teoría del Aprendizaje de Vapnik y Chervonenkis.

Teorema 3.3 (Teorema Clave de la Teoría del Aprendizaje) *Para funciones de pérdida acotadas el principio inductivo de minimización del error empírico es consistente sí y sólo si el error empírico converge uniformemente al valor real del error en el siguiente sentido:*

$$\lim_{n\to\infty} P[\sup_{\omega} |R(\omega) - R_{emp}(\omega)| > \epsilon] = 0 \ , \ \forall \epsilon > 0$$

Cabe recalcar que estas condiciones de consistencia dependen de las propiedades de la clase de funciones elegida. No podemos pretender escoger como clase de aproximación una muy general y seguir manteniendo las condiciones de consistencia del ERM. Aún así el teorema nos está dando condiciones generales para la consistencia del ERM pero son abstractas y no fácilmente aplicables en la práctica. Para ello vamos a estudiar las condiciones de convergencia de ERM que sí serán aplicables en la implementación de algoritmos.

Vamos ahora a particularizar el estudio en el caso de clasificación binaria por ser la materia de estudio que nos ocupa, pues al final tendremos que clasificar instancias en anómalas o no anómalas. Ahora las funciones de pérdida $Q(z, \omega)$ son funciones de pérdida indicadoras. Vamos a notar por $N(Z_n)$ el número de dicotomías que se pueden tener con la clase de funciones

elegidas. Esto es el número de formas de clasificar los datos en las dos clases existentes.

Una vez actualizada nuestra notación podemos definir la entropía aleatoria como $H(Z_n) = \ln N(Z_n)$. Esta cantidad es una variable aleatoria dependiente de los valores de entrenamiento Z_n, podemos definir ahora la entropía de Vapnik-Chervonenkis como el valor medio o esperado de la entropía aleatoria:

$$H(n) = E[\ln N(Z_n)]$$

Esta medida es una cuantificación de la diversidad del conjunto de funciones indicadoras que nos pueden separar los datos en ambas clases.

Por último vamos a definir la función de crecimiento que nos va a permitir hacer cotas y llegar a la condición necesaria y suficiente para la convergencia del ERM. Definimos la función de crecimiento como:

$$G(n) = \ln \max_{Z_n} N(Z_n)$$

Donde aquí estamos notando el máximo número de dicotomías sobre todas las posibles muestras existentes de tamaño n. Es más, como el máximo número de formas de dividir un conjunto de tamaño n en dos clases es 2^n entonces podemos afirmar que $G(n) \leq n \ln(2)$.

Por último y para completar la cadena de desigualdades que buscamos vamos a definir la entropía reforzada de Vapnik-Chervonenkis:

$$H_{ann}(n) = \ln(E[N(Z_n)])$$

Haciendo uso de la conocida desigualdad de Jensen,

$$\sum_{i=1}^{n} a_i \ln(x_i) \leq \ln(\sum_{i=1}^{n} a_i x_i)$$

,

podemos ver claramente que $H(n) \leq H_{ann}(n)$. Por tanto obtenemos la cadena de desigualdades:

$$H(n) \leq H_{ann}(n) \leq G(n) \leq n \ln(2)$$

La condición necesaria y suficiente de Vapnik-Chervonenkis para la convergencia del ERM que hallaron fue que:

$$\lim_{n\to\infty} \frac{H(n)}{n} = 0$$

Pero esta condición no asegura una convergencia rápida asintóticamente al error real. Se dice que el ratio de convergencia es rápido asintóticamente en la Teoría de Vapkin-Chervonenkis si:

$$\forall n > n_0 \; P(R(\omega) - R(\omega^*) < \epsilon) = e^{-cn\epsilon^2} \; con \; c > 0$$

Para poder cumplir esta condición se dio la condición suficiente para la convergencia rápida:

$$\lim_{n\to\infty} \frac{H_{ann}(n)}{n} = 0$$

Estas condiciones son dependientes de la distribución de los datos Z_n como es claro al depender de la esperanza de una variable aleatoria dependiente de Z_n. Es por tanto que esta condición no es del todo general. Para solventar esto se tiene la consistencia y convergencia del ERM con la condición necesaria y suficiente de que:

$$\lim_{n\to\infty} \frac{G(n)}{n}$$

Por tanto este estudio nos ha dado las condiciones de convergencia y consistencia del ERM.

3.4.2. Función de crecimiento y dimensión de Vapnik-Chervonenkis

El objetivo que perseguimos es obtener cotas para la capacidad de generalización de las máquinas de aprendizaje. Para dar el primer paso según hemos visto necesitamos una forma de evaluar la función de crecimiento vista en el apartado anterior, cosa que no es sencilla de llevar a la práctica.

Para continuar avanzando en este camino lo primero que vamos a presentar es el concepto de dimensión de Vapnik-Chervonenkis o dimensión VC. Cuando discutimos cómo medir la complejidad de un modelo ya hablamos

de que la dimensión VC podría ser una buena herramienta para este fin, la introducimos a continuación.

Vapnik y Chervonenkis probaron que la función de crecimiento estaba acotada por una función logarítmica en función del tamaño de la muestra. El punto en el que se tiene $n = h$ donde h es un valor fijo se tiene que el crecimiento de la función de crecimiento empieza a ralentizarse, esta es la conocida como dimensión VC. Si h es un número finito entonces tenemos que la función de crecimiento no va a crecer de forma lineal para muestras de tamaño grande y de hecho se tiene la cota:

$$G(n) \leq h(1 + \ln(\frac{n}{h}))$$

La dimensión de Vapnik-Chervonenkis es intrínseca a la elección del conjunto de funciones y además nos da condiciones sobre la convergencia rápida del ERM. Ya hemos visto antes que la cota más grande de la función de crecimiento es:

$$G(n) \leq n \ln(2)$$

Si comparamos las dos cotas en función de n tenemos la siguiente gráfica:

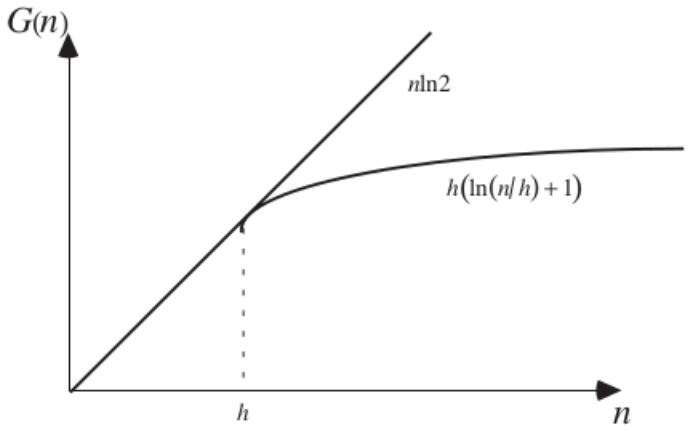

Figura 3.4: Comportamiento de la función de crecimiento [2, p. 107]

Como podemos ver el comportamiento una vez que el tamaño de la muestra alcanza la dimensión VC converge de forma mucho más rápida.

Hasta ahora no hemos definido formalmente la dimensión VC pero hemos dado una característica de la misma que nos garantiza una buena convergencia. Decimos que un conjunto de funciones indicadoras tiene dimensión VC h si existe una muestra de puntos de tamaño h que puede ser dividida pero no existe una muestra de tamaño $h+1$ que cumpla dicha condición. Es decir, podemos decir que la dimensión VC es la máxima dimensión para la que existe una solución óptima a nuestro problema de dividir el conjunto de datos entre datos anómalos y normales.

Veamos esto con un ejemplo gráfico:

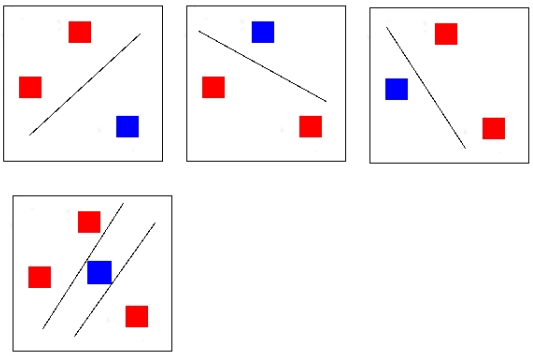

Figura 3.5: Ejemplo de cálculo de dimensión VC Wikimedia

Como podemos ver si estamos considerando funciones lineales para aproximar podemos dividir todas las posibilidades con tamaño de muestra 3, pero no con tamaño de muestra 4 por lo que en este caso concreto la dimensión VC será 3.

Ahora podemos volver a la cadena de desigualdades que hemos visto en la sección anterior y actualizarla con la nueva mejor cota que hemos desarrollado con la dimensión VC:

$$H(n) \leq H_{ann}(n) \leq G(n) \leq h(1 + \ln(\frac{n}{h}))$$

La definición que hemos dado es para funciones indicadoras pero no para funciones reales en general, vamos a generalizar por tanto la definición de la dimensión de Vapnik-Chervonenkis para el caso de funciones reales.

Consideramos como hemos hecho anteriormente funciones de pérdida del tipo $Q(z, \omega)$ pero acotadas superior e inferiormente por constantes:

$$A \leq Q(z, \omega) \leq B$$

Para este caso podemos pensar en una función indicadora que nos diga si $Q(z, \omega)$ está por encima o por debajo de un cierto valor β con $A \leq \beta \leq B$. Podemos por tanto generalizar la dimensión VC para el caso de funciones de pérdida reales como la dimensión VC de este tipo de funciones indicadoras dependientes del parámetro β.

3.4.3. Límites de la generalización

Venimos de discutir las propiedades de convergencia y consistencia del principio inductivo ERM. Ahora bajo este principio vamos a intentar ir un paso mas allá en la generalización e intentar responder a las siguientes dos preguntas:

1. ¿Cómo de cerca están el error real $R(\omega^*)$ y el mínimo error empírico $R_{emp}(\omega^*)$?
2. ¿Cómo de cerca están el error real $R(\omega^*)$ y el mínimo error posible $R(\omega_0) = \text{mín}_\omega\, R(\omega)$?

Estas preguntas las vamos a resolver en el marco que hemos introducido, con todos los conceptos anteriores de la teoría de Vapnik-Chervonenkis en el caso del problema de clasificación binaria que es el que más se ajusta a nuestro problema.

Según Vapnik-Chervonekis se puede acotar el error real cometido por el error empírico con una probabilidad de al menos $1 - \eta$ usando el principio inductivo ERM. La cota que hallaron es la siguiente:

$$R(\omega) \leq R_{emp}(\omega) + \frac{\epsilon}{2}\left(1 + \sqrt{1 + \frac{4 \cdot R_{emp}(\omega)}{\epsilon}}\right)$$

donde:

$$\epsilon = a_1 \cdot \frac{h(\ln(\frac{a_2 n}{h}) + 1) - \ln(\frac{\eta}{4})}{n}$$

cuando el conjunto de funciones de pérdida $Q(z, \omega)$ contiene un número infinito de elementos, en caso contrario:

$$\epsilon = 2\frac{\ln(N) - \ln(\eta)}{n}$$

y los valores a_1, a_2 son constantes sobre los que se exigen condiciones.

Para empezar según la demostración de Vapnik, los valores a_1 y a_2 deben estar en los rangos $0 < a_1 \leq 4$ y $0 < a_2 \leq 2$ siendo la pareja de valores $a_1 = 4$ y $a_2 = 2$ la correspondiente al peor de los casos, dando en como resultado el siguiente valor de ϵ:

$$\epsilon = 4\frac{h(\ln(\frac{2n}{h})) - \ln(\frac{\eta}{4})}{n}$$

Con esta desigualdad estamos dando la cota de cómo se comporta el error real con respecto al error empírico. Podemos ver que, en el mejor de los casos el error real y el error empírico se van a diferenciar en al menos $\frac{\epsilon}{2}$.

Además, para resolver la segunda de las preguntas la teoría de Vapnik-Chervonenkis nos da la siguiente cota con probabilidad al menos $1 - 2\eta$:

$$R(\omega_n^*) - \min_{\omega} R(\omega) \leq \sqrt{\frac{-\ln(\eta)}{2n}} + \frac{\epsilon}{2}\left(1 + \sqrt{1 + \frac{4}{\epsilon}}\right)$$

En este caso podemos ver que la cota es aún mayor que en el caso anterior, teniéndose tanto en esta cota como en la anterior que a mayor nivel de confianza (menor valor de η) mayor es la cota y por tanto menos información tenemos. Es decir, no podemos conocer una buena cota con un nivel de confianza alto. Este hecho no debería de sorprendernos pues seguimos trabajando con un número finito de datos y ya sabemos que no podemos obtener una aproximación todo lo buena que queramos con un número finito de datos de entrenamiento.

Este hecho también se refleja en los valores analíticos que hemos dado. Pensemos en un escenario con $\eta \to 0$, es decir un alto nivel de confianza. Entonces si miramos la expresión de ϵ podemos observar que $-\ln(\frac{\eta}{4}) \to \infty$ y por tanto $\epsilon \to \infty$ con lo que la cota no nos aportaría ninguna información en ninguno de los dos casos.

Por otro lado si lo que crece es el tamaño de la muestra, es decir, $n \to \infty$, estamos aumentando el conocimiento que tenemos sobre el problema y por tanto lo razonable sería que ambas cotas tendieran al valor óptimo. En efecto si observamos el valor de ϵ cuando $n \to \infty$ vemos que tiende a 0 por lo que el error empírico y real están muy cerca y de igual forma el error real y el

mínimo error posible. Por tanto podemos decir que nuestro nivel de certeza depende del tamaño de la muestra. Este hecho fue visto por Vapnik en su teoría y propuso como valor aproximado de la confianza de la desigualdad aquel que lleva asociado el valor:

$$\eta = \text{mín}(\frac{4}{\sqrt{n}}, 1)$$

3.4.4. Principio de minimización del error estructural (SRM)

Hemos visto una construcción en base al principio inductivo ERM y hemos razonado que funciona bien para casos en los que la proporción $\frac{n}{h}$, es decir la proporción del tamaño de la muestra y la dimensión VC, es grande. En este caso quiere decir que tenemos muchos datos comparado con la dimensión VC y por tanto $\epsilon \approx 0$. Por contra cuando tenemos que $\frac{n}{h}$ es pequeño no tenemos mucha información de la cota. Por tanto, al estar el número de datos fijo por el problema, tenemos que buscar un conjunto de funciones para aproximar la salida del sistema que nos den una dimensión VC controlable para hacerla más o menos grande.

El principio inductivo que pretende plasmar esta idea es el principio de minimización del error estructural o SRM. Bajo este principio se le otorga a la clase de funciones de pérdida de una estructura, es decir, tenemos subconjuntos de la forma $S_k = \{Q(z, \omega),\ \omega \in \Omega_k\}$ de forma que:

$$S_1 \subset S_2 \subset ... \subset S_k \subset ...$$

donde cada subconjunto de funciones de pérdida tiene asociada una dimensión VC h_k teniéndose el orden:

$$h_1 \leq h_2 \leq ... \leq h_k \leq ...$$

Al igual que en el Teorema clave de la Teoría del Aprendizaje de Vapnik-Chervonenkis se exigía que las funciones de pérdida estuvieran acotadas en este caso vamos a pedir que las funciones contenidas en cada uno de los S_k o bien estén acotadas o si no que cumplan que:

$$\sup_{\omega \in \Omega_k} \frac{(\int Q^p(z, \omega) dp(z))^{\frac{1}{p}}}{\int Q(z, \omega) dp(z)} \leq \tau_k,\ p > 2$$

para alguna pareja (p, τ_k).

En cuanto a la definición del SRM hay dos estrategias prácticas que se llevan a cabo para su implementación que son:

1. Mantener la dimensión VC fija y minimizar el error empírico.
2. Mantener el error empírico constante y pequeño y minimizar la dimensión VC.

Estas implementaciones realmente quedan muy libres en la práctica y se proponen por tanto diferentes estructuras de minimización del error empírico y la dimensión VC que se saben que funcionan bien.

Por tanto este principio no se basa meramente en el buen ajuste de los datos, si no que además pretende hacer una minimización de la complejidad del modelo propuesto.

3.4.5. Aproximaciones de la dimensión VC

Como hemos estado viendo las cotas que hemos expuesto y desarrollado dependen en mayor o menor medida de la dimensión VC. Como es lógico este valor no es fácil de calcular, y de hecho sólo se sabe para unos cuantos conjuntos de funciones de aproximación. Vapnik propuso una método para poder estimar este valor y así poder obtener unas cotas aproximadas.

El procedimiento propuesto por Vapnik consiste en, dadas dos muestras Z_n^1, Z_n^2 de tamaño n de pares $z_i = (x, y)$ de datos de entrada y salida del sistema vamos a medir el error empírico con nuestro modelo que cometemos observando la máxima desviación de los ratios de error de estas dos muestras independientes, es decir:

$$\xi(n) = \max_{\omega}(|Error(Z_n^1) - Error(Z_n^2|)$$

donde $Error(Z_n^i)$ es la tasa de error empírico cometido por el modelo. De acuerdo con la teoría desarrollada por Vapnik-Chervonenkis tenemos que $\xi(n)$ está acotada:

$$\xi(n) \leq \Phi(\frac{n}{h})$$

donde h es la dimensión VC y:

$$\Phi(\tau) = \begin{cases} 1 & si\ \tau < 0.5 \\ a\frac{\ln(2\tau)+1}{\tau-k}\left(\sqrt{1+\frac{b(\tau-k)}{\ln(2\tau)+1}}+1\right) & en\ otro\ caso \end{cases}$$

donde $\tau = \frac{n}{h}$ y las constantes $a = 0.16$ y $b = 1.2$ son constantes estimadas empíricamente por Vapnik y $k = 0.14928$ tomada así para que $\Phi(0.5) = 1$ de forma que la cota sea muy ajustada.

Por tanto describió con esto el siguiente esquema de obtención de la aproximación de la dimensión VC:

1. Generamos una muestra de tamaño $2n$ etiquetada z_{2n}
2. Dividimos la muestra en dos del mismo tamaño Z_n^1 y Z_n^2
3. Invertir las etiquetas de Z_n^2
4. Mezclar los dos conjuntos de nuevo y entrenar el modelo
5. Separar el conjunto en dos de nuevo e invertir las etiquetas del segundo, volver a mezclarlos y entrenar de nuevo el modelo
6. Medir la diferencia de los errores $\xi(n) = |Error(Z_n^1) - Error(Z_n^2)|$

Como hemos dicho antes la desigualdad $\xi(n) \leq \Phi(\frac{n}{h})$ es muy ajustada y por tanto, podemos obtener la aproximación de h como:

$$h^* = arg\min_h[\xi(n) - \Phi(\frac{n}{h})]$$

Es decir, el valor que haga dicha diferencia menor, es decir que más acerque los valores $\xi(n)$ y $\Phi(\frac{n}{h})$.

3.4.6. Perspectiva

Tras este desarrollo teórico hemos dado un marco sobre el cuál podremos cimentar el resto del trabajo y modelos empleados. Con este capítulo hemos hecho un repaso por toda la teoría básica de Machine Learning y la teoría desarrollada por Vapnik y Chervonenkis.

El siguiente paso que debemos de dar es una breve introducción de estadística multivariante que nos permita introducir nociones de probabilidad

para definir las anomalías desde un punto de vista algo más formal. Tras esto, debemos introducir nociones sobre el Aprendizaje Profundo o Deep Learning desde un punto de vista teórico para poder juntar todo esto en la sección práctica de este trabajo.

Capítulo 4

Introducción de Estadística Multivariante

Para poder proseguir en el estudio debemos hacer un repaso breve de conceptos de estadística multivariante. El contenido de esta sección se ha sacado básicamente de apuntes de la asignatura Estadística Multivariante del grado en Matemáticas, los apuntes de la asignatura Procesos Estocásticos del grado en Matemáticas y el libro Probability Theory de M. Loève [6].

Vamos a dar otra definición de anomalía que no coincide con la que hemos visto basada en distancias, pero antes de dar esa definición debemos hacer un breve repaso de estadística multivariante y probabilidad para poder comprender y enmarcar dicha definición.

4.1. Introducción

En primer lugar vamos a describir conceptos básicos sobre los que poder construir los conceptos que necesitamos para la definición de anomalía basada en probabilidades.

En primer lugar vamos a definir el concepto de variable aleatoria.

Definición 4.1 *Una variable aleatoria es una función $X : \Omega \to E$ que parte de un espacio de probabilidad $(\Omega, \mathcal{F}, \mathcal{P})$ y llega a un espacio medible $(E, \mathcal{B})$, donde X además es una función medible.*

Normalmente ya sabemos que $E \subseteq \mathbb{R}$ y además cabe recordar que $\mathcal{F}$ es

una σ-álgebra. Además cabe recordar la definición de función medible:

Definición 4.2 *Decimos que una función $X : (\Omega, \mathcal{F}, \mathcal{P}) \to (E, \mathcal{B})$ es medible si $X^{-1}(B) \subset \mathcal{F}$, $\forall B \in \mathcal{B}$.*

Esta definición puede extenderse al caso vectorial, introduciendo con esto la noción de vector aleatorio:

Definición 4.3 *Un vector aleatorio $\underline{X} = (X_1, ..., X_p)$ es una aplicación medible $\underline{X} : (\Omega, \mathcal{F}, \mathcal{P}) \to (E, \mathcal{B}^p)$ donde $E \subseteq \mathbb{R}^p$.*

Se puede demostrar además la caracterización:

Proposición 4.1 *Un vector $\underline{X} = (X_1, ..., X_p)$ es un vector aleatorio si y sólo si $X_i : (\Omega, \mathcal{F}, \mathcal{P}) \to (\mathbb{R}, \mathcal{B})$ es una función medible.*

Con este vector aleatorio podemos estudiar o definir la distribución de probabilidad del mismo sobre $(\mathbb{R}^p, \mathcal{B}^p)$ $P_{\underline{X}}$ como:

$$P_{\underline{X}}[B] := P[\underline{X}^{-1}(B)] \ \forall B \in \mathcal{B}$$

con lo que el espacio $(\mathbb{R}^p, \mathcal{B}^p, P_{\underline{X}})$ es un espacio de probabilidad o probabilístico.

Sobre los conocimientos de la definición de la función de distribución univariante podemos hacer una definición análoga para el caso multivariante.

Definición 4.4 *Se define la función de distribución asociada a la probabilidad inducida como:*

$$F_{\underline{X}}(\underline{x}) = P_{\underline{X}}[X_1 \leq x_1, ..., X_p \leq x_p] \ , \ \forall \underline{x} = (x_1, ..., x_p) \in \mathbb{R}^p$$

De igual forma podemos caracterizar la función de densidad como aquella $f_{\underline{X}}$ que, de existir, cumple que:

$$F_{\underline{X}}(\underline{x}) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} ... \int_{-\infty}^{x_p} f_{\underline{X}}(u_1, ..., u_p) du_1 ... du_p$$

Otra forma de determinar de forma única la distribución de un vector aleatorio es mediante la función característica, lo que nos va a dar además una caracterización de la independencia que introduciremos seguidamente.

Definición 4.5 *Dado un vector aleatorio $X = (X_1, ..., X_p)$ se define la función característica como $\Phi_{\underline{X}}(\underline{t}) = E[e^{i\underline{t}\underline{X}}]$ con $\underline{t} = (t_1, ..., t_p) \in \mathbb{R}^p$ donde la función $E[\cdot]$ denota la esperanza, por lo que:*

$$\Phi_{\underline{X}}(\underline{t}) = \int_{\mathbb{R}^p} e^{i\underline{t}\underline{X}} P_{\underline{X}}(d\underline{x})$$

Con esto ya podemos introducir el concepto de independencia en varias variables.

4.1.1. Independencia

Definición 4.6 *Dados dos vectores aleatorios $\underline{X} = (X_1, ..., X_p)$, $\underline{Y} = (Y_1, ..., Y_p)$ se dice que son independientes si:*

$$F_{\underline{X},\underline{Y}}(\underline{x}, \underline{y}) = F_{\underline{X}}(\underline{x}) \cdot F_{\underline{Y}}(\underline{y})$$

Podemos también definir la independencia entre las variables de un vector aleatorio como:

Definición 4.7 *$X = (X_1, ..., X_p)$ se dice que está compuesto de variables independientes si $\forall B = B_1 \times ... \times B_p$ con $B_i \in \mathcal{B}$ se tiene que:*

$$P_{\underline{X}}(B) = P_{X_1}[B_1] \cdot ... \cdot P_{X_p}[B_p]$$

En cuanto a la independencia de sucesos podemos dar dos definiciones de independencia:

Definición 4.8 *Decimos que los eventos $B = (B_1, ..., B_p)$ son independientes dos a dos si para todos $m \neq k$ se tiene que $P(B_m \bigcap B_k) = P(B_m)P(B_k)$*

Definición 4.9 *Se dice que los eventos $B = (B_1, ..., B_p)$ son independientes mutuamente si para todo $k \leq p$ se tiene que $P(\bigcap_{i=1}^{k} B_i) = \prod_{i=1}^{k}(B_i)$*

En cuanto a la definición de independencia entre las variables aleatorias que definen un vector aleatorio podemos dar dos caracterizaciones basadas en la función característica.

Proposición 4.2 *Si las componentes del vector aleatorio $X = (X_1, ..., X_p)$ son independientes entonces:*

$$\Phi_{\underline{X}}(\underline{t}) = E[e^{i\underline{t}\underline{X}}] = \prod_{j=1}^{p} E[e^{it_j X_j}]$$

Proposición 4.3 *Si las componentes del vector aleatorio $X = (X_1, ..., X_p)$ son independientes entonces la función característica de la variable $Y = \sum_{j=1}^{p} X_j$ es:*

$$\Phi_Y(t) = E[e^{itY}] = E[e^{it\sum_{j=1}^{p} X_j}] = \prod_{j=1}^{p} \Phi_{X_j}(t)$$

4.1.2. Probabilidad y esperanza condicionada

En esta sección vamos a describir la probabilidad y esperanza condicionada de una variable aleatoria y no de un vector aleatorio. Este hecho es sencillo de deducir, pues como hemos introducido previamente la distribución de probabilidad de un vector aleatorio viene determinada por una distribución de probabilidad de una variable aleatoria. Por tanto el estudio de la probabilidad y esperanza condicionada en el caso univariante se hace válido para el caso multivariante.

En primer lugar debemos introducir el concepto de probabilidad condicionada tal y cómo la conocemos hasta ahora de Bayes. Partimos de un espacio de probabilidad $(\Omega, \mathcal{A}, \mathcal{P})$.

Definición 4.10 *Definimos la probabilidad condicionada a un suceso $B \in \mathcal{A}$ con $P(B) > 0$ como:*

$$P(\cdot|B) : \mathcal{A} \to [0, 1], \quad P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Esta es una función de probabilidad, por lo que nos lleva a pensar en el espacio de probabilidad que genera, es más podemos pensar en el espacio de probabilidad en el que la probabilidad condicionada no se anula, es decir:

$$\mathcal{A}_B = \{C = A \cap B,\ A \in \mathcal{A}\}$$

Por tanto solemos considerar como espacio de probabilidad condicionada al espacio $(B, \mathcal{A}_B, P(\cdot|B))$.

Partiendo de este espacio de probabilidad podemos considerar una variable aleatoria $X : (\Omega, \mathcal{A}, \mathcal{P}(\cdot|B)) \to (\mathbb{R}, \mathcal{B})$.

Definición 4.11 *Definimos la esperanza de esta variable aleatoria condicionada a B como:*

$$E[X|B] = \int_\Omega X dP(\cdot|B) = \int_\Omega X dP(\cdot|B) = \frac{1}{P(B)} \int_B X dP = \frac{E[X1_B]}{P(B)}$$

Donde 1_B representa la función indicadora del conjunto B.

No sólo podemos estudiar la probabilidad y esperanzas condicionadas a un evento, si no que también las podemos estudiar condicionadas a una σ-álgebra. En este terreno vamos a distinguir dos posibilidades: condicionamiento a una σ-álgebra generada por una partición numerable de sucesos de probabilidad no nula y condicionamiento a una σ-álgebra arbitraria.

Definición 4.12 *Definimos la esperanza condicionada a una σ-álgebra $\mathcal{A}$ generada por $\{B_n\} \subset \mathcal{A}$ con $B_i \cap B_j = \phi,\ i \neq j$, $\bigcup_{n=1}^{\infty} B_n = \Omega$ y $P(B_i) > 0,\ \forall i$. Siendo la $\mathcal{U} = \sigma(\{B_n\})$ la σ-álgebra generada por $\{B_n\}$. Con este marco, definimos la esperanza de una variable aleatoria $X : (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B})$ condicionada a la σ-álgebra $\mathcal{U}$ como:*

$$E[X|\mathcal{U}](\omega) = \sum_{n=1}^{\infty} E[X|B_n] 1_{B_n}(\omega)$$

Propiedades 4.1

1. *$E[X|\mathcal{U}] : (\Omega, \mathcal{U}) \to (\mathbb{R}, \mathcal{B})$ es $\mathcal{U}$-medible.*
2. *$E[E[X|\mathcal{U}]] = \sum_{n=1}^{\infty} E[X|B_n]P(B_n) = \sum_{n=1}^{\infty} E[X1_{B_n}] = E[X]$*

De igual forma podemos definir la probabilidad condicionada a una σ-álgebra generada por una partición numerable de sucesos no nulos.

Definición 4.13 *Definimos la probabilidad de un suceso $A \in \mathcal{A}$ condicionada a la σ-álgebra $\mathcal{U}$ como:*

$$P(A|\mathcal{U}) = E[1_A|\mathcal{U}] = \sum_{n=1}^{\infty} E[1_A|B_n]1_{B_n} = \sum_{n=1}^{\infty} P(A|B_n)1_{B_n}$$

casi seguramente.

Podemos también dar unas propiedades inmediatas de la probabilidad condicionada tomando como base las de la esperanza.

Propiedades 4.2 *1.* $P(A|\mathcal{U})$ *es* $\mathcal{U}$*-medible.*

2. $E[P(A|\mathcal{U})] = P(A)$

Una vez visto esto podemos hacer una definición con una σ-álgebra arbitraria. Cabe decir que en este caso no vamos a poder dar una definición constructiva y fácil de calcular como sí hemos hecho en el caso particular anterior. Lo que sí vamos a tener con esta definición más general es el mantenimiento de las propiedades que hemos visto en primera instancia tanto de la probabilidad como de la esperanza condicionada. Sobra decir además que esta definición coincide con la anterior en el caso particular de una σ-álgebra generada por una partición numerable de sucesos no nulos.

Definición 4.14 *Definimos la esperanza de una variable aleatoria* X *en el marco dado condicionada a una* σ*-álgebra* $\mathcal{U} \subset \mathcal{A}$ *como la única función* $\mathcal{U}$*-medible tal que:*

$$\forall u \in \mathcal{U} \ \int_{\mathcal{U}} E[X|\mathcal{U}]P_{\mathcal{U}} = \int_{\mathcal{U}} X dP$$

casi seguramente $P_{\mathcal{U}}$*. Donde* $\forall u \in \mathcal{U} \ P_{\mathcal{U}}(u) = P(u)$*.*

Igualmente podemos dar una definición de la probabilidad condicionada a una σ-álgebra arbitraria tomando como base la definición de esperanza condicionada.

Definición 4.15 *Definimos la probabilidad de* $A \in \mathcal{A}$ *condicionada a la* σ*-álgebra* $\mathcal{U}$ *como:*

$$P(A|\mathcal{U}) = E[1_A|\mathcal{U}]$$

casi seguramente $P_{\mathcal{U}}$*.*

Por último antes de dar unas propiedades que nos den un poco más de conocimiento y herramientas de trabajo vamos a ver el concepto de probabilidad y esperanza condicionada a una variable aleatoria y no a un suceso o una σ-álgebra como hemos visto previamente.

Partimos igualmente del marco $(\Omega, \mathcal{A}, P)$ con dos variables aleatorias X, Y.

Definición 4.16 *Definimos la σ-álgebra generada por la variable aleatoria Y como la menor σ-álgebra que hace medible a la variable aleatoria Y y la notaremos como $\sigma(Y)$.*

Ahora si podemos definir la esperanza de una variable aleatoria condicionada a otra.

Definición 4.17 *Definimos la esperanza de la variable aleatoria X condicionada a la variable aleatoria Y como:*

$$E[X|Y] = E[X|\sigma(Y)]$$

Como anotación cabe decir que esta esperanza condicionada es una función dependiente de la variable aleatoria Y, es decir podemos expresarla como:

$$g(y) = E[X|Y = y]$$

Ahora que tenemos la definición de la esperanza condicionada a una variable aleatoria podemos usar el concepto como hemos hecho anteriormente para definir la probabilidad de un suceso condicionado a una variable aleatoria.

Definición 4.18 *Para todo $A \in \mathcal{A}$ definimos la probabilidad de A condicionada a la variable aleatoria Y como:*

$$P(A|Y) = E[1_A|\sigma(Y)]$$

casi seguramente $P_{\sigma(Y)}$

Ahora estamos en condiciones de dar una propiedades elementales y de suavizamiento que nos van a dar herramientas con las esperanzas condicionadas. En este punto ya hemos visto que, al haber hecho las definiciones de

esperanza y probabilidades usándolas indistintamente las propiedades que vamos a dar para la esperanza se pueden emplear para las probabilidades utilizando sus definiciones que impliquen el uso de esperanzas.

Sobre estas propiedades vamos a realizar algunas de las demostraciones de las propiedades elementales y de las de suavizamiento que vamos a dar para poner de relieve cómo podemos hacer uso de la probabilidad y esperanza condicionada.

Propiedades 4.3 (Propiedades elementales) *Partimos de un espacio de probabilidad $(\Omega, \mathcal{A}, P)$, $\mathcal{U}$ una σ-álgebra contenida en $\mathcal{A}$ y X, Y variables aleatorias integrables.*

1. $E[cte|\mathcal{U}] = cte$ *casi seguramente* $P_{\mathcal{U}}$
2. *Sean* $a, b \in \mathbb{R}$ $E[aX + bY|\mathcal{U}] = aE[X|\mathcal{U}] + bE[Y|\mathcal{U}]$ *casi seguramente* $P_{\mathcal{U}}$*, es decir, la esperanza condicionada cumple la propiedad de linealidad.*
3. $X \geq Y$ *casi seguramente* $P \Rightarrow E[X|\mathcal{U}] \geq E[Y|\mathcal{U}]$ *casi seguramente* $P_{\mathcal{U}}$*.*
4. $|E[X|\mathcal{U}]| \leq E[|X||\mathcal{U}]$

Demostración 4.1 *Vamos a demostrar la propiedad 1 para ver como trabajar con las igualdades casi seguras.*

1. *Como la igualdad es casi seguramente podemos aplicar integrales en la misma con lo que obtenemos lo siguiente:*

   $$\forall u \in \mathcal{U} \int_u E[cte|\mathcal{U}]dP_{\mathcal{U}} = \int_u ctedP = cteP(u) = cteP_{\mathcal{U}}(u) = \int_u ctedP_{\mathcal{U}}$$

   Como la igualdad es con integrales, podemos decir por tanto que $E[cte|\mathcal{U}] = cte$ *casi seguramente* $P_{\mathcal{U}}$*.*

Propiedades 4.4 (Propiedades de suavizamiento) *Partimos del marco del espacio probabilístico $(\Omega, \mathcal{A}, P)$ con una σ-álgebra $\mathcal{U} \subset \mathcal{A}$.*

1. *Si X es una variable aleatoria integrable y $\mathcal{U}$-medible entonces se tiene que $E[X|\mathcal{U}] = X$ casi seguramente $P_{\mathcal{U}}$*
2. *Sean X, Y variables aleatorias con X $\mathcal{U}$-medible, Y integrable y XY integrable, entonces se tiene que $E[XY|\mathcal{U}] = XE[Y|\mathcal{U}]$ casi seguramente $P_{\mathcal{U}}$.*

3. *Se dice que X es independiente de $\mathcal{U}$ si X y $1_{\mathcal{U}}$ son independientes. Si X es independiente de $\mathcal{U}$ entonces $E[X|\mathcal{U}] = E[X]$ casi seguramente $P_{\mathcal{U}}$.*

4. *Sean $\mathcal{U}_1 \subset \mathcal{U}_2 \subset \mathcal{A}$ y X una variable aleatoria integrable, entonces:*

$$E[X|\mathcal{U}_1] = E[E[X|\mathcal{U}_1]|\mathcal{U}_2] = E[E[X|\mathcal{U}_2]|\mathcal{U}_1]$$

casi seguramente $P_{\mathcal{U}}$.

Vamos a hacer la demostración de las 4 propiedades para dar así una pincelada de cómo aplicar los conceptos vistos hasta ahora.

Demostración 4.2 *Demostremos las propiedades de suavizamiento:*

4. *Sabemos que $E[X|\mathcal{U}_1] = Z$ es $\mathcal{U}_1$-medible y por tanto es $\mathcal{U}_2$-medible, por lo que $E[Z|\mathcal{U}_2] = Z$ casi seguramente $P_{\mathcal{U}_2}$.*

   Vamos a utilizar ahora el hecho de que las igualdades son casi seguramente y por tanto vamos a ver si aplicando integrales en ambos lados de la igualdad obtenemos el mismo resultado y confirmamos la igualdad.

   $\forall u \in \mathcal{U}_1 \subset \mathcal{U}_2$ *tenemos* $\int_u E[E[X|\mathcal{U}_1]|\mathcal{U}_2]dP_{\mathcal{U}_2} = \int_u E[X|\mathcal{U}_1]dP_{\mathcal{U}_1} = \int_u XdP$

   Veamos ahora desarrollando el otro término.

   $\int_u E[E[X|\mathcal{U}_2]|\mathcal{U}_1]dP_{\mathcal{U}_1} = \int_u E[X|\mathcal{U}_2]dP_{\mathcal{U}_2} = \int_u XdP$

   Al haber llegado a la misma igualdad en integrales tenemos por tanto la igualdad casi seguramente que buscábamos.

1. *Como X es $\mathcal{U}$-medible entonces tenemos que $\forall u \in \mathcal{U}$ $\int_u E[X|\mathcal{U}]dP_{\mathcal{U}} = \int_u XdP = \int_u XdP_{\mathcal{U}}$ pues al ser $\mathcal{U}$-medible tenemos que $E[X] = \int_\Omega XdP = \int_\Omega XdP_{\mathcal{U}}$.*

3. $\forall u \in \mathcal{U}$ $\int_u E[X|\mathcal{U}]dP_{\mathcal{U}} = \int_u XdP = \int_\Omega 1_u XdP = E[1_u X] =$
   $= E[1_u]E[X] = P(u)E[X] = P_{\mathcal{U}}(u)E[X] = \int_u E[X]dP_{\mathcal{U}}$

Ya hemos dado las definiciones y propiedades de probabilidad y esperanza condicionadas, para finalizar vamos a ver algunas desigualdades famosas que utilizaremos y sus demostraciones.

4.1.3. Desigualdades y fórmulas famosas

Teorema 4.1 (Desigualdad de Markov) *Sea X una variable aleatoria que toma valores no negativos. Entonces para cualquier constante α satisfaciendo $E[X] < \alpha$ se cumple que:*

$$P(X > \alpha) \leq \frac{E[X]}{\alpha}$$

Demostración 4.3 *Denotemos como $f_X(x)$ la función de densidad de la variable aleatoria X. Entonces tenemos:*

$$E[X] = \int_x x f_X(x)dx = \int_{0 \leq x \leq \alpha} x f_X(x)dx + \int_{x>\alpha} x f_X(x)dx$$

$$\geq \int_{x>\alpha} x f_X(x)dx \geq \int_{x>\alpha} \alpha f_X(x)dx$$

La primera de las desigualdades se sigue de la no negatividad de X y la segunda se sigue de que la integral está definida sobre los puntos en los que $x > \alpha$, de hecho:

$$\int_{x>\alpha} \alpha f_X(x)dx = \alpha P(X > \alpha)$$

Con lo que tenemos finalmente que:

$$E[X] \geq \alpha P(X > \alpha) \Leftrightarrow P(X > \alpha) \leq \frac{E[X]}{\alpha}$$

■

Teorema 4.2 (Desigualdad de Chebychev) *Sea X una variable aleatoria arbitraria. Entonces para cualquier constante α se tiene que:*

$$P(|X - E[X]| > \alpha) \leq \frac{Var[X]}{\alpha^2 d}$$

Demostración 4.4 *Sabemos que la desigualdad $|X - E[X]| > \alpha$ es cierta si y sólo si $|X - E[X]|^2 > \alpha^2$*

Vamos a definir la variable aleatoria $Y = (X - E[X])^2$ *es es no negativa. Con esta definición se tiene que* $E[Y] = Var[X]$ *por la propia definición de la variable aleatoria* Y*.*

Entonces la parte izquierda de la desigualdad del teorema se puede expresar como $P(|X - E[X]| > \alpha) = P(Y > \alpha^2)$*. Aplicando aquí la desigualdad de Markov obtenemos que:*

$$P(Y > \alpha^2) \leq \frac{E[Y]}{\alpha^2} = \frac{Var[X]}{\alpha^2}$$

■

Teorema 4.3 (Cota inferior de Chernoff) *Sea* X *una variable aleatoria que se puede expresar como la suma de* N *variables aleatorias independiente de Bernoulli, cada una tomando el valor* 1 *con probabilidad* p_i*.*

$$X = \sum_{i=1}^{N} X_i$$

Entonces para todo $\delta \in (0, 1)$ *tenemos que:*

$$P(X < (1 - \delta)E[X]) < e^{-E[X]\delta^2/2}$$

Teorema 4.4 (Cota superior de Chernoff) *Sea* X *una variable aleatoria que se puede expresar como la suma de* N *variables aleatorias independiente de Bernoulli, cada una tomando el valor* 1 *con probabilidad* p_i*.*

$$X = \sum_{i=1}^{N} X_i$$

Entonces para todo $\delta \in (0, 2 \cdot e - 1)$ *tenemos que:*

$$P(X > (1 + \delta)E[X]) < e^{-E[X]\delta^2/2}$$

Ambas cotas disponen de una demostración que no es constructiva, por lo que no es relevante su demostración para el estudio. Como ejemplo de una demostración de un estilo similar haremos la demostración de la siguiente desigualdad.

Teorema 4.5 (Desigualdad de Hoeffding) *Sea X una variable aleatoria que se puede expresar como suma de N variables aleatorias independientes acotadas en intervalos $[l_i, u_i]$.*

$$X = \sum_{i=1}^{N} X_i$$

Entonces para todo $\theta > 0$ se tienen las cotas:

$$P(X - E[X] > \theta) \leq e^{-\frac{2\theta^2}{\sum_{i=1}^{N}(u_i - l_i)^2}}$$

$$P(E[X] - X > \theta) \leq e^{-\frac{2\theta^2}{\sum_{i=1}^{N}(u_i - l_i)^2}}$$

Demostración 4.5 *Sólo haremos la demostración de la primera desigualdad de forma resumida y sin entrar en los detalles más complejos que se alejan del interés del estudio.*

En primer lugar debemos probar que para todo $t \geq 0$ se cumple la desigualdad:

$$P(X - E[X] > \theta) = P(e^{t(X - E[X])} > e^{t\theta})$$

Usando la desigualdad de Markov podemos probar que $P(e^{t(X-E[X])} > e^{t\theta})$ es como mucho $E[e^{(X-E[X])}]e^{-t\theta}$.

Además al ser variables aleatorias independientes las que componen la variable aleatoria X podemos descomponer el término teniendo la desigualdad:

$$P(X - E[X] > \theta) \leq e^{-t\theta} \prod_i E[e^{t(X_i - E[X_i])}]$$

Cada uno de los términos de este producto se puede probar que vale como mucho $e^{t^2(u_i - l_i)/8}$ usando argumentos de convexidad y el Teorema de Taylor.

Por tanto se cumple:

$$P(X - E[X] > \theta) \leq e^{-t\theta} \prod_i e^{t^2(u_i - l_i)^2/8}$$

Nos interesa hallar el valor de $t = t^*$ que ajusta la desigualdad. Puede demostrarse que ese valor es:

$$t^* = \frac{4\theta}{\sum_{i=1}^{N}(u_i - l_i)^2}$$

Sustituyendo en la desigualdad con este valor de t tenemos el resultado que queríamos probar. ■

Capítulo 5

Concepto probabilístico de anomalía

Tras la introducción dada de estadística multivariante ya tenemos los conceptos necesarios para dar la definición de anomalía basada en probabilidades. Hay muchas formas de definir el concepto de anomalía, pero ninguna es algo estático ni verdad en todos los casos. Esto quiere decir que es muy dependiente del ejemplo de uso que tengamos. Hay veces que el concepto de anómalo como algo separado del resto no es muy intuitivo y por ello vamos a introducir el concepto formal de anomalía basada en probabilidades, con la intención de generalizar algo más el concepto básico. Esta definición está formalmente descrita en el artículo de Fabian Keller [7].

En primer lugar cabe decir que esta definición, al igual que el criterio ya explicado no engloba todas las anomalías y por tanto es algo difícil de medir. Esta definición hace referencia, según mi criterio, a un enfoque que se debe poner junto a la definición basada en distancias y no en contraposición. El objetivo de esta definición es obtener anomalías que no son triviales y se esconden entre los datos.

La base del razonamiento de este tipo de anomalías surge del hecho de que un objeto puede ser anómalo en un subespacio concreto de los datos, pero no en el espacio total. Vamos a introducir un ejemplo para visualizar un tipo de anomalía que encaje con esta definición.

Veamos la siguiente figura:

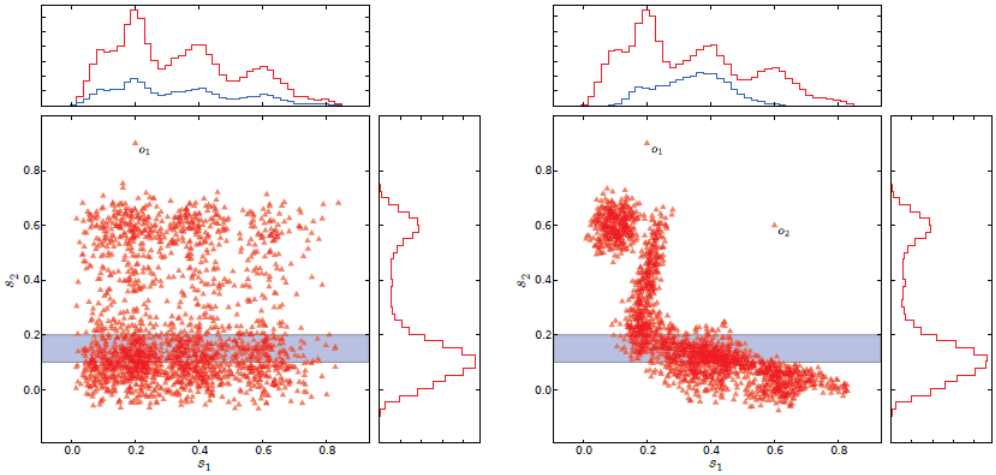

Figura 5.1: Ejemplo de anomalía [**?**]

Como se puede observar tenemos dos espacios: el izquierdo no presenta datos correlados y el derecho sí presenta correlación. Podemos ver que en ambos casos se comparte una anomalía etiquetada como O_1. Esta anomalía en el caso del espacio no correlado es perfectamente detectable de forma trivial observando las proyecciones de los datos en una dimensión. En cambio, en el segundo caso, ninguna de las dos anomalías etiquetadas O_1, O_2 son detectables de esta forma trivial, pues si hacemos las proyecciones uno dimensionales ninguno de los dos datos es discordante en dichas proyecciones. Estas anomalías son las que decimos que son no triviales. En cambio si observamos los datos en una proyección de orden superior como la que estamos viendo de dimensión 2 podemos observar claramente que se salen de la correlación de datos que muestra el resto. Es aquí donde podemos ver que en el conjunto de la derecha ninguno de los puntos es una anomalía en las proyecciones de dimensión uno pero sí lo son en la proyección de dimensión 2.

Vamos por tanto a definir más formalmente este concepto especial de anomalía. Necesitamos introducir en primer lugar un poco de notación.

Partimos de un conjunto de datos $X = \{x_1, ..., x_n\}$ de n objetos cada uno tomando d valores, es decir, $x_i = (x_{s_1}, ..., x_{s_d}) \in \mathbb{R}^d$. Notamos un subespacio del conjunto de valores como:

$$S = \{s_i | s_i \in \{s_1, ..., s_d\} \; con \; i \in \Delta\}$$

Dado un subespacio $S = \{s_1, ..., s_p\}$ notamos la proyección de los objetos del conjunto de datos como $X_S = \{x_{s_1}, ..., x_{s_p}\}$.

Esta proyección está distribuida según una distribución conjunta desconocida de S:

$$p_{s_1,...,s_p}(x_{s_1}, ..., x_{s_p})$$

Notamos la distribución marginal asociada al atributo s_i como:

$$p_{s_i}(x_{s_i})$$

Definición 5.1 *Decimos que un subespacio S es un espacio incorrelado si y sólo si:*

$$p_{s_1,...,s_p}(x_{s_1}, ..., x_{s_p}) = \prod_{i=1}^{p} p_{s_i}(x_{s_i})$$

Por tanto si estamos bajo la suposición de un espacio incorrelado podemos decir que la densidad esperada es:

$$p_{esp}(x_{s_1}, ..., x_{s_p}) \equiv \prod_{i=1}^{p} p_{s_i}(x_{s_i})$$

Recordemos que nuestras anomalías no triviales no están en este tipo de subespacios, si no en los correlados. Por tanto vamos a definirlo de la siguiente forma:

Definición 5.2 *Decimos que un objeto x_S es una anomalía no trivial respecto al subespacio S si:*

$$p_{s_1,...,s_p}(x_{s_1}, ..., x_{s_p}) \ll p_{esp}(x_{s_1}, ..., x_{s_p})$$

Es decir, si la probabilidad esperada es significativamente mayor que la probabilidad conjunta.

Por cómo hemos definido los espacios correlados e incorrelados es claro que no podemos tener anomalías en espacios no correlados como es evidente pues la densidad conjunta y esperada serían iguales.

Este concepto como podemos observar no comparte ninguna relación con la definición clásica de anomalías vista en el máster, por lo que nos ayuda a complementar el concepto.

Capítulo 6

Redes Neuronales y Deep Learning

En este capítulo vamos a dar un repaso a la teoría básica de Redes Neuronales y Deep Learning antes de entrar en la práctica. Repasaremos los fundamentos del aprendizaje con Redes Neuronales, veremos las estructuras de Aprendizaje Profundo o Deep Learning así como las capas de dichas redes que emplearemos en la práctica. Por último veremos una estructura de red que será utilizada en algunos de los modelos, los Autoencoders.

Para la elaboración de este capítulo nos basaremos en los libros de Bengio [8] y Zaccone [9].

6.1. Aprendizaje de las Redes Neuronales

No todas las redes que vamos a emplear en la parte práctica corresponden al modelo "Feedforward", ya que también vamos a elaborar redes neuronales con capas recurrentes, pero vamos a estudiar el comportamiento primero de estas redes para luego explicar las modificaciones que dichas arquitecturas añaden.

En primer lugar, llamamos a este tipo de redes prealimentadas o "Feedforward" en inglés, porque la información fluye siempre en un sentido, desde la entrada hasta la salida obtenida. En las redes denominadas como recurrentes este sentido de la información se revierte en algunos puntos, realimentando la red con la propia salida de algunas capas o de todas ellas.

La representación más común de una red neuronal profunda es a través

de la composición de funciones. Supongamos que estamos aproximando la función $f*(x)$ con una función $f(x)$ construida mediante tres funciones distintas: $f^{(1)}, f^{(2)}$ y $f^{(3)}$ en este mismo orden. Entonces la representación de la función f quedaría como:

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))),$$

donde x es la entrada de la red neuronal o lo que es lo mismo, una instancia o varias de nuestro conjunto de datos. En este caso además decimos que $f^{(1)}$ es la primera capa de la red, $f^{(2)}$ la segunda capa, etcétera.

Este tipo de estructuras se llaman profundas al tener varias capas y por tanto varias funciones que, al componerlas, aproximarán la función objetivo que tenemos. Las capas que se encuentran entre la primera y la última, al no ser capas "visibles" desde el exterior de la red las denominamos capas ocultas.

Estas capas están compuestas de unidades que denominamos neuronas haciendo una equivalencia con el modelo biológico. Una neurona recibe un número de entradas fijado, por ejemplo n. Para cada una de las entradas que recibe va aprendiendo un peso w, que luego multiplicará a cada una de las entradas y sumará para obtener un valor ponderado. A este valor ponderado se le aplica una función de activación que nos convierte dicho valor al rango que nosotros queramos para nuestro problema. Por tanto, una neurona produce como salida la función de activación aplicada a una combinación lineal de las entradas que recibe.

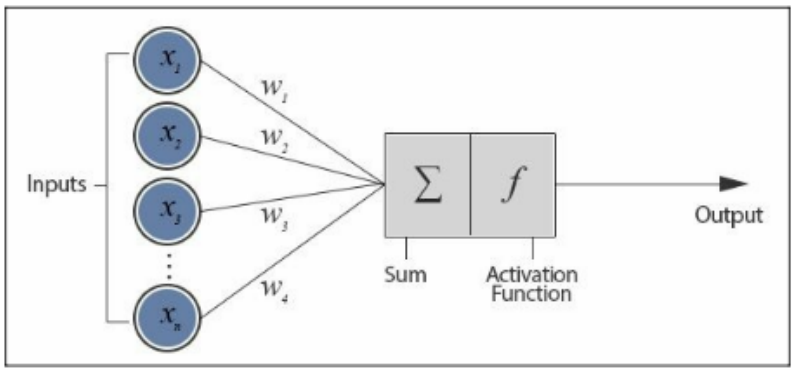

Figura 6.1: Representación de una neurona en una Red Neuronal.

Como un detalle más a tener en cuenta, solemos añadir a las entradas una que denominamos como sesgo. Este sesgo es 1 y se suma a la combinación

lineal, haciendo de término independiente en la ecuación de la recta que se está representando en el espacio de dominio de las instancias.

En cuanto a funciones de activación tenemos muchas sobre las que escoger, veamos las más comunes:

- Rectified Linear Unit (ReLU): esta función de activación se define como
$$ReLU(x) = x^+ = max(0, x) \ , \ x \in \mathbb{R}^d,$$
es decir, cero en los negativos y la función identidad en los positivos.

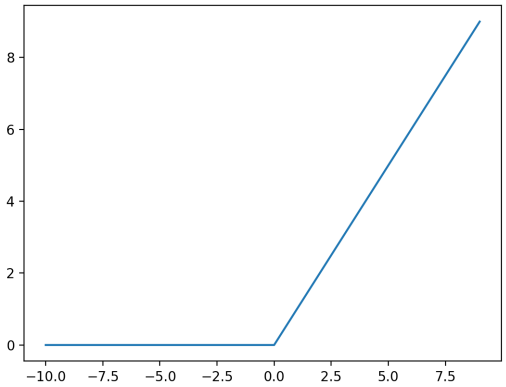

Figura 6.2: Función ReLU.

- Softmax: esta función de activación se define como:
$$softmax : \mathbb{R}^d \to [0, 1]^d$$
$$softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{d} e^{x_k}}$$
Con esta función se obtiene un valor con el mismo número de dimensiones que el que tuvieran los datos de entrada. Además, la salida se puede emplear para representar una distribución de probabilidad. Normalmente esta función de salida se emplea en problemas de clasificación donde vamos a obtener un valor entre 0 y 1 para cada una de las clases, siendo el mayor de estos la clase que el modelo predice.

- Sigmoide: esta función de activación se define como:
$$sigmoide(x) = \frac{1}{1 + e^{-x}}$$

Esta función es una función real de variable real muy conocida y estudiada en matemáticas, con propiedades interesantes como que posee dos asíntotas horizontales y tiene una primera derivada no negativa.

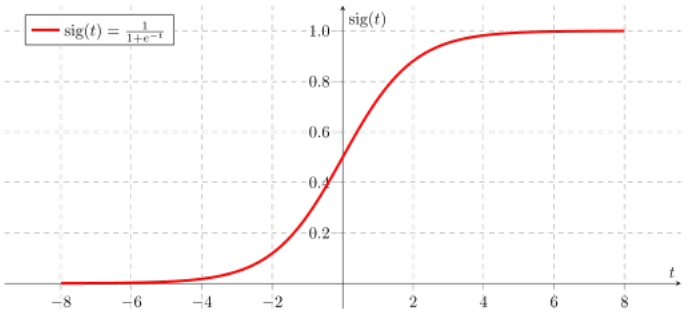

Figura 6.3: Función Sigmoide.

Su valor máximo es 1 y su valor mínimo es 0.

- Tangente hiperbólica: la función tangente hiperbólica se define como:

$$tanh : \mathbb{R} \to \mathbb{R} \ , \ tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

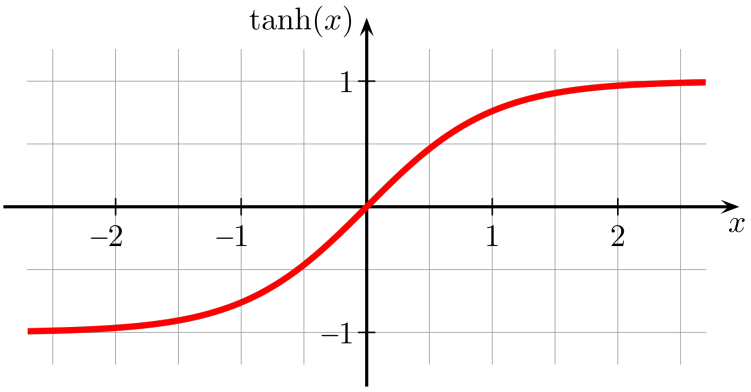

Figura 6.4: Función tangente hiperbólica.

Como podemos ver esta función de activación puede tomar valores en el intervalo $(-1, 1)$.

Una vez que sabemos cuál es el comportamiento básico de una neurona vamos a ver cómo funciona el algoritmo empleado en el aprendizaje de las redes: Backpropagation.

Si describimos el proceso de forma sencilla podemos resumirlo en 4 pasos:

1. Inicializar la red con pesos aleatorios.

2. Calcular el error cometido en la predicción con los pesos actuales (pasada hacia delante) y, para cada una de las capas, ir volviendo hacia atrás desde la salida hasta la entrada.

3. Enseñarle a la red el valor que debía predecir.

4. Modificar los pesos en cada capa para mejorar la predicción.

Lo primero que tenemos que hacer es definir las funciones de coste. Cuando estamos entrenando una red neuronal queremos tener una función de coste que optimizar, como hemos visto antes en la sección de Machine Learning. En este sentido podemos emplear varias funciones de coste en función de nuestras necesidades, pero es importante tener en cuenta que vamos a necesitar una para el proceso de aprendizaje.

En los cuatro pasos que hemos descrito, tras la inicialización de los pesos, tenemos el paso de la propagación hacia delante. Veamos este paso en

pseudocódigo:

Propagación hacia delante

Entrada: Profundidad de la red l
Entrada: Matriz de pesos para la capa i-ésima $W^{(i)}$
Entrada: Vector de sesgos de la capa i-ésima $b^{(i)}$
Entrada: Instancia a procesar x
Entrada: Salida objetivo y^*
Entrada: Función de activación g
Entrada: Función de similitud L
Entrada: Función de regularización Ω
Entrada: Parámetros del modelo θ
Entrada: Ponderación de la regularización λ

$h^{(0)} \leftarrow x$
para $k = 1, ..., l$ **hacer**
  $z^{(k)} \leftarrow W^{(k)} h^{(k-1)} + b^{(k)}$
  $h^{(k)} \leftarrow g(z^{(k)})$
fin
$y \leftarrow h^{(l)}$
$J \leftarrow L(y, y^*) + \lambda \Omega(\theta)$

Salida: Predicción hecha y
Salida: Error cometido J

Algoritmo 1: Propagación hacia delante

Como podemos ver, este algoritmo se encarga de ir pasando la información desde la entrada (la propia instancia) por cada una de las capas, multiplicando por su peso correspondiente, sumando el sesgo y aplicando la función de activación hasta que obtenemos una salida final.

Hemos visto en las secciones de Machine Learning que, una forma de hacer que nuestros algoritmos aprendan, es necesario obtener el gradiente del error para poder avanzar en el aprendizaje. Para realizar esta labor tenemos el algoritmo Backpropagation, que nos ayudará a calcular dicho gradiente de forma eficiente. Pensemos que tenemos que calcular el gradiente del modelo derivando con respecto a todos los parámetros. Cuando hablamos de Deep Learning es común tener muchas capas ocultas con un número elevado de neuronas, lo que aumenta muchísimo el número de parámetros y por tanto la complejidad del cálculo del gradiente.

Vamos a poner un ejemplo para poder ver la complejidad del cálculo del gradiente:

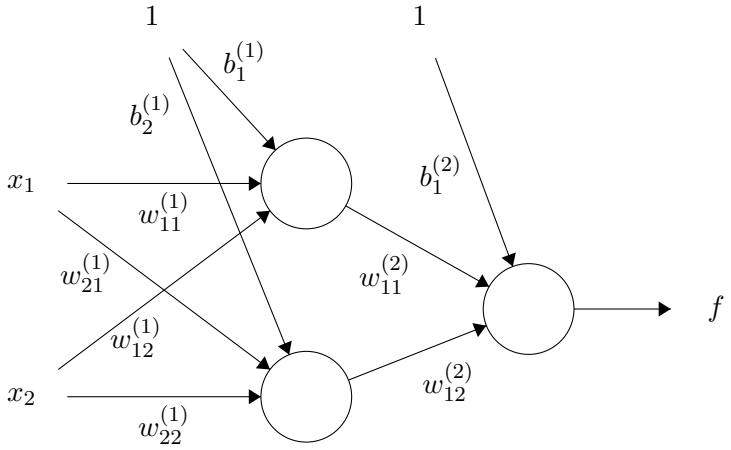

Figura 6.5: Ejemplo de una red neuronal sencilla.

Aquí podemos ver una red neuronal con 2 entradas y dos capas. Cada una de las capas tiene sus pesos y sus sesgos correspondientes. Entonces, en el modelo que hemos puesto como ejemplo, tenemos el siguiente vector de parámetros que determina nuestra red neuronal:

$$\theta = \left(w_{11}^{(1)}, w_{12}^{(1)}, w_{21}^{(1)}, w_{22}^{(1)}, b_1^{(1)}, b_2^{(1)}, w_{11}^{(2)}, w_{12}^{(2)}, b_1^{(2)}\right)$$

Con estos parámetros, la expresión de la salida de la red neuronal es:

$$f(x_1, x_2; \theta) = g(w_{11}^{(2)} g(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}) + w_{12}^{(2)} g(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}) + b_1^{(2)})$$

Para simplificar las expresiones de las parciales vamos a notar lo siguiente:

$$\alpha = g'(w_{11}^{(2)} g(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}) + w_{12}^{(2)} g(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}) + b_1^{(2)}),$$

$$\beta = g'(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)})\ y$$

$$\gamma = g'(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}).$$

Con esto, podemos sacar las parciales de f, que nos van a ser necesarias en el cálculo del gradiente del coste J.

$$\frac{\partial f}{\partial w_{11}^{(1)}}(x_1, x_2; \theta) = \alpha w_{11}^{(2)} \beta x_1, \qquad \frac{\partial f}{\partial w_{12}^{(1)}}(x_1, x_2; \theta) = \alpha w_{11}^{(2)} \beta x_2,$$
$$\frac{\partial f}{\partial w_{21}^{(1)}}(x_1, x_2; \theta) = \alpha w_{12}^{(2)} \gamma x_1, \qquad \frac{\partial f}{\partial w_{22}^{(1)}}(x_1, x_2; \theta) = \alpha w_{12}^{(2)} \gamma x_2,$$
$$\frac{\partial f}{\partial b_{1}^{(1)}}(x_1, x_2; \theta) = \alpha w_{11}^{(2)} \beta, \qquad \frac{\partial f}{\partial b_{2}^{(1)}}(x_1, x_2; \theta) = \alpha w_{12}^{(2)} \gamma,$$
$$\frac{\partial f}{\partial w_{11}^{(2)}}(x_1, x_2; \theta) = \alpha g(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}),$$
$$\frac{\partial f}{\partial w_{12}^{(2)}}(x_1, x_2; \theta) = \alpha g(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}), \qquad \frac{\partial f}{\partial b_{1}^{(2)}}(x_1, x_2; \theta) = \alpha.$$

Viendo las expresiones que tenemos de las derivadas parciales se puede entender mejor la necesidad de eficiencia. En primer lugar, podemos ver que no tenemos una red para nada grande y ya tenemos que calcular 9 derivadas parciales. Por otro lado, podemos ver que hay términos que se repiten bastante con lo que, calculando primero estos términos, simplificamos la complejidad computacional de este problema.

Veamos el algoritmo de Backpropagation en pseudocódigo:

Propagación hacia atrás

Calculamos el gradiente de la capa de salida.
$d \leftarrow \nabla_y J(y, y^*; \theta) = \nabla_y L(y, y^*)$
para $k = l, ..., 1$ **hacer**
  Aplicamos la regla de la cadena ($\odot$ es el producto componente a componente).
  $d \leftarrow \nabla_{z^{(k)}} J = d \odot g'(z^{(k)})$
  Calculamos los gradientes en los pesos y sesgos añadiendo también la regularización si la hubiera.
  $\nabla_{b^{(k)}} J = d + \lambda \nabla_{b^{(k)}} \Omega(\theta)$
  $\nabla_{W^{(k)}} J = d(h^{(k-1)})^T + \lambda \nabla_{W^{(k)}} \Omega(\theta)$
  $d \leftarrow \nabla_{h^(k-1)} J = \left(W^{(k)}\right)^T d$
fin
Salida: Valor final del gradiente d

Algoritmo 2: Propagación hacia atrás

Como podemos ver, este algoritmo simplemente aplica la regla de la cadena y va calculando los gradientes sin repetir las derivadas capa a capa. Esta aproximación hace que el cálculo sea más sencillo y no se repita.

Llegados a este punto ya tenemos la salida que nos produce nuestra red

neuronal con el algoritmo de propagación hacia delante y tenemos el cálculo del gradiente con la propagación hacia atrás. Ahora nos queda optimizar dicho coste con las herramientas que tenemos, es decir, resolver el problema de optimización que tenemos con el gradiente.

Para resolver este problema tenemos la aproximación clásica de Gradiente Descendente Estocástico aunque no es la única herramienta que podemos usar para esto. Veremos algunas de las más utilizadas y cómo nos ayudan para obtener nuevos parámetros que mejoren el desempeño de las redes.

En primer lugar cabe explicar el algoritmo más empleado en esta tarea: Gradiente Descendente Estocástico. En Deep Learning se emplea el entrenamiento por lotes o batches en inglés, lo que hace inviable el uso de ténicas como Gradiente Descendente. Es por ello que se suele emplear la aproximación estocástica de Gradiente Descendente al ir calculando una aproximación del gradiente con números pequeños de muestras. Veamos el algoritmo en pseudocódigo:

Gradiente Descendente Estocástico en la iteración k-ésima

Entrada: Tasa de Aprendizaje ϵ_k
Entrada: Parámetros iniciales θ
Entrada: Tamaño de los lotes de datos m

mientras *no se cumpla el criterio de parada* **hacer**
  Escoger un batch de datos de tamaño m $x^{(1)}, ..., x^{(m)}$ con correspondientes objetivos $y^{(1)}, ..., y^{(m)}$
  Calculamos una estimación del gradiente.
  $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  Actualizamos los parámetros.
  $\theta \leftarrow \theta - \epsilon_k \hat{g}$
fin

Salida: Nuevos parámetros θ

Algoritmo 3: Gradiente Descendente Estocástico

Como podemos ver, lo que estamos haciendo a cada paso es mejorar los parámetros del modelo en función del gradiente del coste, o más bien en este caso una aproximación del gradiente del coste.

Con esto ya si tenemos el algoritmo completo: propagación hacia delante para obtener la salida predicha y el error, propagación hacia atrás para obtener el gradiente del coste y Gradiente Descendente Estocástico (u otro método) para optimizar los parámetros del modelo (o lo que es lo mismo,

los pesos) hacia el mejor resultado.

También es común emplear variaciones de este algoritmo, pues puede mejorar con respecto a SGD aunque depende siempre del problema y los datos asociados. Las variaciones más comunes de este algoritmo son:

- SGD con momento: a veces el algoritmo de Gradiente Descendente y el algoritmo de Gradiente Descendente Estocástico presentan una oscilación alrededor del mínimo de la función que pretenden minimizar. Veamos un ejemplo de esto:

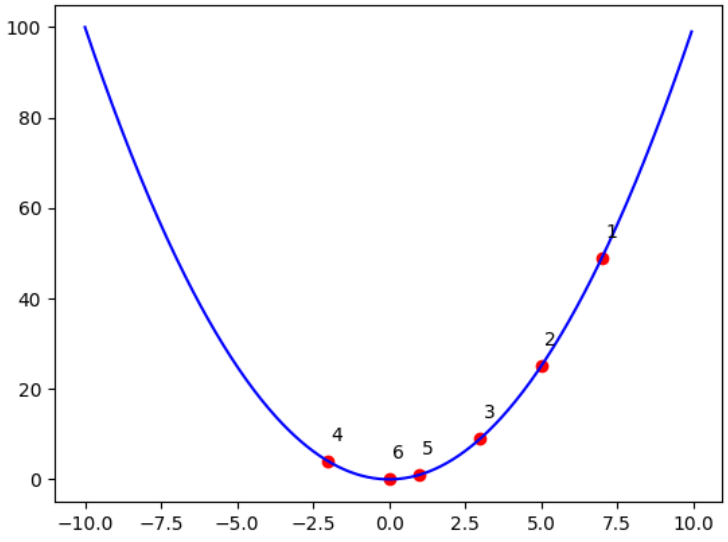

Figura 6.6: Oscilación alrededor del mínimo.

En esta figura podemos ver cómo se produce una oscilación alrededor del mínimo. En funciones más complejas esta oscilación puede ser aún mayor y condicionar el funcionamiento del algoritmo de optimización. Esta variante del algoritmo hace que este zigzagueo u oscilación sea

más leve, veamos el algoritmo en pseudocódigo:

Gradiente Descendente Estocástico con momento

Entrada: Tasa de Aprendizaje ϵ
Entrada: Parámetros iniciales θ
Entrada: Tamaño de los lotes de datos m
Entrada: Momento α
Entrada: Velocidad inicial v

mientras *no se cumpla el criterio de parada* **hacer**
  Escoger un batch de datos de tamaño m $x^{(1)}, ..., x^{(m)}$ con correspondientes objetivos $y^{(1)}, ..., y^{(m)}$
  Calculamos una estimación del gradiente.
  $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  Actualizamos la velocidad.
  $v \leftarrow \alpha v - \epsilon \hat{g}$
  Actualizamos los parámetros.
  $\theta \leftarrow \theta + v$
fin

Salida: Nuevos parámetros θ

Algoritmo 4: Gradiente Descendente Estocástico con momento

- AdaGrad: esta es una versión adaptativa de SGD. El algoritmo varía los parámetros de forma inversamente proporcional a la raíz cuadrada de la suma de los cuadrados de los valores anteriores. Es decir, actualiza la tasa de aprendizaje decrementándola más rápido cuanto mayores sean las derivadas parciales. Como consecuencia de esto el avance en

el espacio es más rápido que SGD. Veamos el pseudocódigo:

Adagrad

Notación: $\odot$ nota el producto componente a componente, $\sqrt{\cdot}$ es la raíz cuadrada componente a componente y las divisiones correspondientes son componente a componente.
Entrada: Tasa de Aprendizaje ϵ
Entrada: Parámetros iniciales θ
Entrada: Tamaño de los lotes de datos m
Entrada: Constante inicial δ pequeña

$r \leftarrow 0$
mientras *no se cumlpa el criterio de parada* **hacer**
  Escoger un batch de datos de tamaño m $x^{(1)}, ..., x^{(m)}$ con correspondientes objetivos $y^{(1)}, ..., y^{(m)}$
  Calculamos una estimación del gradiente.
  $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  $r \leftarrow r + \hat{g} \odot \hat{g}$
  $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$
  $\theta \leftarrow \theta + \Delta\theta$
fin

Salida: Nuevos parámetros θ

Algoritmo 5: Adagrad

- RMSProp: en vez de ir acumulando los gradientes se hace una media exponencial, lo que teóricamente mejor el comportamiento al minimi-

zar funciones no convexas (aquellas idóneas para SGD).

RMSProp

Notación: $\odot$ nota el producto componente a componente, $\sqrt{\cdot}$ es la raíz cuadrada componente a componente y las divisiones correspondientes son componente a componente.
Entrada: Tasa de Aprendizaje ϵ
Entrada: Parámetros iniciales θ
Entrada: Tamaño de los lotes de datos m
Entrada: Constante inicial δ pequeña
Entrada: Tasa de decaimiento ρ

$r \leftarrow 0$
mientras *no se cumlpa el criterio de parada* **hacer**
  Escoger un batch de datos de tamaño m $x^{(1)}, ..., x^{(m)}$ con correspondientes objetivos $y^{(1)}, ..., y^{(m)}$
  Calculamos una estimación del gradiente.
  $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  $r \leftarrow \rho r + (1 - \rho)\hat{g} \odot \hat{g}$
  $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$
  $\theta \leftarrow \theta + \Delta\theta$
fin

Salida: Nuevos parámetros θ

Algoritmo 6: RMSProp

- Adam: es una modificación de RMSProp con momento. Veamos el

pseudocódigo:

RMSProp

Notación: $\odot$ nota el producto componente a componente, $\sqrt{\cdot}$ es la raíz cuadrada componente a componente y las divisiones correspondientes son componente a componente.
Entrada: Tasa de Aprendizaje ϵ
Entrada: Parámetros iniciales θ
Entrada: Tamaño de los lotes de datos m
Entrada: Constante inicial δ pequeña
Entrada: Tasas de decaimiento $\rho_1, \rho_2 \in [0, 1)$

$s \leftarrow 0$
$r \leftarrow 0$
$t \leftarrow 0$
mientras *no se cumlpa el criterio de parada* **hacer**
  Escoger un batch de datos de tamaño m $x^{(1)}, ..., x^{(m)}$ con correspondientes objetivos $y^{(1)}, ..., y^{(m)}$
  Calculamos una estimación del gradiente.
  $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  $t \leftarrow t + 1$
  $s \leftarrow \rho_1 \cdot s + (1 - \rho_1)\hat{g}$
  $r \leftarrow \rho_2 \cdot s + (1 - \rho_2)\hat{g} \odot \hat{g}$
  $\hat{s} \leftarrow \frac{s}{1-\rho_1^r}$
  $\hat{r} \leftarrow \frac{r}{1-\rho_2^r}$
  $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{\hat{r}}}\hat{s}$
  $\theta \leftarrow \theta + \Delta\theta$
fin

Salida: Nuevos parámetros θ

Algoritmo 7: RMSProp

Con esto ya hemos cubierto cómo aprende una red neuronal de forma completa.

6.2. Capas empleadas

6.2.1. Capas densas o totalmente conectadas

En las redes neuronales tenemos distintos tipos de capas o neuronas que podemos utilizar en la construcción de la arquitectura de la red. La estruc-

tura explicada con anterioridad asumía el uso de capas densas o totalmente conectadas. Estas capas tienen exactamente el comportamiento descrito en la figura 6.1. Debemos tener en cuenta que vamos a emplear un tipo de capa u otro en función de la información que queramos obtener de nuestros datos.

En primer lugar vamos a hacer un pequeño repaso de las capas densas o totalmente conectadas. Estas capas están ya descritas en la sección anterior, por lo que no nos vamos a detener más en esto. Las capas densas son empleadas cuando tenemos entradas de tipo numérico, ya sea una única entrada numérica o un vector de números.

Lo más común es que esta capa siempre aparezca en la mayoría de estructuras de redes neuronales, pues aunque sea una red por ejemplo con mayoría de capas convolucionales, normalmente las últimas o las primeras son capas densas que nos permiten igualar la dimensión de los datos a lo que nosotros queramos (en función de si es un problema de predicción, regresión o clasificación u otro tipo).

El funcionamiento de este tipo de capas es muy sencillo. Tenemos una neurona que recibe tantas conexiones como neuronas o entradas hubiera en la capa anterior más un sesgo. Todas estas conexiones tienen un peso asignado como vimos en el ejemplo **??**. Este comportamiento se replica en toda la capa, es decir, todas las neuronas de la capa reciben como entrada todas estas conexiones con un valor numérico y un peso asignado. Estos pesos y valores se combinan de forma lineal, es decir, multiplicando cada valor numérico por cada uno de los pesos y sumándose. Tras la suma simplemente se añade el término de sesgo y se aplica a todo esto la función de activación, que es intrínseca de la capa. Esto quiere decir que, para que el funcionamiento de la capa sea homogéneo, se aplica la misma función de activación a todas las neuronas que componen dicha capa. Esto lo que nos va a producir como salida es un vector de valores numéricos, que son las salidas de cada una de las neuronas de nuestra capa.

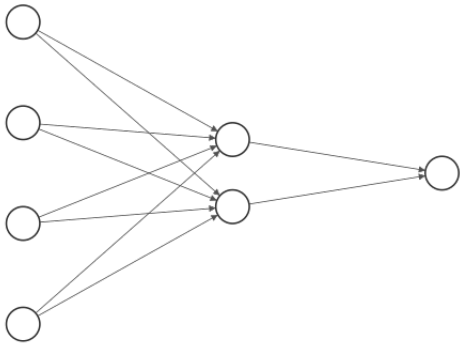

Figura 6.7: Ejemplo de red neuronal con capas densas.

En este ejemplo podemos ver que tenemos 3 capas, una capa de entrada, una capa oculta y una de salida. La capa de entrada tiene 4 neuronas, por lo que recibe como entrada un vector numérico de 4 valores que se asocian cada uno con una neurona. La capa oculta tiene 2 neuronas que reciben como entrada cada una un vector numérico de 4 valores, produciendo dicha capa al final un vector de dos valores numéricos. Por último la capa de salida tiene una única neurona, por lo que la salida de la red es un único valor numérico, recibiendo como entrada un vector de dos valores.

Esta capa nos servirá en todas las estructuras Deep Learning elaboradas que emplearemos más tarde en la aplicación práctica.

6.2.2. Capas convolucionales

En esta sección vamos a explicar el funcionamiento de las capas de convolución, los datos que reciben y los datos que obtenemos a través de la operación de convolución.

Lo primero que tenemos que hacer es definir el operador de convolución para poder ver cómo y donde emplearlo en una red neuronal. Supongamos que tenemos una serie de datos que dependen de una variable t temporal, es decir, por ejemplo podemos tener distintos valores de una serie temporal. Si la función que nos da dichos valores de la serie temporal es $x(t)$, podemos definir una función que suavice a $x(t)$ de la siguiente forma:

$$s(t) = \int x(a) \cdot w(t-a)da = (x * w)(t)$$

Donde $w(a)$ es una ponderación que hacemos a los valores de la función

$x(t)$. Además, se suele notar esta operación con el asterisco como hemos hecho. Esta función suavizada de la original es lo que llamamos operación de convolución.

En un caso real de aplicación no vamos a tener la función $x(t)$ que nos da la salida u objetivo de nuestro problema en cada instante de tiempo, pues entonces no habría problema al estar resuelto y modelado. Por contra, lo que vamos a tener normalmente es una serie de valores de ejemplo de salida. Al ser este un número finito de muestras, sabemos que la integral se define como una suma y por tanto podemos definir el operador de convolución como:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Está claro que esta definición que hemos dado no tiene sentido cuando tenemos más de un valor, es decir, cuando la función x no es real, si no que da como salida un vector de números.

Normalmente empleamos como entrada en este tipo de redes con capas convolucionales un array de datos, por ejemplo, una imagen o varias instancias de una serie temporal. Esto hace que tengamos como input una matriz bidimensional o tridimensional normalmente como entrada, por lo que tenemos que pensar en hacer la operación de convolución en varios ejes. La aplicación de esta operación nos va a dar como resultado de nuevo una matriz, como es natural. Por tanto, suponiendo que I es nuestra imagen de entrada o grupo de datos de una serie temporal, podemos definir el operador de convolución sobre dos ejes como:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Donde K es un núcleo o kernel de dos dimensiones, como la entrada, pero no tiene por qué tener el mismo tamaño que la misma. Como característica de esta operación tenemos que es conmutativa, es decir, $(I * K)(i,j) = (K * I)(i,j)$ al invertirse únicamente el producto que tenemos dentro de la sumatoria.

Veamos un ejemplo de como aplicamos esta operación de convolución a una matriz de dos dimensiones $3x4$ con un kernel $2x2$:

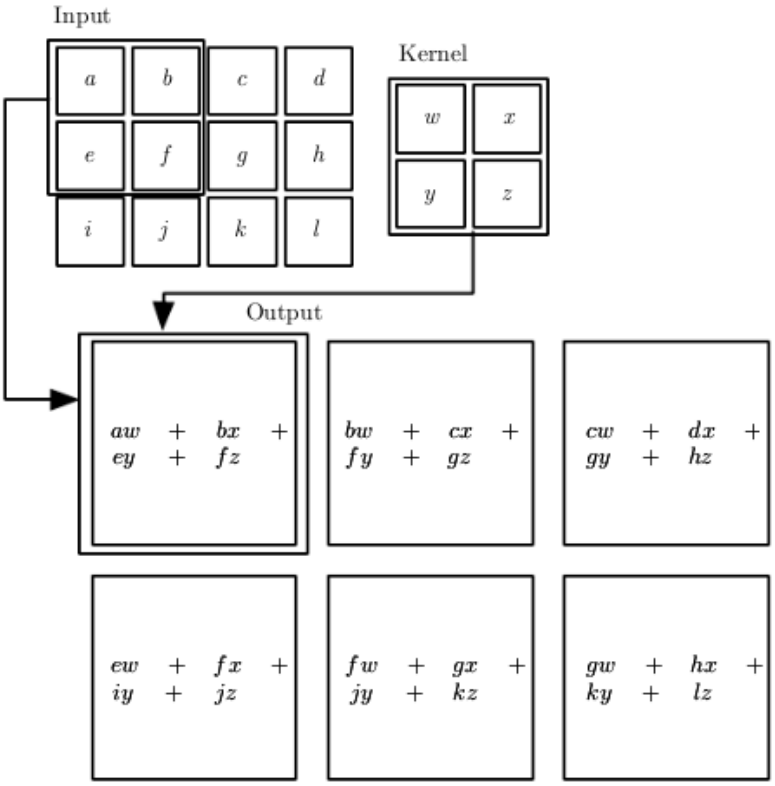

Figura 6.8: Ejemplo de operación de convolución.

Como podemos ver, el resultado de esta operación es tomar el kernel e ir deslizándolo por la matriz para obtener, por cada una de estas posiciones, un valor real y construir de esta forma una matriz de salida. En este caso, con un kernel $2x2$ y con una matriz de entrada $3x4$ obtenemos una matriz de salida $2x3$.

Precisamente en este tipo de capas, los pesos son los núcleos que aplicamos en la operación de convolución, por lo que nuestro algoritmo de backpropagation se orientará en optimizar dichos valores.

Normalmente, una capa de convolución dentro de una red neuronal hace esta operación de convolución con varios filtros o kernels, por ejemplo 32, 64, 128 u otro número. Si elegimos tamaño de kernel $3x3$ entonces tendríamos 32, 64, 128 u otro número de matrices con 9 pesos en cada una que entrenaremos con backpropagation.

Con esta capa de convolución se suele combinar una capa u operador de Pooling o agrupación. Este operador nos va a servir para agrupar los datos

y reducir la dimensión. El operador de agrupación se aplica a cada una de las salidas de la convolución, es decir, al resultado de hacer $(I * K)$ con cada uno de los núcleos. Podemos aplicar aquí por ejemplo un agrupamiento $2x2$ que nos reducirá la dimensión de la salida a la mitad para cada núcleo.

Esta operación, a parte de reducir la dimensión de la salida, nos va a ayudar a que nuestra salida sea más robusta. Pensemos por ejemplo en que tenemos imágenes como datos de entrada, si las imágenes de entrada varían ligeramente (una ligera traslación) entonces la operación de Pooling nos va a ayudar a que la información que extraemos sea invariante frente a estas pequeñas variaciones.

Las dos operaciones de agrupamiento que se suelen emplear en la capa de Pooling son el máximo y la media. Esto lo que hace es obtener como valor final la media o el máximo del tamaño de Pooling que estemos haciendo. Pensemos por ejemplo que la salida de la convolución es una matriz $8x8$ y nosotros hacemos un agrupamiento $2x2$. Entonces vamos a ir pasando por la salida una matriz $2x2$ y calculando la media o el máximo de dicha matriz para obtener un único valor, después desplazamos uno a la derecha y hacia abajo una unidad también cuando acabemos la fila completa. De esta forma iremos construyendo una matriz con los máximos o medias de la salida de la convolución.

Ya que entendemos un poco mejor cómo funciona la operación de convolución y cómo funcionan las capas convolucionales y de agrupamiento, vamos a repasar brevemente el objetivo y el uso de estas capas en la redes neuronales.

Para poder aplicar la operación de convolución de forma coherente debemos tener datos que tengan una dependencia local, por ejemplo las imágenes o una serie temporal sobre la que agrupamos las instancias consecutivas en el tiempo para formar un vector (en el caso de una serie temporal real) o una matriz (en el caso de una serie temporal de varias variables de salida).

Con la operación de convolución, al ir repitiendo esta operación varias veces junto con el agrupamiento, obtenemos información local interesante e importante para nuestro problema. Por ejemplo sería algo común que, tras la aplicación de varias capas de convolución, obtuviéramos como salida de esos parches formas, objetos o secciones de imágenes que sean de relevancia para nuestra tarea final. Esto ocurre de igual forma con las series temporales u otro tipo de datos, pero es más sencillo explicar este fenómeno con las imágenes por la capacidad de visualizar los datos.

Por tanto estas capas no están pensadas para ser la última o primera capa de nuestra red, si no para formar parte de ella para obtener características

de nuestros datos que nos ayuden en la tarea de predicción, regresión o clasificación que tengamos como objetivo.

6.2.3. Capas recurrentes y LSTM

Este tipo de redes y capas están pensadas precisamente para recibir como entrada varias instancias consecutivas en el tiempo de una serie temporal. Para poder hacer una red con un mejor desempeño en este tipo de datos, vamos a hacer que los pesos de una capa en un momento determinado de la serie temporal, puedan afectar a momentos posteriores y capas posteriores dependiendo de la estructura de la red. Con ello, la intención es poder sacar patrones que no dependan del tiempo y sean poco sensibles a variaciones como reflexiones de los datos (cambiar inicio por fin y fin por inicio) y procesar datos de distinto tamaño.

En este tipo de datos siempre tenemos una dependencia de los datos con los anteriores en el tiempo, es decir, podemos describir el dato en el instante t de tiempo a partir de los datos en los instantes anteriores. Supongamos que tenemos representado el estado del sistema como $s^{(t)}$ entonces esto se traduce en que:

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

es decir, el valor en el instante t depende del $t-1$ y así sucesivamente. En una aplicación real de este tipo de redes, fijamos un número de pasos en el tiempo que vamos a analizar en bloque. Por tanto este proceso recurrente se puede desarrolla de forma extensiva. Si fuese con 3 pasos en el tiempo tendríamos que $s^{(3)}$ se podría expresar como:

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

Ahora vamos a hacer este ejemplo algo más real. Supongamos que tenemos nuestro sistema $h(t)$ (se nota con h porque será oculto dentro de la red, hidden) y una señal externa (la serie temporal) entonces podemos describir nuestro sistema dinámico recurrente como:

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

Esta función que modela nuestro sistema con el estado del mismo y con una señal externa la podemos desenrollar, es decir, eliminar la componente

recurrente al estar basado en un número finito de muestras como hemos hecho anteriormente.

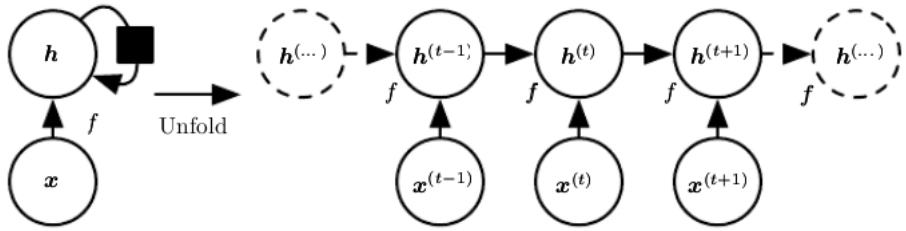

Figura 6.9: Desenrollado de la función recurrente.

Como podemos ver en el esquema, el estado anterior del sistema influye sobre el estado actual y la señal exterior actual influye también en el estado del sistema.

Con esta idea de compartir pesos y que el estado actual vaya dependiendo de los anteriores, llevando una dependencia temporal, podemos elaborar distintos tipos de redes neuronales.

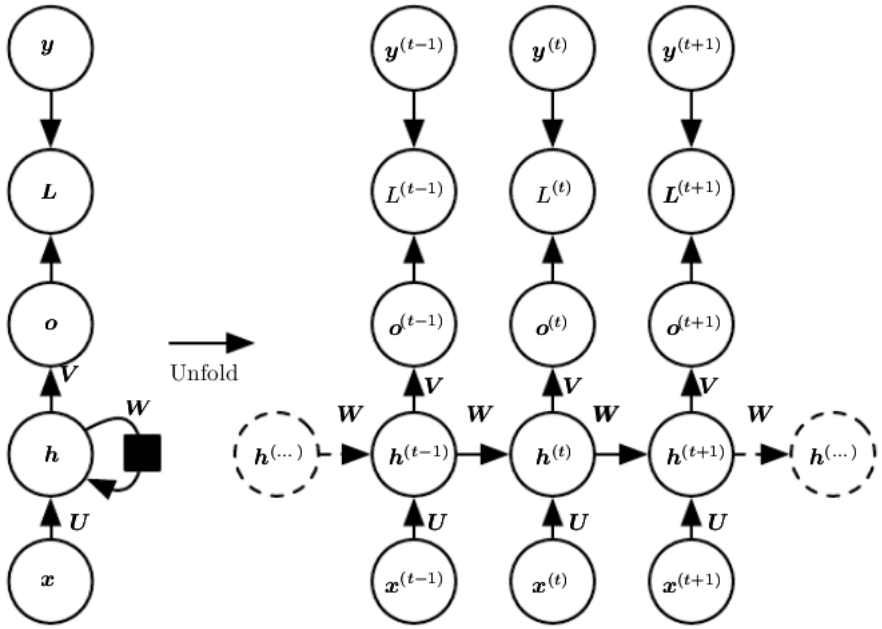

Figura 6.10: Red recurrente que produce una salida en cada paso temporal con conexiones entre las unidades ocultas.

En este caso vemos un esquema, en el que tenemos las entradas o estímulos externos que describíamos anteriormente, el estado interno del sistema representado con h, la salida producida o, la función de pérdida de la salida predicha frente a la real L y la salida real y.

En este tipo de red neuronal tenemos la entrada ponderada por unos pesos U, el estado interno ponderado por pesos W y la salida ponderada por pesos V. Podemos ver que esta estructura nos da, para cada unidad de tiempo, su salida correspondiente y como podemos ver, los pesos de la capa oculta solo se usan de un instante de tiempo a otro de la misma capa oculta o estado del sistema.

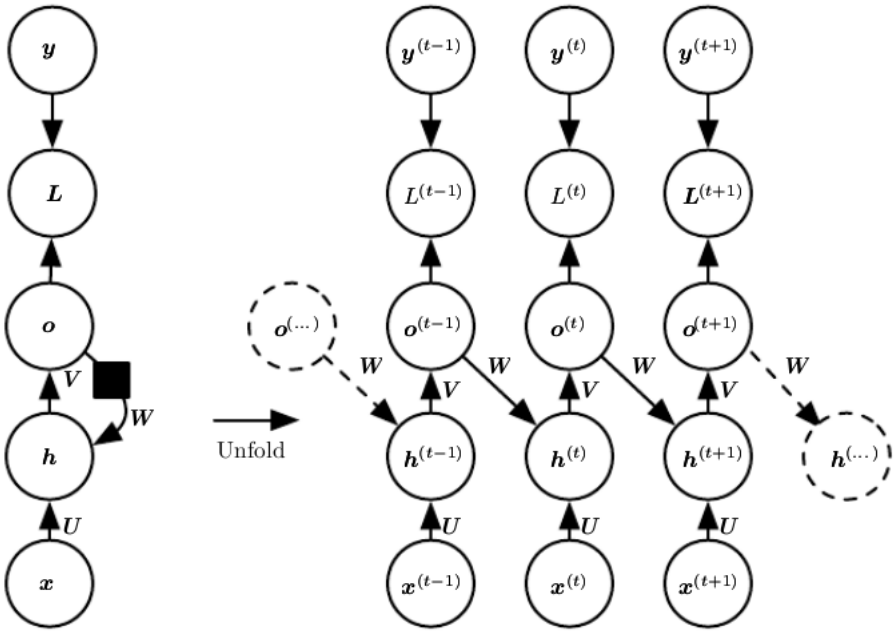

Figura 6.11: Red recurrente que produce una salida en cada paso temporal con conexiones de la salida a las unidades ocultas.

En este caso podemos ver como no hay conexiones de pesos entre las neuronas ocultas, si no de la salida anterior a la neurona del estado del sistema en el instante de tiempo siguiente. Esto nos está dando información de la salida anterior para condicionar la siguiente salida, de hecho es la única información que recibe el siguiente estado de tiempo de la red neuronal además de la señal externa.

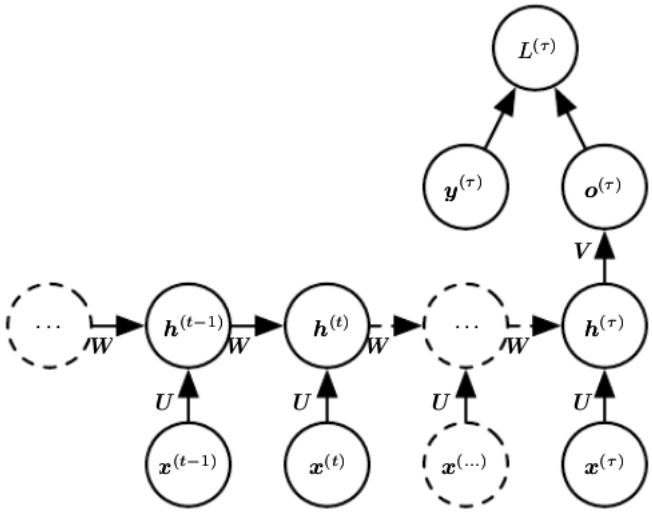

Figura 6.12: Red recurrente que produce una única salida con conexiones entre las unidades ocultas.

Como podemos ver, en este caso tenemos que los pesos se comparten sólo en la capa oculta. Esta red va almacenando conocimiento hasta llegar al último paso temporal, en el cual se emplea lo obtenido de los instantes de tiempo anteriores para producir una única salida. Este tipo de redes pueden usarse por ejemplo para predecir el siguiente valor de una serie temporal conociendo los valores de los instantes de tiempo anteriores.

Estos ejemplos que hemos dado no son una taxonomía, si no una serie de modelos que podemos utilizar y empleamos como ejemplo. Cada problema tiene unos requerimientos y objetivos que queremos cumplir y debemos ajustar nuestra red al problema que tengamos en consideración.

Hemos hecho un breve repaso de la idea de las redes neuronales recurrentes. En nuestro caso de aplicación práctica hemos empleado capas LSTM o Long-Short Term Memory, por lo que vamos a repasar cómo funcionan estas capas de forma teórica.

Las capas LSTM entran dentro de un tipo de redes neuronales recurrentes llamadas redes recurrentes con puertas. La idea es que estas redes elaboran caminos entre las entradas, salidas y estados internos que permiten recordar información u olvidarla si ya no nos es útil. Veamos esto con un esquema de cómo funcionan las LSTM:

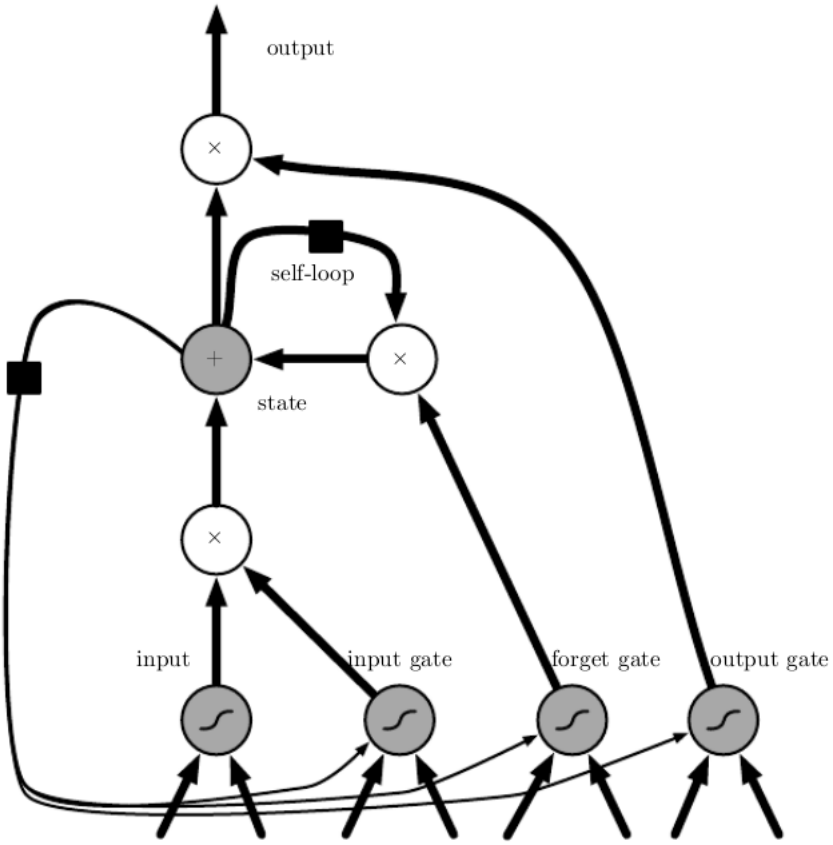

Figura 6.13: Esquema de funcionamiento de una celda LSTM.

Estas celdas o neuronas están conectadas entre sí a través de los estados del sistema como en una red recurrente normal, por lo que cada uno de los estados tendría un peso y una conexión con la siguiente neurona. Podemos ver que la entrada se procesa con una neurona tradicional como teníamos en las capas densas, llegando esta entrada al estado de la celda y siendo acumulado o añadido si la función de activación y el resto de la celda lo permite. Podemos ver que el estado tiene un ciclo consigo mismo, moderado por la puerta de olvido. Esta puerta puede permitir olvidar el estado si se considera que es más beneficioso que conservarlo. El estado se realimenta de nuevo a la entrada tanto a la unidad de entrada, como a la de olvido y salida para poder tener su propio estado anterior como entrada del siguiente y decidir su acción. La neurona de salida puede cortar la salida producida por la celda.

Todo este esquema permite que la celda adquiera el estado de la anterior y lo integre como una entrada, procese su propia entrada basándose también

en su estado anterior y decida si acumula el conocimiento, olvida o corta su salida en función de lo que mejor función de pérdida otorgue. Todo esto hace que nuestra celda pueda recordar todo el contenido de la serie temporal u olvidarlo en algún punto determinado u obviar una entrada si no se percibe de interés.

Este tipo de redes son actualmente muy exitosas en aplicación sobre texto, predicciones en series temporales y uso en vídeo para detección de trayectorias entre otros ejemplos. Nosotros las emplearemos en la parte práctica tanto solas como combinadas con convoluciones para obtener características algo mejores para alimentar la red.

6.3. Autoencoders

En la sección práctica veremos el uso de Autoencoders para la tarea de detección de anomalías que queremos desarrollar, por lo que necesitamos un poco de fundamento teórico para poder entender bien el funcionamiento de este tipo de arquitecturas.

El objetivo principal de estas arquitecturas es conseguir una codificación de los datos, es decir, una representación de los mismos en un espacio de menor dimensión que permita obtener suficiente información de ellos como para reconstruirlos a los originales cometiendo un error pequeño asumible. Esta arquitectura se puede ver dividida por tanto en dos partes, la función codificadora que codifica el dato y la función decodificadora que reconstruye el dato a partir de la codificación.

Normalmente no estamos interesados en la salida del Autoencoder, es decir en la decodificación del dato, si no en la codificación pero en nuestro caso estaremos interesados en la reconstrucción. Aunque nuestro objetivo sea obtener una salida final de calidad es fundamental fijarse en la codificación obtenida y que esta sea buena.

El tipo de Autoencoders que vamos a emplear son denominados como Autoencoders incompletos, ya que el objetivo es que la codificación sea de menor dimensión que la que poseen los datos originales. El proceso de aprendizaje de estos Autoencoders es sencillo: introducimos como entrada los datos originales y como salida predicha los mismos datos de entrada para que la red pueda aprender a reconstruirlos.

Para el proceso de detección de anomalías vamos a usar esta arquitectura del siguiente modo: entrenaremos con datos limpios sin anomalías e intentaremos que la reconstrucción sea lo más ajustada posible pero siem-

pre con una codificación del menor tamaño posible para forzar la extracción de las características meramente esenciales. Tras esto lo que haremos será predecir, cuando obtengamos una reconstrucción con poco error estaremos ante un dato no anómalo o parecido a los no anómalos vistos previamente por la red y por tanto es capaz de reconstruirlos con poco error. Por contra si no es capaz de reconstruirlos bien y se comete mucho error estaremos ante un dato que no es normal y por tanto no ha sido visto por la red neuronal, cometiendo un mayor error de reconstrucción.

Veamos un ejemplo de un autoencoder de capas totalmente conectadas:

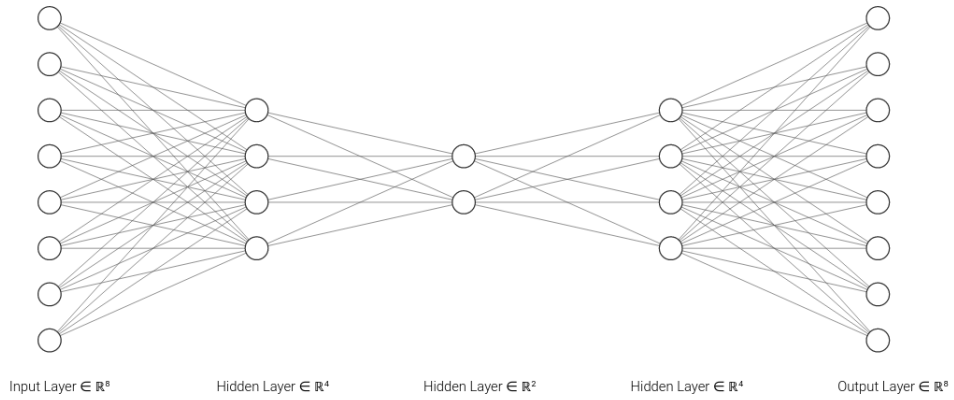

Figura 6.14: Ejemplo de Autoencoder con capas totalmente conectadas.

Como podemos observar en la figura 6.14 tenemos datos de entrada con 8 variables, reducimos primero la dimensión a 4 y finalmente la codificación reduce la dimensión a dos variables. Tras esto viene la decodificación que devuelve los datos primero a 4 variables y finalmente a las 8 iniciales.

Esta misma arquitectura se puede emplear con varias capas distintas, en la práctica nosotros emplearemos capas densas y capas LSTM para extraer una codificación de menor dimensionalidad que la original.

Bibliografía

[1] Abu-Mostafa Yaser, Magdon-Ismail Malik, and Lin Hsuan-Tien. *Learning from Data: a short course*.

[2] Vladimir Cherkassky and Filip M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons. 02476.

[3] Charu C. Aggarwal. *Outlier Analysis*. Springer-Verlag.

[4] Carreño Ander, Inza Iñaki, and Lozano Jose A. Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework.

[5] Vapnik V. *The Nature of Statistical Learning Theory*. New York: Springer. 00049.

[6] M. Loève. *Probability Theory*. Springer-Verlag. 00032.

[7] Fabian Keller, Emmanuel Müller, and Klemens Böhm. HiCS: High constrast subspaces for density-based outlier ranking. page 12. 00000.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press.

[9] Zaccone Giancarlo, Karim Md. Rezaul, and Menshawy Ahmed. *Deep Learning with TensorFlow*. Packt.

[10] Juanjo Nieto and Antonia Delgado. Apuntes modelos matemáticos 2. 00000.

[11] Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 212–221. IEEE.

[12] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 37–46. ACM. event-place: Santa Barbara, California, USA.

[13] Hastie T., R. Tibshirani, and J. Friedman. *The elements of Statistical Learning: Data Mining Inference and Prediction.* New York: Springer. 44014.

[14] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A python toolbox for scalable outlier detection.

[15] Ignacio Aguilera Martos. Detección de anomalías basada en técnicas de ensembles.

[16] Charte David, Charte Francisco, García Salvador, del Jesus María J.º, and Herrera Francisco. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines.

[17] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. pages 6231–6239.