



Analítica aplicada a contabilidad empresarial

Trabajo de Fin de Grado

Convocatoria: 10 de junio de 2024

Alumno/a: Javier Coque Fernández

Tutor/a: Dr. D. Pablo Ramos Criado

Grado: Ingeniería de software

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor: Pablo Ramos. Gracias por su seguimiento en el trabajo, sus propuestas, sugerencias y correcciones. Ha sido un mentor excepcional.

Aprovecho que este es mi último trabajo de carrera para agradecer a todo el profesorado; con especial mención a Víctor Gayoso y Mar Angulo.

Es imposible escribir esta sección sin mencionar a mi compañero de carrera David García Lleyda, quien me ha ayudado y apoyado durante toda la carrera. Gracias también al resto de mis compañeros de carrera. Todos ellos, de alguna manera u otra, también han aportado de manera directa a este trabajo.

Por último, agradecer a quienes siempre han estado y siempre estarán a mi lado: mis padres. Se lo merecen todo.

Tabla de contenidos

1	Introducción	13
1.1	Justificación del contexto	14
1.2	Planteamiento del problema	15
1.3	Objetivos del trabajo	16
2	Estado de la cuestión	17
2.1	Marco teórico del trabajo	17
2.1.1	Contabilidad	17
2.1.1.1	Cuentas Activo, Pasivo y Patrimonio Neto	18
2.1.1.2	Cuentas de compras / gastos y ventas / ingresos	19
2.1.1.3	Plan General Contable	20
2.1.1.4	Libro Mayor	21
2.1.1.5	Libro Diario y asientos contables	23
2.1.2	Business Intelligence & Business Analytics	25
2.1.2.1	Series temporales	25
2.1.2.2	Servicios en la nube	28
2.1.2.3	Visualización de datos	29
2.1.3	Aprendizaje automático	30
2.2	Trabajos relacionados	47
2.2.1	Aplicación de Business Intelligence en una pequeña empresa mediante el uso de Power BI	47
2.2.2	Transformers in Time Series: A Survey	48
3	Aspectos metodológicos	49
3.1	Metodología	49
3.1.1	MLOps	49
3.2	Técnicas aplicadas	50
3.3	Tecnologías aplicadas	51
3.3.1	Entorno y librerías	51
3.3.2	Herramientas Software empleadas	51
3.3.2.1	Git y Github	52

3.3.2.2	Mlflow	52
3.3.2.3	Dagshub	52
3.3.2.4	Prefect	52
3.3.2.5	Power BI	53
3.3.2.6	Azure	53
4	Desarrollo del trabajo	54
4.1	Preprocesamiento de los datos	55
4.2	Procesamiento de los datos: Creación de modelos	58
4.2.1	Creación de modelos	59
4.2.1.1	Técnicas de evaluación	59
4.2.1.2	Optimización de hiperparámetros	60
4.2.1.3	Obtención de mejores resultados	60
4.3	MLOps: monitorización y automatización del proceso	61
4.3.1	Monitorización de los modelos	61
4.3.2	Automatización del proceso	65
4.4	Comparativa de modelos	69
4.4.1	Compras	70
4.4.2	Ventas	72
4.4.3	Datos finales	74
4.5	Almacenamiento de los datos	74
4.5.1	SQL Server	78
4.6	Visualización de los datos	79
4.6.1	Actualización incremental	80
4.6.2	Resultado final	82
4.6.3	Publicación del dashboard en la nube	86
5	Conclusiones	88
6	Apéndice	90
6.1	Procedimiento especulativo de cuenta divisionaria	90
6.2	Plan General Contable español: cuentas más importantes	91
6.3	Predicción intrapolar vs. extrapolar	92
6.4	Gradient Boosting	92

6.4.1	Descenso del gradiente	92
6.4.2	Regularización L1 y L2	101
6.5	Desvanecimiento del gradiente y gradiente explosivo	105
6.6	Transformers: origen y explicación	105
6.6.1	Embedding y codificador posicional	108
6.6.2	Encoder	110
6.6.2.1	Atención Encoder	110
6.6.2.2	Bloque residual	115
6.6.2.3	Red de retroalimentación	116
6.6.3	Decoder	116
6.6.3.1	Outputs	117
6.6.3.2	Enmascaramiento de atención	119
6.6.3.3	Atención Decoder	120
6.6.4	Salida de Transformer	121
6.7	Informer	122
6.7.1	Codificación Informer	122
6.7.2	Algoritmo ProbSparse	123
6.7.3	Comparativa Informer	125
Referencias		126

Listas de Figuras

2.1	Ejemplo balance general. $LP = \text{largo plazo}$, $CP = \text{corto plazo}$	19
2.2	Secciones del Plan General Contable español. Fuente: https://www.plangeneralcontable.com	21
2.3	Estructura de una cuenta en el Libro Mayor.	22
2.4	Ejemplo cuentas activo y pasivo Libro Mayor.	22
2.5	Ejemplo cuentas compra/gasto y venta/ingreso Libro Mayor.	23
2.6	Estructura de un asiento contable.	24
2.7	Ejemplo transacción básica Libro Diario y Libro Mayor.	24
2.8	Ejemplo de serie temporal con tendencia ascendente.	26
2.9	Ejemplo de serie temporal con estacionalidad.	26
2.10	Ejemplo del componente cíclico.	27
2.11	Ruido.	27
2.12	Componentes de una serie temporal por separado.	28
2.13	Ejemplo sencillo aplicando árboles de decisión para un problema de regresión.	32
2.14	Algoritmo de GBM de Friedman. Fuente: (Natekin & Knoll, 2013)	33
2.15	Ejemplo de dos modelos simples de árboles de decisión. Estos modelos se añaden para generar un resultado final. Fuente: (Chen & Guestrin, 2016)	34
2.16	Relación entre inteligencia artificial, aprendizaje automático y aprendizaje profundo.	37
2.17	Ejemplo arquitectura de un perceptrón multi-capa con dos capas ocultas.	38
2.18	Mientras que la atención del Transformer original puede asignar una alta dependencia entre los dos datos señalados en la figura de la izquierda, una atención local puede percibir mejor las formas de la serie temporal (figura de la derecha). Fuente: (Li et al., 2019)	41
2.19	Arquitectura Informer. Fuente: (H. Zhou et al., 2021)	42
2.20	Input y tres primeras capas del Encoder en el Informer. Nótese cómo se va reduciendo la dimensionalidad (L) de las capas de atención (rojo) a la mitad. Esto se consigue con un MaxPooling de 2×2 . Fuente: (H. Zhou et al., 2021)	43
2.21	Ejemplo entre una secuencia input y cómo se obtiene la secuencia de output a través de la operación de convolución.	44
2.22	Convolución con zero-padding. Se obvia la convolución para hacerlo más claro.	45
2.23	Ejemplo de inviabilidad al aumentar capas.	45
2.24	Dilatación. Fuente: (Oord et al., 2016)	46

2.25 Bloque Residual TCN. Fuente: (Bai et al., 2018)	46
4.1 Arquitectura software del proyecto.	54
4.2 Carpeta tfg_module (izquierda) junto a su contenido (derecha).	56
4.3 Ejemplo abstracción código con tfg_module.	57
4.4 Importación tfg_module.	58
4.5 Procesamiento de los datos y creación de los modelos.	58
4.6 División entrenamiento TCN.	59
4.7 División entrenamiento XGBoost, LightGBM e Informer.	59
4.8 Mlflow alojado en Dagshub.	61
4.9 Proceso de vinculación entre el repositorio de Github con el Dagshub.	62
4.10 Comandos para conectarse a Dagshub y alojar modelos en Mlflow	63
4.11 Ejemplo de modelos almacenados (best_MAE y best_RMSE) para el Experimento “Compras LightGBM”.	64
4.12 Modelo almacenado en Mlflow, junto con la gráfica, datos de predicciones y el respectivo link para recuperarlo.	64
4.13 Prefect en la arquitectura del proyecto.	65
4.14 Decoradores de Prefect en el código.	66
4.15 Flow ejecutado con Prefect.	67
4.16 Conceptos deployment, work-pool y worker de manera visual.	68
4.17 Datos de testeo junto con las predicciones de cada modelo de Compras visualizado de una manera gráfica.	71
4.18 Datos de testeo junto con las predicciones de cada modelo de Ventas visualizado de una manera gráfica.	73
4.19 Arquitectura Azure.	74
4.20 Contenido de Azure Blob Storage.	75
4.21 Arquitectura pura de Azure.	76
4.22 Configuración <i>Datasets</i>	76
4.23 Mappeo entre origen y destino.	77
4.24 Configuración de upsert con su respectiva columna ID.	77
4.25 Esquema de la Copy Activity realizada.	78
4.26 SQL Server endpoint.	78
4.27 Última sección de la arquitectura del proyecto: Power BI.	79

4.28 Esquema relacional Power BI.	80
4.29 Política de configuración de actualización incremental.	81
4.30 Pestaña General dashboard.	82
4.31 Pestaña Cuentas dashboard.	83
4.32 Pestaña Análisis dashboard.	84
4.33 Pestaña Predicciones dashboard.	85
4.34 Resultado final del proyecto en teléfono móvil personal.	86
4.35 Gateways creadas en el servidor de Power BI.	87
6.1 Ejemplo predicción intrapolar.	92
6.2 Problema regresión lineal.	93
6.3 Recta de regresión para distintos valores del parámetro θ	94
6.4 Función de coste para distintos valores de θ	95
6.5 Coste asociado a $\theta = 9$	95
6.6 Gradiente.	96
6.7 Ejemplo de mala práctica: Learning rate α	96
6.8 Ejemplo de mala práctica: Learning rate α demasiado bajo.	97
6.9 Ejemplo de un learning rate adecuado.	98
6.10 Solución problema recta de regresión.	98
6.11 Ejemplo de función no-convexa con las flechas indicando supuestos valores iniciales aleatorios de θ	100
6.12 Six-Hump Camel function.	101
6.13 Error-Varianza. Fuente: https://aprendeia.com/bias-y-varianza-en-machine-learning/	102
6.14 Gráficas $ x $ y x^2	104
6.15 Palabras representadas en vectores de tres dimensiones. Cada punto es el final del vector, i.e. cada palabra. Fuente: https://projector.tensorflow.org/	106
6.16 Ejemplo, retención de memoria en texto. La palabra <i>su</i> hace referencia a una palabra muy anterior a ella (<i>chico</i>).	106
6.17 Arquitectura del Transformer original. Fuente: (Vaswani et al., 2017)	107
6.18 Parte izquierda: embedding y encoding posicional. Fuente: (Vaswani et al., 2017)	108
6.19 Atención múltiple. Fuente: (Vaswani et al., 2017)	110
6.20 Ejemplo de <i>self-attention</i> entre todas las palabras de una secuencia. Cuanto más oscuro es el color, más atención le presta una palabra a la correspondiente. Fuente: (Vaswani et al., 2017)	112

6.21 Visualización entre ángulo de vectores parejos (izquierda) y vectores disparejos (derecha).	112
6.22 Ejemplo de atención múltiple.	114
6.23 Atención llevada a más de una dimensionalidad. Fuente: (Vaswani et al., 2017)	114
6.24 Matriz final de atención.	115
6.25 Ajuste de valores. Fuente: (Vaswani et al., 2017)	115
6.26 FFN. Fuente: (Vaswani et al., 2017)	116
6.27 Outputs. Fuente: (Vaswani et al., 2017)	117
6.28 Concatenación Decoder.	118
6.29 Subcapa de enmascaramiento. Fuente: (Vaswani et al., 2017)	119
6.30 Atención Decoder. Fuente: (Vaswani et al., 2017)	120
6.31 Salida de Transformer. Fuente: (Vaswani et al., 2017)	121
6.32 Codificación completa del Informer. Fuente: (H. Zhou et al., 2021)	122
6.33 Ejemplo matriz de atención. Fuente: (Bahdanau et al., 2014)	123
6.34 Distribución de las atenciones con forma de cola larga. Fuente: (H. Zhou et al., 2021)	123
6.35 Pseudo-código algoritmo ProbSparse. Fuente: (H. Zhou et al., 2021).	124
6.36 Tabla comparativa Informer. Fuente: (H. Zhou et al., 2021)	125

Lista de Tablas

4	Comparativa de métricas entre distintos modelos para Compras.	70
5	Comparativa de métricas entre distintos modelos para Ventas.	72
6	Patrón en codificación binaria.	109
7	Tabla 6 traspuesta: Frecuencia para número binarios.	109
8	Representación gráfica de vector <i>key</i> de cada palabra (columna de la izquierda) y el vector <i>query</i> de cada palabra (columna de la derecha).	111
9	Comparación entre características de la palabra “El” con la descripción del resto de palabras.	111

Resumen

El crecimiento de datos es exponencial año a año. Saber tratar y visualizar estos datos es fundamental para muchas áreas: deportes, historia, finanzas, etc. En concreto, la contabilidad, encargada del estado financiero de la empresa, procesar y analizar estos datos para ofrecer una clara visión de la salud financiera de la empresa es esencial para poder llevar un negocio de la mejor manera posible. Este trabajo crea una arquitectura software que va desde el almacenamiento de los datos en la nube hasta el preprocesamiento de estos, creación de modelos, almacenamiento de los modelos, comparativa de estos y finalmente guardar nuevos datos con predicciones para poder visualizar no solo datos hasta el presente, sino estimaciones futuras también. Todo esto de la manera más automatizada posible. El resultado final es un *dashboard* BI/BA a partir de datos contables para aportar conocimiento financiero empresarial de manera visual.

Palabras clave: BI, BA, aprendizaje automático, MLOps, nube, series temporales, visualización de datos.

Abstract

The growth of data is exponential year by year. Knowing how to handle and visualize these data is essential for many areas: sports, history, finance, etc. Specifically, accounting, which is responsible for the financial state of a company, needs to process and analyze these data to offer a clear view of the company's financial health, which is essential for running a business in the best possible way. This work creates a software architecture that ranges from storing data in the cloud to preprocessing it, creating models, storing the models, comparing them, and finally saving new data with predictions to visualize not only data up to the present but also future estimates. All of this in the most automated way possible. The final result is a BI/BA dashboard based on accounting data to provide financial business insights in a visual manner.

Key words: BI, BA, machine learning, MLOps, cloud, time series, data visualization.

Glosario

API	Application Programming Interface
BA	Business Analytics
BI	Business Intelligence
CNN	Convolutional Neural Network
CV	(K-fold) Cross Validation
DAX	Data Analysis Expressions
DBaaS	DataBase as a Service
DL	Deep Learning
EFB	Exclusive Feature Bundling
GB	Gradient Boosting
GBDT	Gradient Boosting Decision Trees
GBM	Gradient Boosting Machines
GOSS	Gradient-based One Side Sampling
GRU	Gated Recurrent Units
ETL	Extraction, Transformation and Load (of data)
IA	Inteligencia Artificial
IaaS	Infrastructure as a Service
IRPF	Impuesto sobre la Renta de las Personas Físicas
IVA	Impuesto sobre el Valor Añadido
L1	Norma Manhattan

L2 Norma Euclidiana

LASSO Least Absolute Shrinkage and Selection Operator

LSTM Long Short-Term Memory (neural network)

MAE Mean Absolute Error

ML Machine Learning

MLP Multi-layer Perceptron

NLP Natural Language Processing

PaaS Platform as a Service

PGC Plan General Contable

PyMEs Pequeñas y Medianas Empresas

RAM Random Access Memory

RNN Recurrent Neural Network

RMSE Root Mean Squared Error

RRNN Redes Neuronales

SaaS Software as a Service

SQL Structured Query Language

STaaS Storage as a Service

TCN Temporal Convolutional Network

Notación

c/p	corto plazo.
ec.	Ecuación.
e.g.	Procede del latín <i>exampli gratia</i> que traducido al español es <i>dado como ejemplo</i> .
i.e.	Procede del latín <i>id est</i> que traducido al español es <i>esto es</i> .
l/p	largo plazo.
palabra ^{AX}	En la sección de apéndice número X se puede encontrar más información.
\mathbb{R}	Números reales.
$v \in \mathbb{R}^X$	Vector v de números reales de X dimensiones.
	Tal que (en ecuaciones matemáticas).
\in	Pertenece (en ecuaciones matemáticas).
(a, b)	Intervalo abierto. Incluye números de a hasta b sin incluir a estos.
$[a, b]$	Intervalo cerrado. Incluye números de a hasta b incluyendo a estos.
$\{a, b\}$	Intervalo que únicamente comprende números a y b .
$[a, b, c]$	Intervalo que únicamente comprende números a , b y c .

1 Introducción

Cada año, España cuenta con más personas a las que le gustaría abrir su propio negocio. Según el Directorio Central de Empresas (DIRCE), en España, hay 3.2 millones de empresas activas (Instituto Nacional de Estadística (INE), 2023). Si también se tienen en cuenta las personas españolas a las que le gustaría abrir su propio negocio, estas son tres de cada diez (Confederación Española de Asociaciones de Jóvenes Empresarios (CEAJE), 2024).

El objetivo de toda persona con un negocio ya activo, o con la idea de emprenderlo, es que este negocio sea próspero, i.e. que genere beneficios. Para ello, mantener un buen funcionamiento de las cuentas y de la trayectoria financiera de la empresa resulta esencial (Ysidora et al., 2016). Estas tareas —entre otras— se encarga de llevarlas a cabo el departamento de **contabilidad** de una empresa.

Esta actividad (Esteve, 2002) lleva acompañando al ser humano mucho antes de lo que uno se puede imaginar. Investigaciones de los años noventa, permitieron a diversos arqueólogos e historiadores afirmar que los primeros documentos escritos descubiertos¹ por el ser humano no eran otra cosa que tablas de arcilla con números y cuentas. Este descubrimiento permitió a estos investigadores llegar a la conclusión de que, el nacimiento de la escritura en el año 3.000 a.c. no fue por el deseo de transmitir conocimiento a generaciones futuras —como se pensaba en primera instancia—, sino de la mera necesidad de conservar cuentas de procesos productivos y administrativos.

La contabilidad de hoy en día se rige por el método de la partida doble². Curiosamente, fue España, con las pragmáticas de Cigales de 1549 y de Madrid de 1552, el país que introdujo este método al obligar a los comerciantes de aquella época a llevar libros de cuentas consigo mismos mediante el método de partida doble (Instituto de Contabilidad y Auditoría de Cuentas (ICAC), 2023).

No fue hasta finales del siglo pasado que, con el desarrollo de la tecnología software, empresas empezaron a ver el potencial de trasladar sus registros contables de libros a una herramienta informática diseñada para hacer esta tarea de una manera más amena y eficiente. La primera de estas herramientas fue *Peachtree Accounting*, desarrollada por la empresa Peachtree Software en 1975.

Actualmente, la actividad de contabilidad, salvo en casos muy específicos de algunas pequeñas

¹Hace más de 5.000 años.

²Principio fundamental en contabilidad que establece que cada transacción económica afecta al menos a dos cuentas (deudora y acreedora), y que la suma de los débitos siempre debe ser igual a 0.

empresas, no se lleva a papel. La mayoría de empresas, en cambio, se apoyan del software para tareas contables. Hoy en día no existe una, sino múltiples herramientas informáticas de las que el usuario dispone. Un ejemplo de una de ellas es Odoo, pero hay muchas otras.

1.1 Justificación del contexto

La contabilidad explica la historia de una empresa con números. Esto lo consigue llevando un orden claro y preciso de las cuentas, actividades, recursos y dinero. En otras palabras, la contabilidad se encarga de llevar un registro de todas las operaciones de una empresa; de todo el dinero que entra y sale de ella. Esto, no solo es necesario para respaldar los datos económicos de la empresa ante las Agencias Tributarias, sino que también provee información del estado financiero actual de la empresa; así como su evolución durante un periodo determinado de tiempo. Por ende, este departamento es de suma importancia en cualquier empresa, sin importar el tamaño de esta.

El éxito de un negocio —ya sea conseguirlo o aumentarlo— es determinado por las decisiones que se tomen en él. La mayoría de estas elecciones tienen como objetivo maximizar el beneficio de la empresa. Es por ello que la contabilidad es vital para informar de, o bien del resultado de decisiones tomadas en el **pasado**, o de decisiones por tomar en el **presente** que tendrán un impacto en el **futuro**.

Para determinar si la empresa se está beneficiando de una decisión tomada en el pasado, la contabilidad aporta datos sobre la situación financiera pasada y actual de esta. Al estudio de estos datos pasados y presentes se le conoce como *Business Intelligence* (BI). De esta manera, una corporación puede saber cuánto antes las consecuencias que han conllevado sus decisiones; y, por ende, si por un lado debe seguir por el mismo camino, u optar por otro para tratar de revertir la situación.

Respecto a la toma de decisiones presentes con repercusión en el futuro, la contabilidad también juega un papel crucial. Conocer el estado financiero actual de la empresa es fundamental para saber el límite máximo de riesgo e inversión que una empresa puede asumir e incurrir respectivamente. Aunque riesgo e incertidumbre siempre va a haber a la hora de tomar una decisión con impactos futuros, un correcto análisis de los datos que una empresa posee puede reducir este riesgo a que sea el mínimo posible. El análisis del impacto de estos datos en el futuro —i.e. predicción— se le conoce como **BA** (*Business Analytics*).

A estas tomas de decisión basadas en datos se le conoce como *data-driven decision making*.

1.2 Planteamiento del problema

Como ya se ha mencionado con anterioridad, la contabilidad es la encargada de llevar, de la manera más rigurosa posible, todas entradas y salidas de dinero de una empresa. La actividad diaria de una empresa es la de realizar operaciones y transacciones internas con clientes, proveedores, etc. Es decir, no hace falta que una empresa sea muy grande para que esta genere un gran número de flujos de entrada y salida todos los días.

Estos flujos de entrada y salida son datos; un gran número de datos que una empresa obtiene día a día. Estos datos, a menos que se procesen y analicen, no sirven de mucho y es muy difícil extraer conclusiones a partir de ellos que puedan resultar útiles para la compañía. Es pues, a través de un análisis de estos datos, cuando se obtiene información relevante acerca de ellos: proveedores a los que más se compra, deudas de la empresa, sectores en los que la empresa tiene más éxito, clientes que más compran, etc.

El análisis de este gran volumen de datos, que se producen en un breve periodo de tiempo, se vuelve un reto para las empresas. Además, a medida que estas van creciendo, más flujos de entrada y salida generan a diario; lo cual supone más datos y esto implica mayor dificultad a la hora de estudiarlos.

Para obtener hallazgos significativos a partir de ellos —y por ende, permitir a la empresa prosperar—, los dos análisis comentados en el apartado anterior —el análisis de la situación pasada y actual, así como una proyección hacia el futuro— son cruciales.

Por una parte, BI ayuda a explicar la situación actual de la empresa y cómo ha llegado a estar donde se encuentra actualmente. Es decir, la evolución de esta en un determinado periodo de tiempo. Algunas de las preguntas que podría responder BI son: ¿por qué este producto (no) ha tenido éxito? ¿Por qué han aumentado las ventas en este sector y disminuido en este otro? ¿Fue correcta una decisión en concreto?

Por otra parte, BA permite analizar tendencias y averiguar hacia dónde debe la empresa centrar sus esfuerzos y atención. Las preguntas que podría responder son: ¿qué evolución debería experimentar la empresa en un periodo de tiempo determinado? ¿Se debería contratar más personal porque se espera que la empresa evolucione considerablemente en el futuro? ¿Cómo se espera que evolucionen las tendencias de distintos sectores?

Todas estas preguntas no son triviales de responder y, para poder contestarlas de la manera más rigurosa

y fiable posible, se requieren de herramientas software capaces de procesar una gran cantidad de datos (i.e. *Big Data*).

1.3 Objetivos del trabajo

Este trabajo tiene como objetivo llevar a cabo una herramienta BI/BA para el análisis de datos contables. Para ello, se espera realizar un *dashboard* en donde mostrar los datos. El trabajo parte del punto de vista que quien usa la herramienta es un directivo —o cualquier otro puesto con responsabilidad decisiva— al que le interesa que se le muestren datos contables de una empresa de una manera fácil e intuitiva. De esta manera, las tomas de decisión se vuelven más claras y más sencillas. Para ello, esta herramienta deberá contar con:

- Interactividad para que el usuario pueda ver distintos datos, y cómo afectan unos a otros.
- Una automatización para que el proceso de extracción de datos sea lo más ameno posible.
- Arquitectura enfocada a *Big Data* preparada para poder procesar una gran cantidad de datos.
- Un alojamiento seguro de los datos, solo visible para aquellas personas con permisos.
- Esta herramienta deberá ser accesible desde internet.
- Además, se mostrarán estimaciones futuras. Para ello, sería ideal un seguimiento de los modelos de aprendizaje automático, así como un registro de estos, sus métricas, sus parámetros y otra información relevante.
- Por último, y no por ello menos importante, la herramienta deberá ser sencilla de utilizar e intuitiva.

2 Estado de la cuestión

La contabilidad, con el paso del los años, se ha vuelto cada vez más complicada. Con la explotación de las tecnologías en auge a lo largo de este siglo XXI, no solo ha provocado que haya más datos distribuyéndose por la red, sino que también cada vez sea más fácil realizar transacciones. La facilidad de hoy en día en cuanto a viajar y comunicarse con otros clientes, permite a las empresas cada vez ir expandiéndose más. Las compras *online*, junto con que cada vez hay soluciones más específicas para cada tarea, también le añaden una capa de complejidad mayor a la contabilidad. Esto genera una necesidad vital de tener que depender de una o de varias herramientas informáticas para poder llevar a cabo esta tarea tan fundamental en una empresa.

No solo estas herramientas software facilitan el trabajo a las personas en este departamento haciendo los procesos contables más rápidos y precisos, sino que también abren puerta al análisis y exploración de los datos contables. Con esto, se puede adquirir un conocimiento mucho más profundo de la situación de la empresa y de cómo afrontar y determinar el futuro de esta.

En esta sección se explicarán, en una primera instancia, conceptos de contabilidad necesarios para entender el desarrollo de este trabajo. Una vez finalizado este apartado, se tratará un apartado BI/BA donde se profundizará más en características de los datos como su tipo, almacenamiento y visualización. Además, la parte teórica se cerrará explicando la sección de inteligencia artificial. Por último, se comentarán algunos trabajos relacionados con este mismo.

2.1 Marco teórico del trabajo

2.1.1 Contabilidad

Para poder llevar a cabo toda la información financiera de la empresa, la contabilidad recoge las transacciones diarias en lo que se denomina **Libro Diario**. Cualquier transacción afecta mínimo a dos cuentas. Por ejemplo, la compra de un producto por parte de la empresa, afecta a la adquisición de este producto por un parte (cuenta 1) y al proveedor a quien se le compra este producto (cuenta 2). Hay distintos tipos de cuentas y todas ellas tienen un número asociado. Este número asociado a cada cuenta es el mismo en todas las empresas españolas para todas las cuentas. Esto es porque dichas cuentas están regularizadas por el **Plan General Contable** (PGC) de España.

Para explicar de la mejor manera posible conceptos posteriores, se divide estas cuentas de la siguiente

manera:

2.1.1.1 Cuentas Activo, Pasivo y Patrimonio Neto

Activo

El **activo** se refiere al conjunto de bienes y derechos que una empresa posee en un determinado momento. Se puede dividir en:

- **Corriente** (corto plazo): dinero en caja, dinero en banco, inventario, etc.
- **No corriente** (largo plazo): terrenos, edificios, vehículos, etc.

Por lo general, largo plazo (l/p) se suele considerar un periodo de tiempo de más de un año y corto plazo (c/p) un periodo de tiempo menor a un año.

Pasivo

El **pasivo** se refiere al conjunto de deudas y obligaciones a pagar por la empresa. Al igual que el activo, el pasivo puede ser:

- **Corriente** (corto plazo): deuda a un proveedor a pagar en menos de un año o retenciones de IRPF practicadas a empleados o proveedores, etc.
- **No corriente** (largo plazo): e.g. deuda con el banco de un préstamo a cuatro años.

Un aspecto importante que el departamento de contabilidad deberá tener en cuenta, es que el pasivo corriente no deberá ser mayor al activo corriente, pues sería un indicativo de que la empresa tendrá un problema financiero a corto plazo.

Patrimonio Neto

El **patrimonio neto** es un indicativo de la salud de la empresa. En otras palabras, el patrimonio neto es básicamente el *activo – pasivo*. Por tanto, una empresa debería buscar siempre tener un patrimonio neto positivo; y cuanto mayor sea este, mejor.

Balance general

El **balance general** —también llamado balance de situación— es un documento que refleja la situación de la empresa en términos monetarios en un momento determinado. El balance es pues, como una

fotografía o resumen instantáneo de la situación de la empresa (Rajadell Carreras et al., 2014). En este documento, están reflejadas las tres cuentas que recién se han mencionado en este apartado. Está dividido en dos columnas: activos en la columna de la izquierda; y, por otro lado, pasivo y patrimonio neto en la columna de la derecha (véase la Figura 2.1).

ACTIVO		PATRIMONIO NETO Y PASIVO	
<i>Activo No Corriente</i>	<i>18.100</i>	<i>Patrimonio neto</i>	<i>14.000</i>
Inmov. Inmaterial	600	Capital	4.500
Inmov. Material	17.500	Reservas	2.500
		Beneficios no distribuidos	7.000
<i>Activo Corriente</i>	<i>15.500</i>	<i>Pasivo no Corriente</i>	<i>5.100</i>
Existencias	7.500	Deuda a LP	5.100
Clientes	6.500		
Tesorería	1.500	<i>Pasivo Corriente</i>	<i>14.500</i>
		Proveedores	6.000
		Deudas a CP	8.500
Total	33.600	Total	33.600

Figura 2.1: Ejemplo balance general. *LP = largo plazo, CP = corto plazo.*

Algo a tener en cuenta es, que la suma de los elementos de la columna izquierda (activo) es igual a la suma de los elementos de la derecha (pasivo y patrimonio neto). Esto se conoce como **la ecuación fundamental de la contabilidad**:

$$\text{Activo} = \text{Pasivo} + \text{Patrimonio neto}$$

La columna de la derecha se dice que es el **origen de fondo** de una empresa. Es decir, de ahí provienen los recursos para adquirir los activos de una empresa. Luego, la columna de la izquierda se dice que es la **aplicación** de dicho fondo.

2.1.1.2 Cuentas de compras / gastos y ventas / ingresos

El balance de situación, como se ha mencionado, sirve para proveer información acerca de la situación de la empresa en un momento dado. Pero este documento es informativo, no orientativo. En cambio,

para saber si la empresa está generando beneficios en un periodo de tiempo concreto, es la **cuenta de resultados** la que informa acerca de esto.

Compras y gastos

La cuenta de compras y gastos es la que la empresa emplea para registrar la compra de existencias —i.e. productos que la empresa compra para luego venderlos³—; así como los gastos que esta tiene como consecuencia de su actividad. Estos gastos son muy variados, pero los más comunes son: los sueldos que se pagan a los empleados, facturas de la luz, alquiler, etc.

Ventas e Ingresos

La cuenta de ventas e ingresos es justo la opuesta. Esta cuenta se emplea cuando la empresa vende alguna mercadería previamente adquirida a algún proveedor; así como los ingresos que la empresa genere más allá de estas ventas, como puede ser el alquiler de una planta de su edificio a otra empresa, prestación de servicios, etc.

Cuenta de resultados

Es a través de estas dos cuentas, con las que es posible realizar la cuenta de resultados. Este documento, a diferencia del balance general, no es informativo, sino que es orientativo. Es decir, a través de este documento se sabe si la empresa está generando ganancias, o, por el contrario, pérdidas. La ecuación para calcular la cuenta de resultados es bastante trivial⁴:

$$\text{Cuenta de Resultados} = \text{Ingresos} - \text{Gastos}$$

2.1.1.3 Plan General Contable

El Plan General Contable (PGC) es un estándar que siguen las empresas en su contabilidad. Se puede definir como el diccionario de la contabilidad, pues se encarga de asignar un **número concreto** a cada cuenta. Es decir, por ejemplo, la cuenta de compras y gastos tiene el número asociado 6XX.

³Cabe remarcar que esto es así porque se va a emplear el procedimiento especulativo de cuenta divisionaria^{A1}. Se recomienda leer el siguiente punto 2.1.1.3 antes de ir al apéndice.

⁴Debido al procedimiento especulativo de cuenta divisionaria, habría que añadir la variación de existencias (cuenta 610) a la ecuación, pero no se va a entrar en tanto detalle en este trabajo.

Dependiendo del tipo de compra o gasto, la operación tendrá un número más concreto. Dos ejemplos muy comunes son la cuenta 600 asignada a la compra de mercaderías y la cuenta 640 para los salarios que se pagan a los empleados —son un gasto para la empresa—. En definitiva, la cuenta de compras y gastos es una sección (grupo) del PGC⁵; y esta sección está dividida en subsecciones para diferenciar unos gastos de otros. Pero la cuenta de compras y gastos no es la única.

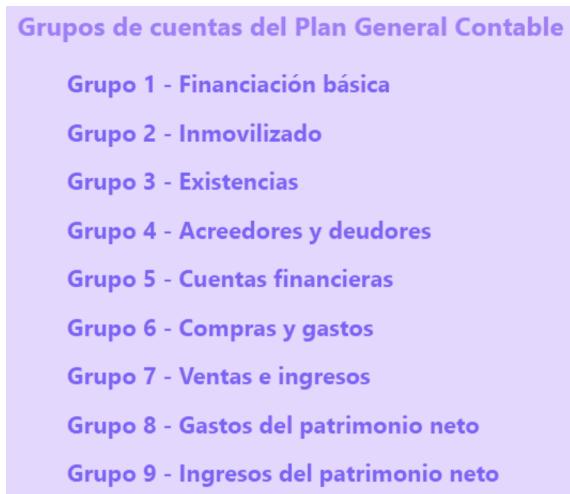


Figura 2.2: Secciones del Plan General Contable español. Fuente: <https://www.plangeneralcontable.com>

De la Figura 2.2, el grupo 6 y grupo 7 son las cuentas de compras/gastos y ventas/ingresos que se han mencionado en el apartado anterior. Por tanto, en el grupo 6 irán a parar todas aquellas transacciones relacionadas con las compras y gastos, como puede ser la factura de la luz; mientras en el grupo 7 quedarán registradas las transacciones de ventas/ingresos, como puede ser la prestación de servicio a una empresa. Las otras tres cuentas restantes del apartado anterior —i.e. activo, pasivo y patrimonio neto— están algo repartidas porque cubren muchas transacciones distintas. Las de patrimonio neto se encuentran principalmente en el grupo 1, grupo 8 y grupo 9. Los activos, en cambio, están distribuidos sobre todo en el grupo 2, grupo 3, grupo 4 y grupo 5. Los pasivos están mayoritariamente en los grupos 1 y 5.

2.1.1.4 Libro Mayor

Cada una de estas cuentas, siempre que se haga una operación que la involucre, quedará registrada en el **Libro Mayor**. Este, por lo general, suele "empezar de cero" a inicio de año. Cada cuenta tiene la

⁵Se deja en el *apéndice*^{A2} el PGC algo más detallado sobre qué es cada grupo, así como los números de cuentas —i.e. subgrupos— más empleados.

siguiente estructura:

Nombre cuenta (nº cuenta)

Debe	Haber

Figura 2.3: Estructura de una cuenta en el Libro Mayor.

La Figura 2.3 es la representación de una cuenta en el Libro Mayor. Dicha cuenta se indica arriba con el nombre de la cuenta y su número. Si se hace una anotación en el debe, se dice que se **carga la cuenta**. De lo contrario, si se hace una anotación en el haber, se dice que se **abona**. Estas cuentas, como ya se ha visto, pueden ser de 4 tipos: activo, pasivo (pasivo y patrimonio neto), compra/gasto y venta/ingreso. De manera breve, las cuentas de activo funcionan igual que las cuentas de compra/gasto; y dichas cuentas son opuestas a las de pasivo y ventas/ingresos respectivamente.

Debe y Haber en el activo y pasivo

Cuando la cuenta de un activo aumenta, este aumento queda reflejado en el **Debe**. En cambio, cuando disminuye, esto se anota en el **Haber**. De manera antagónica funcionan las cuentas de pasivo. Véase la Figura 2.4.

Activo		Pasivo	
Debe	Haber	Debe	Haber
⊕	⊖	⊖	⊕

Figura 2.4: Ejemplo cuentas activo y pasivo Libro Mayor.

La Figura 2.4 se puede interpretar como que las columnas de la izquierda (Debe) cuento más aumenten, mejor para la empresa —pues supone más activo y menos deuda respectivamente—. Justo lo contrario

ocurre con las columnas de la derecha (Haber). A final de año, cuando se evalúen estas cuentas, si el Debe en una cuenta es mayor al Haber, se dice que la cuenta presenta un saldo **deudor**. De lo contrario, la cuenta presenta un saldo **acreedor**. En caso de que el Debe y el Haber sean iguales, entonces la cuenta presenta **saldo cero**.

Debe y Haber en la compra/gasto y venta/ingreso

Estas cuentas son análogas a las anteriores. Es decir, quedarían de la siguiente manera:

Compra / Gasto		Venta / Ingreso	
Debe	Haber	Debe	Haber
+	-	-	+

Figura 2.5: Ejemplo cuentas compra/gasto y venta/ingreso Libro Mayor.

Tanto la Figura 2.4 como la Figura 2.5 están regidas por el **método de la partida doble**. Este es un principio fundamental de la contabilidad y establece que cualquier transacción económica afecta al menos a dos cuentas. Dicha transacción tiene la misma parte positiva que negativa. Por ejemplo, si se adquieren mercancías a un proveedor, la parte positiva es que la empresa adquiere dichas mercancías y la parte negativa es que hay que pagarlas. Otro ejemplo es el pago de una deuda: la parte negativa es el gasto que supone y la parte positiva es que la empresa se ha deshecho de la deuda. Como consecuencia de esto, el **balance general es siempre cero** (como ya se vio en la Figura 2.1).

En la siguiente sección se presenta un ejemplo sencillo de esto que se acaba de mencionar.

2.1.1.5 Libro Diario y asientos contables

El Libro Mayor —que recoge todas las operaciones llevadas a cabo por la empresa durante un periodo de tiempo— no es más que el resultado de operaciones diarias de la empresa. Estas operaciones diarias están registradas en el **Libro Diario**. En otras palabras: el Libro Mayor es la consecuencia de muchos Libros Diarios.

Las operaciones de los Libros Diarios se llevan a cabo en lo que se conoce como **asientos contables**. Estos asientos tienen mucho que ver con la estructura de cuentas de la sección anterior. Se dividen en

Debe y Haber también, y en él están reflejados las transacciones. Cada transacción, debido al método de la partida doble, involucra al menos dos cuentas. Además, estas cuentas involucradas deben estar repartidas en Debe y Haber, de tal manera que, el dinero en dicha transacción se anule entre Debe y Haber —i.e. cada transacción tiene la misma parte positiva que negativa, como se ha dicho con anterioridad—. Véase la Figura 2.6.

(nº asiento)	Debe	dd/mm/yyyy	Haber	
Dinero €	Nombre cuenta 1 (nº cuenta)	a	Nombre cuenta 2 (nº cuenta)	Dinero €

Figura 2.6: Estructura de un asiento contable.

Para asentar los conceptos de los puntos anteriores, se va a poner un ejemplo muy simple de cómo quedaría reflejada una transacción en el Libro Diario y en el Libro Mayor. Supóngase que se compran unas mercancías a un proveedor por valor de cien euros. Esta transacción quedaría reflejada de la siguiente manera en contabilidad:

Libro Diario		Libro Mayor	
Contabilidad		Contabilidad	
(1) Debe	1/1/24	Haber	
100 € Compra de mercancías			
(600)			
21 € H.P. ⁶ IVA soportado			
(472)	a	Proveedor 121 €	
		(400)	
(2) Debe	1/1/24	Haber	
121 € Proveedor			
(400)	a	Caja 121 €	
		(570)	
	1		2

Figura 2.7: Ejemplo transacción básica Libro Diario y Libro Mayor.

⁶H.P. = Hacienda Pública

En el ejemplo de la Figura 2.7, la compra de mercaderías es una compra (valga de redundancia) y, como aumenta, se anota en el Debe. A esta compra hay que añadirle el IVA soportado. En esta operación la empresa incurre en una deuda, i.e. pasivo que, como aumenta —pues antes no había deuda— se anota en el Haber. En el siguiente asiento, la empresa se deshace inmediatamente de dicha deuda pagando en efectivo al proveedor. Luego, el pasivo disminuye —ya no hay deuda— y por ende, se anota en el Debe. Por otra parte, el dinero en caja de la empresa es activo no corriente y, como disminuye al pagar al proveedor, se anota en el Haber.

De este ejemplo caben a destacar dos cosas:

- Se podría hacer hecho un único asiento contable, cuya transacción involucrase la compra de mercaderías y la caja —sin pasar por el proveedor—. Esto es una mala práctica, pues si las mercaderías resultan defectuosas, en la Figura 2.7 se refleja a qué proveedor se compró dichas mercaderías. De la otra manera, no.
- La suma entre las columnas de Debe y Haber el Libro Mayor es cero. Esto, como ya se comentó con anterioridad, siempre debe ser así.

2.1.2 Business Intelligence & Business Analytics

Como ya se mencionó con anterioridad, el *Business Intelligence* hace referencia a los datos hasta el momento. Estos datos aportan información sobre el recorrido de, en el caso de la contabilidad, el estado financiero de la empresa a lo largo del tiempo. Con esta información se es capaz de explicar y entender este aspecto esencial de una empresa.

Por otro lado, *Business Analytics* se entiende como el conjunto de técnicas y procesos dentro del ámbito empresarial que tienen como objetivo analizar esta información y sacar conclusiones a futuro a partir de esta. Estos datos pueden ser representados de varias maneras, pero una de las más comunes es en serie temporal, i.e. que cada valor de un dato, tiene asociado un *timestamp*.

2.1.2.1 Series temporales

Una serie temporal es un conjunto de datos observados a lo largo del tiempo. Estos datos son representados de manera muy clara en una gráfica, en la que el eje *x* de la gráfica estará el tiempo y en el eje *y* la(s) variable(s) de estudio. En este trabajo, se van a tratar con series temporales univariantes, regulares y continuas.

Una de las particularidades de las series temporales —y la razón por la que requieren un estudio

específico— es la dependencia que hay en sus datos. Es decir, cabe pensar que para pronosticar la situación de la empresa para el mes que viene, un buen punto de partida es saber el estado de esta en los últimos dos meses (por ejemplo). Las series temporales tienen cuatro componentes principales: tendencia, estacionalidad, ciclos y ruido.

Tendencia

La **tendencia** (Hyndman & Athanasopoulos, 2018) se puede interpretar como la dirección que toma una serie temporal. La Figura 2.8 muestra un ejemplo de una serie temporal con tendencia ascendente.

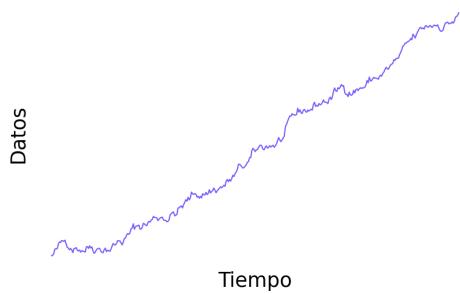


Figura 2.8: Ejemplo de serie temporal con tendencia ascendente.

Estacionalidad

La **estacionalidad** es uno o varios patrones recurrentes que ocurren en una serie temporal en períodos de tiempo regulares. La duración de dichos patrones es constante a lo largo del tiempo.

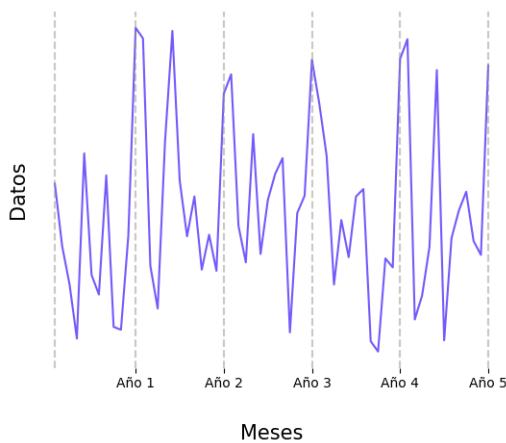


Figura 2.9: Ejemplo de serie temporal con estacionalidad.

La serie temporal en la Figura 2.9 es estacional porque justo antes del final de cada año —i.e. periodo

regular—, se puede ver una subida repentina en los datos. Además, la duración de estas subidas es constante (en torno a un mes, aproximadamente).

La Figura 2.9 también sirve para aclarar que una serie temporal con estacionalidad no es lo mismo que una serie temporal periódica. Que sea periódica implica que sea estacional, pero la relación inversa no se cumple.

Ciclos

Los **ciclos** ocurren cuando una serie temporal presenta subidas y bajadas repentinas que no son periódicas y tampoco regulares. Véase la Figura 2.10.

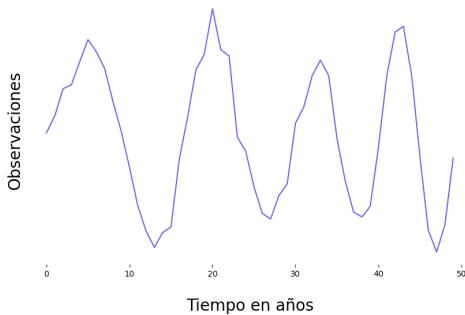


Figura 2.10: Ejemplo del componente cíclico.

No hay que confundir estacionalidad con patrones cíclicos; no tienen nada que ver: el ciclo no tiene duración fija ni periodicidad. Además, es más impredecible y lo normal es que se supere el año (como mínimo) entre un ciclo y el siguiente.

Residuo

El residuo (o ruido) es un componente que presentan todas las series temporales. No sigue ningún patrón y por tanto, es impredecible. Véase la Figura 2.11.

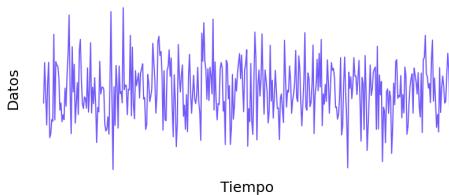


Figura 2.11: Ruido.

En el residuo, están recogidos todos aquellos eventos impredecibles como catástrofes, azar, pandemias, etc.

En la Figura 2.12 se puede ver de una manera más clara cada componente de manera aislada dada una serie temporal⁷.

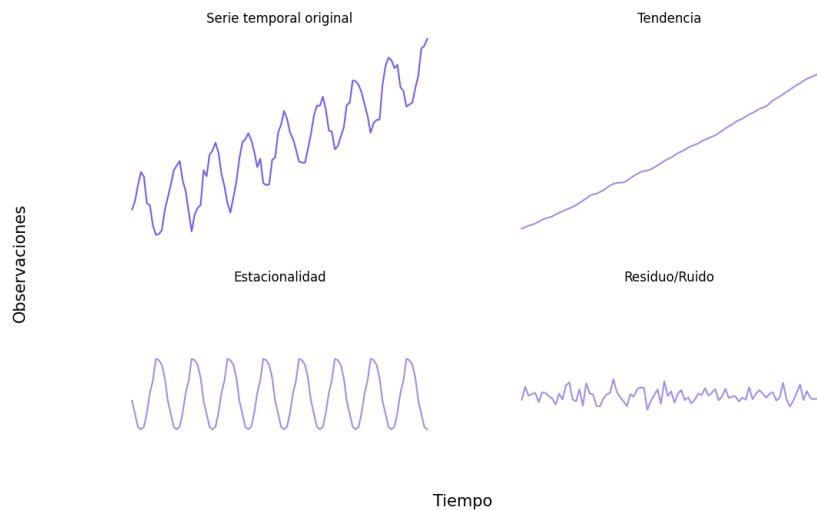


Figura 2.12: Componentes de una serie temporal por separado.

Ahora, tan importante es tener los datos en sí, como conseguir almacenarlos en un lugar seguro y accesible para posteriormente poder procesarlos y visualizarlos. Esto es lo que tratarán los siguientes apartados.

2.1.2.2 Servicios en la nube

Mientras que hace no mucho tiempo el almacenamiento de datos era *on-premise* —es decir, dentro de la propia empresa—, en el siglo XXI han surgido nuevas soluciones de almacenamiento en la nube. Los proveedores más importantes de soluciones en la nube son Amazon Web Services de Amazon (AWS, 2006), Google Cloud Platform de Google (GCP, 2009) y Azure de Microsoft (Azure, 2009). Los servicios que principalmente ofrecen estas (y otras) empresas son tres: IaaS (*Infraestructure as a Service*)

⁷El componente de ciclo, suele estar incluido en la tendencia. En muchos libros se habla de tendencia-ciclo como una única componente (Hyndman & Athanasopoulos, 2018), mientras que en otros hacen una clara distinción entre estas dos componentes (Mills, 2019) (Brockwell & Davis, 1991).

as a Service), PaaS (Platform as a Service) y SaaS (Software as a Service). Estos son los servicios principales y de los que cuelgan otros servicios como DBaaS (*DataBase as a Service*) o STaaS (*Storage as a Service*). Este trabajo en concreto se enfocará en PaaS.

Una de las principales ventajas de contratar servicios en la nube es que uno no se tiene que preocupar ni del mantenimiento, ni de la escalabilidad, ni de las actualizaciones de lo que contrata. PaaS es un término utilizado cuando se contrata un servicio en el que el programador se quiere centrar únicamente en el desarrollo de la aplicación. Es decir, que como ventajas adicionales a las recientemente dichas, en el servicio como plataforma, el desarrollador no se debe de preocupar de la infraestructura que hay por debajo, ni de la configuración a bajo nivel; únicamente de la aplicación que este esté desarrollando.

2.1.2.3 Visualización de datos

La visualización de datos es un tema fundamental en el análisis de datos. La necesidad de saber interpretar datos es un tema común en una infinidad de áreas como pueden ser el tiempo, la historia, estadística, deportes, videojuegos, política, finanzas, economía, etc. Y todas aquellas áreas donde esto sea necesario, será vital contar con una o varias visualizaciones que muestren información que desea al público objetivo.

Y es que ser capaz de mostrar en una gráfica la información relevante de una manera clara, sencilla, con los colores adecuados y otros aspectos estéticos importantes es un reto para un ingeniero de datos. Como principales aspectos a tener en cuenta en una visualización están los Principios de Gestalt (Todorovic, 2008), los mecanismos de atención estudiados por la psicóloga Ann Treisman (Treisman & Gelade, 1980) y la teoría y psicología del color.

Los gráficos, además de ser claros y seguir las reglas recién mencionadas, deben tener en cuenta a personas con discapacidades (visuales sobre todo) y deben hablar por sí mismos. Una buena gráfica no necesita que se explique. De esta manera, una visualización sirve como puente de comunicación entre un experto en datos y una persona sin ningún conocimiento en esta área.

Algo muy característico y muy importante hoy en día es la visualización de datos que todavía no han ocurrido; visualizar predicciones y poder sacar conclusiones acerca de estas para tomar las decisiones adecuadas y anticiparse a los acontecimientos. Para ello, los algoritmos de aprendizaje automático son fundamentales. Estos se van a encargar de estudiar los datos, aprender de ellos (características, patrones, etc.) y proyectar este aprendizaje en el futuro. De esta manera, los algoritmos de aprendizaje automático tratarán de aprender estas componentes para realizar una predicción. Este aprendizaje es

fundamental para generar nuevos datos, con las mismas características y patrones en el futuro.

2.1.3 Aprendizaje automático

El aprendizaje automático (también conocido como *Machine Learning* en inglés) es una rama de la inteligencia artificial. Esta rama se centra en el diseño de algoritmos matemáticos con el objetivo de que aprendan por sí solos (de ahí el nombre) principalmente a través de errores que cometan en los datos de entrenamiento y experiencia.

Hay tres tipos de aprendizaje automático: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Este trabajo, al tratarse de predicción de series temporales, se le proporciona al algoritmo tanto el *timestamp* de un dato como su valor; luego, es un aprendizaje supervisado.

Actualmente, hay numerosos algoritmos disponibles —y todos ellos sólidos— que un programador puede emplear para resolver un problema de predicción. De esta manera, en una primera instancia no se sabe cuál va a ser el algoritmo con el que mejores resultados se van a obtener. Es por ello que, es común aplicar varios algoritmos con el fin de poder compararlos y elegir al mejor. Otra puerta que deja abierta el aplicar varios algoritmos para un mismo problema, es lo que se conoce como *bagging*⁸.

Los algoritmos pues, que se van a emplear en este trabajo son cuatro: XGBoost, LightGBM, Transformer (Informer) y TCN (*Temporal Convolutional Networks*). Todos ellos son muy potentes a día de hoy y se utilizan en numerosas aplicaciones —no solo en series temporales—. Sin ir más lejos, una de las tecnologías que más popularidad ha ido ganando en los últimos años es el famoso chatbot ChatGPT. Bien, pues la T en *GPT* es de *Transformer*.

Los modelos generados por estos algoritmos, dado que van a generar nuevos datos en el futuro —i.e. que no existen— se dice que son **modelos generativos**. Esto es debido a otra de las características de las series temporales: su predicción es **extrapolar** y no intrapolar⁹.

Un concepto importante a destacar es la diferencia entre un algoritmo y un modelo —dos términos que a menudo se usan indistintamente—. De manera simple, **un algoritmo genera un modelo**. En otras palabras, el algoritmo se encarga de aprender las características de los datos y el modelo es la representación de este aprendizaje.

⁸*Bagging* es una técnica de aprendizaje automático que consiste en obtener un resultado predictivo final a partir de promediar resultados de varios modelos.

⁹Esto se explica un poco más en detalle en el *apéndice*^{A3}.

Gradient Boosting: XGBoost & LightGBM

XGBoost y LightGBM no son algoritmos de aprendizaje automático, sino que son bibliotecas de programación que emplean métodos *ensemble*; concretamente *boosting*.

Ensemble, en español, significa **conjunto**. Métodos *ensemble*, por tanto, es una práctica de aprendizaje automático en la que se aplican varios modelos para obtener una predicción final. Cabe resaltar que cada uno de estos modelos puede haber sido generado por un algoritmo distinto. La filosofía detrás de utilizar métodos *ensemble* está en que varios modelos son capaces de predecir mejor que un único modelo.

A pesar de que haya varios métodos *ensemble* como *bagging* o *stacking*, XGBoost y LightGBM pertenecen al método de **boosting** (impulso). Este, consiste en generar una regla de decisión final muy precisa, a partir un conjunto de reglas muy simples. En otras palabras, *boosting* consiste en generar un modelo final muy preciso, a partir una **suma ponderada** de modelos simples.

Un ejemplo donde se puede ver este concepto de manera muy clara es en un problema de clasificación, en el que se trata de clasificar un animal como gato o no gato. Si un modelo muy simple es capaz de detectar, a partir de una imagen, si un animal tiene o no tiene bigotes; otro modelo si tiene orejas puntiagudas; otro modelo si el animal tiene cuatro patas, etc. Al final, el conjunto de estas reglas simples son capaces de generar un modelo final muy preciso. A estos modelos simples se les conoce como *weak learners* o como *base learners*¹⁰. Está demostrado que un conjunto de estos son capaces de generar un modelo preciso (*strongly learner*) (Schapire, 1990).

Aunque cada algoritmo de *boosting* sigue sus reglas específicas, el proceso general es el siguiente:

- 1) Se crea un modelo muy simple (*weak learner*).
- 2) Se identifican los elementos que se han predicho bien y mal por el modelo.
- 3) Se calcula la precisión del modelo.
- 4) Se crea otro modelo simple (*weak learner*) que se enfoca en corregir los errores del anterior, y por tanto, mejora la precisión del anterior.
- 5) Se repiten pasos 1-4 hasta llegar a una condición de parada como puede ser generar M modelos,

¹⁰Weak learner o base learner es cualquier modelo simple capaz de generar predicciones con algo más de precisión que un modelo aleatorio. Modelo simple, *weak learner* y *base learner* se emplean de manera intercambiable en este trabajo.

alcanzar una precisión requerida, etc.

Uno de los primeros algoritmos más populares de *boosting* fue AdaBoost (Freund & Schapire, 1997) (Freund, Schapire, et al., 1996) que asignaba mayores pesos a los datos mal clasificados para que el modelo posterior se centrara más en ellos. Es decir, en AdaBoost, no se elige el siguiente modelo basándose en una función de perdida. Esta manera de enfocar la elección del siguiente modelo (con base en una función de pérdida) fue propuesta no mucho después (Breiman, 1997). Es a partir de esta idea, que surge el concepto de *Gradient Boosting* (GB) —también conocido como *Gradient Boosting Machines* (GBM)— (Friedman, 2001).

XGBoost significa *Extreme Gradient Boosting* y LightGBM *Light Gradient Boosting Machine*. Es decir, estas dos **bibliotecas** (no algoritmos) aplican *Gradient Boosting* (algoritmo) para realizar una predicción final.

Gradient Boosting es un método que aporta mucha flexibilidad a la hora de ser empleado. No solo permite al programador elegir entre varias funciones de coste distintas, sino que también permite utilizar varios algoritmos de aprendizaje automático para la generación de modelos. No obstante, debido a sus excelentes resultados, el algoritmo por excelencia empleado por los métodos de *Gradient Boosting* —y el que emplean XGBoost y LightGBM— son los árboles de decisión (Chang et al., 2019).

Un árbol de decisión (Quinlan, 1986) consiste en decisiones simples *if-then* respecto a las distintas características de una variable. Estas características se van ramificando con el objetivo de ir segmentando los datos a partir de estas (características). En la Figura 2.13 se puede ver un ejemplo simple para predecir las calorías quemadas por una persona dependiendo del tiempo del entrenamiento y su intensidad.

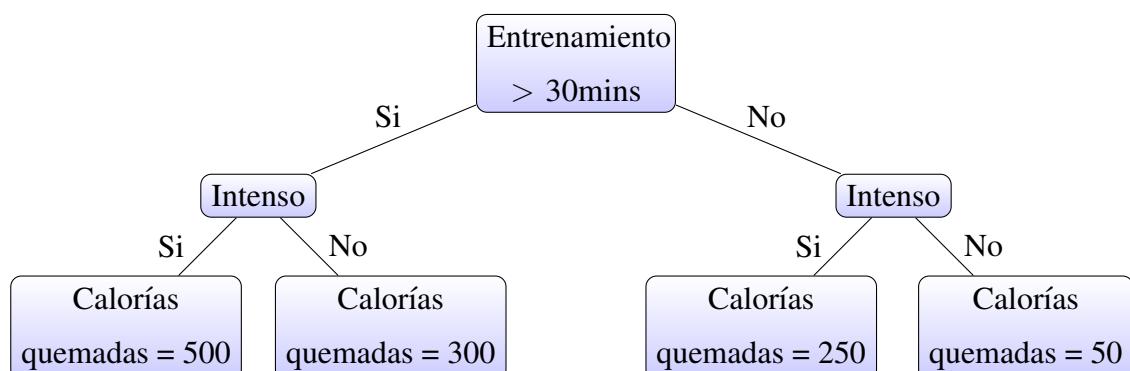


Figura 2.13: Ejemplo sencillo aplicando árboles de decisión para un problema de regresión.

Entrando más en detalle en cómo funciona este método, *Gradient Boosting* hace referencia a aplicar un descenso de gradiente^{A4.1} respecto a una función de coste, para ir obteniendo *base learners* que vayan reduciendo esta función de coste, y finalmente poder hacer una predicción final con todos ellos. El algoritmo general es el siguiente:

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^N$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

- 1: initialize \hat{f}_0 with a constant
 - 2: **for** $t = 1$ to M **do**
 - 3: compute the negative gradient $g_t(x)$
 - 4: fit a new base-learner function $h(x, \theta_t)$
 - 5: find the best gradient descent step-size ρ_t :
 - $$\rho_t = \arg \min_{\rho} \sum_{i=1}^N \Psi[y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)]$$
 - 6: update the function estimate:
 - $$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x, \theta_t)$$
 - 7: **end for**
-

Figura 2.14: Algoritmo de GBM de Friedman. Fuente: (Natekin & Knoll, 2013)

El algoritmo de la Figura 2.14 se puede ver como una combinación entre el algoritmo del descenso del gradiente con los pasos de *boosting* descritos con anterioridad. Consiste en inicializar un modelo inicial (\hat{f}_0) a una constante. Cada modelo que se genere a partir de ahora, se puede interpretar como una función, que depende de unos parámetros (pesos) que genera una salida (resultado del modelo) en base a una función de pérdida¹¹. Esta salida será mejor o peor dependiendo del valor de los parámetros. El número de modelos generados viene determinado por el número de iteraciones M , especificadas como entrada en el algoritmo 2.14. t hace referencia a una iteración en concreto, luego $t \in [0, M]$.

Cada uno de estos nuevos modelos generados, se va a tratar de un modelo muy simple (en muchas ocasiones son árboles de un nivel, también conocidos como *tree stump*, véase Figura 2.15), que tienen como objetivo mejorar de manera leve una función de coste (definida también en la entrada del algoritmo 2.14). Este nuevo modelo originado se añade a los que ya se han generado anteriormente. Pero no se añade el modelo completo, sino solo un factor de este, determinado por ρ . Este factor

¹¹Cabe recalcar que esta interpretación no es específica de *Gradient Boosting*, sino que es aplicable a cualquier algoritmo que genere un modelo.

determina la distancia a recorrer en dirección contraria al gradiente; o lo que es lo mismo, el grado en que la función de coste se actualiza. Por tanto, es como una tasa de aprendizaje de cada modelo (adicionalmente, luego se aplica una tasa de aprendizaje α , también conocido en GBM como *shrinkage*, constante a todos los modelos (Friedman, 2001)).

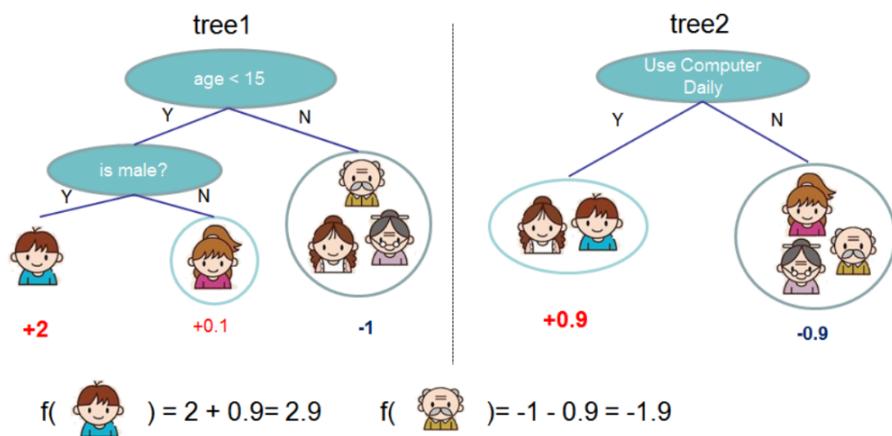


Figura 2.15: Ejemplo de dos modelos simples de árboles de decisión. Estos modelos se añaden para generar un resultado final. Fuente: (Chen & Guestrin, 2016)

Luego, el resultado de haber aplicado este algoritmo durante M iteraciones, es un modelo final compuesto por M modelos simples capaces de realizar predicciones muy sólidas. Este es el algoritmo detrás de XGBoost y LightGBM.

XGBoost

XGBoost (Chen & Guestrin, 2016) es una librería que implementa el algoritmo recientemente descrito, pero de una manera muy eficaz. Esto ha provocado que XGBoost sea relativamente común a la hora de ser empleado en competiciones de aprendizaje automático¹². Destaca, no solo por los resultados que consigue obtener, sino que también por ser una librería que permite implementar *Gradient Boosting* con las siguientes características:

- De manera escalable, siendo capaz de procesar miles de millones de datos.
- Es “paraleizable”, lo que supone un aprendizaje más rápido y, por tanto, permite una mayor exploración de modelos (se recuerda que se pueden aplicar varias funciones de coste; luego, esto es importante). Adicionalmente, otras técnicas se emplean para eficiencia.

¹²Quizá la más conocida ha sido *The Netflix Prize* (Bennett, Lanning, et al., 2007)

- Se implementa una aproximación del algoritmo *exact greedy algorithm* para mejorar el resultado obtenido por el modelo, haciendo que los árboles de decisiones se dividan de la mejor manera posible.
- Robusto ante la falta de datos.
- XGBoost es capaz de tratar con *datasets* que son tan grandes, que no caben en la memoria RAM. A esto se le conoce con el término *out-of-core*, y lo consigue al dividir el *dataset* en trozos, e ir leyendo dichos trozos de disco según se vayan necesitando. En otras palabras, no se procesa el *dataset* de una sola vez.
- Como última característica a destacar de XGBoost es la implementación de *shrinkage* (tasa de aprendizaje en GBM) (Friedman, 2001), regularización L1 y L2^{A4.2} y *column subsampling* para evitar *overfitting*. Esto es porque, si ya de por sí, los métodos de *boosting* pueden dar problemas de overfitting (Caruana & Niculescu-Mizil, 2006), los árboles de decisión suelen acarrear este mismo problema también (Bramer, 2007).

LightGBM

Pese a la escalabilidad y eficiencia que XGBoost ofrece, debido al incremento exponencial en la cantidad de datos en el mundo debido al *Big Data*, nuevas soluciones eran necesarias para tratar cada vez con más datos y más variables de dichos datos. Es por esta razón que surge LightGBM (Ke et al., 2017). Esta librería también implementa, al igual que XGBoost, GBDT (*Gradient Boosting Decision Trees*). Mejora la eficiencia del algoritmo, acelerando el proceso de entrenamiento en un factor de $\times 20$ respecto a anteriores implementaciones de GBDT —como puede ser XGBoost—, al mismo tiempo que mantiene prácticamente la misma precisión. Esto es posible al conseguir reducir el número de datos del *dataset*, y el número de variables de dichos datos. Son los algoritmos GOSS (*Gradient-based One-Side Sampling*) y EFB (*Exclusive Feature Bundling*) los encargados de esta tarea.

- GOSS, es un algoritmo que permite reducir el número de variables del *dataset* de una manera eficaz y sin perder información. Esto lo consigue al “eliminar” parte de los datos con un gradiente pequeño asociado.

De manera simplificada, si un dato tiene un gradiente pequeño asociado, quiere decir que ese dato se ha predicho (considerablemente) bien y se puede eliminar del *dataset*. En otras palabras, no interesa mantenerlo porque no genera mejora en el entrenamiento. En cambio, un dato con

un gradiente alto, supone que el algoritmo tenga que aprender a predecirlo mejor y, por tanto, interesa mantenerlo.

De esta manera, GOSS, a un nivel alto de abstracción, mantiene los datos del *dataset* con un gradiente asociado alto, mientras que descarta aquellos datos con un gradiente asociado bajo (esto supone que la distribución de los datos sea alterada y matemáticas adicionales son implementadas; pero esto va más allá del alcance de este trabajo). Que el gradiente sea alto o bajo es determinado por un umbral.

- EFB, por el contrario, se encarga de reducir las variables (características) de estos datos. Esto es de suma importancia porque simplifica mucho el *exact greedy algorithm* descrito con anterioridad en XGBoost. Dicho algoritmo (XGBoost), a pesar de que se haga una aproximación, es muy costoso, pues tiene que evaluar la información que aporta cada variable para poder hacer la mejor división de árbol posible.

Esto se puede entender muy bien con el ejemplo de la Figura 2.13. En dicha figura se tratan las variables tiempo de entrenamiento e intensidad para predecir las calorías quemadas. Supóngase que también estuviesen las variables binarias: casado y con empleo. Aunque puede ser que estas variables puedan tener algún tipo de correlación (leve) con las calorías quemadas por una persona durante un entrenamiento, cabe pensar que las variables tiempo de entrenamiento e intensidad son mucho más representativas. En otras palabras, estas dos últimas variables aportan mucha más información que la de saber si una persona está casada y con trabajo a la hora de predecir sus calorías quemadas en un entrenamiento.

Por tanto, las primeras divisiones de árbol en los primeros modelos, deberán ser respecto a las variables que más información aporten. Luego, el algoritmo deberá calcular la información que aporta cada variable para determinar la mejor división de árbol posible y, por tanto, **cuantas menos variables haya, más rápido será este cálculo**.

Esta reducción de variables se consigue al agrupar (*bundle*) variables que son mutuamente exclusivas, i.e. que rara vez tienen valores distintos de cero al mismo tiempo. Un claro ejemplo de esto es la transformación de variables categóricas en *one-hot encoding*¹³. Esto aumenta de manera considerable las columnas del *dataset* a pesar de que todas estas variables representan una misma cosa (además de ser variables mutuamente exclusivas). EFB consigue agrupar todas estas columnas en una sola¹⁴, consiguiendo reducir la dimensionalidad del *dataset* al mismo tiempo que se conserva la información.

Otra manera de ver estos algoritmos, es que GOSS reduce las filas del *dataset*, mientras que EFB disminuye las columnas. Esto provoca que el entrenamiento sea más *ligero* —de ahí el nombre de LightGBM—.

Redes Neuronales: TCN y Transformers

Estos algoritmos son algo particulares, pues son redes neuronales (RRNN). Hay tantas, tan variadas —para distintas áreas— y se emplean tanto, que tienen su propio campo dentro del aprendizaje automático: **aprendizaje profundo** (*Deep Learning*). Véase la Figura 2.16.

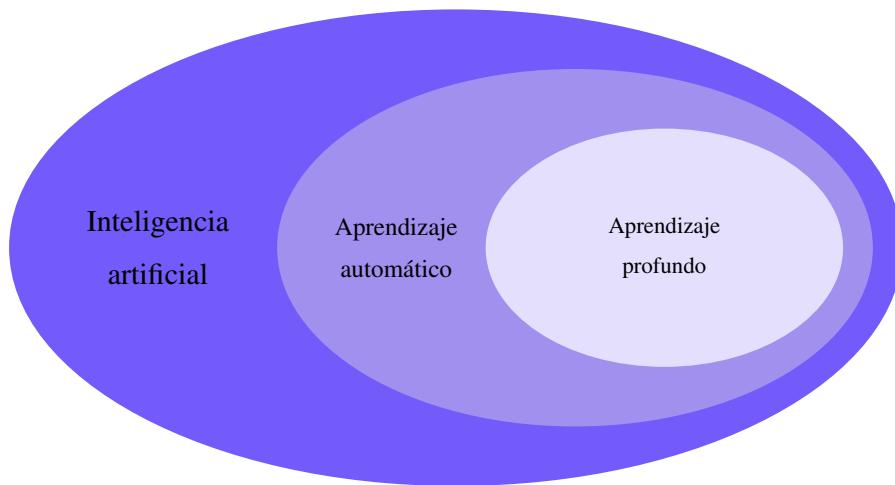


Figura 2.16: Relación entre inteligencia artificial, aprendizaje automático y aprendizaje profundo.

El inicio de las redes neuronales se da en 1943, con el primer modelo matemático sugerido de una simple neurona artificial (McCulloch & Pitts, 1943). La base lógica de las redes neuronales se basa en tratar de imitar el comportamiento de las neuronas biológicas en un ordenador. El cerebro —a un nivel alto de abstracción— está compuesto por un gran número de neuronas, interconectadas entre sí y recibiendo impulsos de sus conexiones vecinas. Si la suma de estos impulsos supera un umbral, esta neurona emite un impulso eléctrico a sus neuronas vecinas.

¹⁴Los algoritmos no entienden de texto (i.e. variables categóricas), entienden de números. Entonces, si se tiene la variable *localidad* con valores posibles: Madrid y Barcelona, esta columna de localidad se transforma entonces en dos: una de Madrid y otra de Barcelona. Estas columnas son binarias, y tendrán un 1 si dicha localidad es *True* y 0 de lo contrario. A pesar de ser dos columnas, representan una misma cosa (localidad), y además son, por lo general, mutuamente exclusivas (la localidad o es Madrid o es Barcelona).

¹⁵Cabe remarcar que este agrupamiento de características no es solo a variables que han sido transformadas a *one-hot encoding*.

Una arquitectura que involucrase más de una neurona artificial y con capacidad de auto-aprendizaje fue introducida en 1958, por el psicólogo Frank Rosenblatt (Rosenblatt, 1958). Esta arquitectura, conocida como **perceptrón**, consistía únicamente de una capa de entrada y otra de salida.

Años más tarde, surge la arquitectura MLP (*Multi-layer Perceptron*) (Werbos & John, 1974), con capas ocultas¹⁶ entre la capa de entrada y de salida. Véase la Figura 2.17.

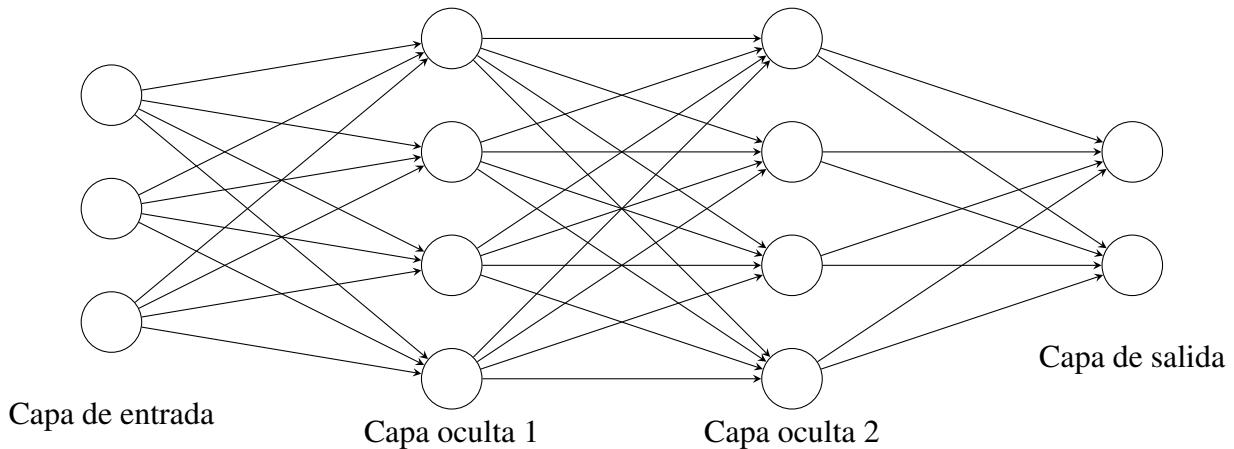


Figura 2.17: Ejemplo arquitectura de un perceptrón multi-capas con dos capas ocultas.

Las RRNN se volvieron muy populares gracias al algoritmo de *backpropagation* (Rumelhart et al., 1986) el cual mejoraba la manera en la que estas aprendían. Consiste en propagar hacia atrás los errores cometidos por las neuronas para que estas aprendan *por sí solas*. Esto se consigue al aplicar una función de coste cuyo objetivo es minimizarla lo máximo posible.

Las MLP dieron lugar a otras redes neuronales, específicas para otras áreas de la inteligencia artificial. Por ejemplo, las RNN (*Recurrent Neural Networks*) (Jordan, 1986)¹⁷, o las CNN (*Convolutional Neural Network*) (LeCun et al., 1998).

- Las RNN, se dice que tienen *memoria*. Se aplican en campos como el procesamiento de lenguaje (NLP), reconocimiento de voz o **series temporales**. Sin embargo, estas sufrían de dos problemas principales: el desvanecimiento de gradiente y/o explosión de gradiente^{A5}; y, por otra parte, que no eran capaces de retener memoria a largo plazo. Las LSTM¹⁸ (Hochreiter & Schmidhuber, 1997) y las GRU¹⁹ (Cho, Van Merriënboer, Bahdanau, & Bengio, 2014) solventaban estos problemas; y, por ende, obtenían un mejor rendimiento que las RNN (Chung et al., 2014).

¹⁶El término **Deep Learning** viene del gran número de capas —i.e. profundidad → *deep*— que estas poseen.

- Las segundas, las CNN, fueron un gran avance en el campo de la visión por computador para procesamiento de imágenes.

Transformers: Informer

Los Transformer (Vaswani et al., 2017), no son una red neuronal específica de las series temporales, ni mucho menos. De hecho, los Transformer tienen su origen en otro campo: el campo del **procesamiento del lenguaje natural** (NLP por sus siglas en inglés). De manera muy resumida, los Transformer sustituyeron a las RNN en esta área al obtener resultados significativamente mayores²².

Los Transformer, una de las grandes ventajas que tienen —y uno de los motivos por los que están establecidos como estado del arte en muchas áreas— es por su capacidad de aprender **distintos contextos** en un *input* (datos). Esto los convierte, *a priori*, en una solución viable para ser aplicados a series temporales. Por poner un ejemplo muy claro donde se ve la necesidad de aprender varios contextos series temporales, es pensando en datos de tráfico. Si estos datos son recogidos cada hora durante un año entero, hay varios contextos (dependencias) que se pueden dar: patrones diarios (hay más tráfico por las mañanas y por las tardes que a medio día y muy de noche); pero también puede haber patrones semanales (el tráfico los fines de semana por la noche es mayor que los días de diario), patrones por vacaciones (salida de Semana Santa) y patrones estacionales/anuales (suele haber menos tráfico en verano, pues la gente está de vacaciones).

Es la necesidad de aprender estos contextos varios, la razón por la que los Transformer se convierten en una opción más viable a otros algoritmos que suponían el estado del arte hasta entonces (Zerveas et al., 2021) como pueden ser: ARIMA(Box et al., 2015), LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho, Van Merriënboer, Bahdanau, & Bengio, 2014) y modelos Seq2Seq con atención (Bahdanau et al., 2014) .

Otro de los motivos por el cual los Transformer son tan atractivos para las series temporales, es porque su mecanismo de atención le permite al algoritmo aprender relaciones complejas entre **datos muy**

¹⁹Long Short-Term Memory.

²⁰Gated Recurrent Units.

²¹Como curiosidad, Michael I. Jordan fue alumno de Rumelhart (uno de los autores del artículo de *backpropagation*). De hecho, en ambos artículos: (Jordan, 1986) y (Rumelhart et al., 1986) se mencionan las RNN.

²²Se deja en el apéndice^{A6} más información acerca de los Transformer y qué ventajas suponen frente a RNN; así sobre cómo funcionan y cómo es su arquitectura.

distantes entre sí. Esto es crucial en series temporales, pues en el ejemplo mencionado recientemente, se ha explicado que puede haber dependencia entre datos entre un año y otro.

Sin embargo, la arquitectura *estándar* de un Transformer, como se ha dicho, fue diseñada para procesamiento de texto y no para datos temporales. Aunque ambos tipos de datos tengan algunos puntos en común, difieren en algunos aspectos que implican que la arquitectura del Transformer original (Vaswani et al., 2017) no sirva para series temporales. Se mencionan algunas de estas diferencias:

- En una primera instancia, la atención calculada en el Transformer se hace mediante las matrices Q y K . Las filas de dichas matrices representan un vector por cada palabra. Esto implica que la complejidad para calcular la atención en los Transformer sea exponencial ($\mathcal{O}(L^2)$). Luego, añadir un nuevo dato a la secuencia (L)—ya sea texto, serie temporal, etc.—incrementa de manera exponencial la complejidad del algoritmo —lo que lo convierte en una vía poco viable para secuencias de datos muy largas—. Esto, aunque suponga un problema para secuencias largas (independientemente de si es texto, series temporales, etc.), si es cierto que el texto, no solo se puede dividir de una manera más sencilla —e.g. en párrafos que no tengan relación entre sí—, sino que también su secuencia tiende a ser bastante más corta que la de una serie temporal²³. Además, la dependencia entre datos (palabras) muy distanciadas entre sí no es tan extrema como en series temporales. Esto se traduce en que **la manera de calcular la atención del Transformer original sea inviable para series temporales** (Wen et al., 2022).
- Otra diferencia a destacar entre texto y series temporales es la **importancia del contexto local** en esta última. Es decir, en series temporales son muy importantes los patrones, i.e. formas en la gráfica que se repiten en cierta ventana periodo de tiempo. Mientras que el Transformer original dispone de *MultiHead Attention* para capturar múltiples contextos, puede no ser suficiente al no indicarle explícitamente la importancia del contexto local. Es por esto que es importante que la atención del Transformer se enfoque especialmente en datos cercanos a uno dado para poder capturar las formas de la serie temporal —i.e. contexto local—. Véase la Figura 2.18.

²³Se puede pensar en las preguntas que se le hacen a ChatGPT; de media, suelen ser pocas palabras respecto a los cientos y miles de datos que suelen formar una serie temporal.

²⁴En el ejemplo de tráfico mencionado anteriormente, datos recopilados cada hora, pueden tener relación con datos de hace un año, i.e. con datos a una distancia de $365 \times 24 = 8760$ periodos. Un texto, normalmente dividido en palabras, muy difícil es que una palabra necesite el contexto de otra a tanta distancia.

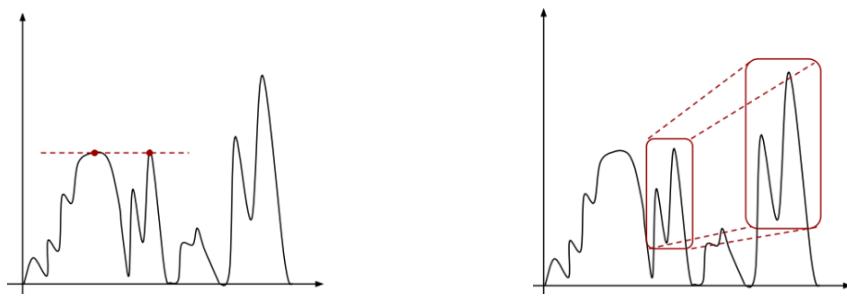


Figura 2.18: Mientras que la atención del Transformer original puede asignar una alta dependencia entre los dos datos señalados en la figura de la izquierda, una atención local puede percibir mejor las formas de la serie temporal (figura de la derecha). Fuente: (Li et al., 2019)

- Aunque los autores del Transformer (Vaswani et al., 2017), afirman que sus resultados con un codificador de posicionamiento fijo (*fixed*) —senos y cosenos— se obtienen resultados muy similares a los de emplear un codificador de posicionamiento 2.19 “aprendible” (*learnable*) (Gehring et al., 2017), estudios posteriores (Dai et al., 2019) han demostrado que el segundo es generalmente mejor; sobre todo para series temporales (Zerveas et al., 2021).

Pese a que las primeras modificaciones realizadas a la arquitectura del Transformer para aplicarlas a series temporales tenían en cuenta estos puntos (Child et al., 2019), (Li et al., 2019), el **Informer** (H. Zhou et al., 2021) va un paso más allá añadiendo ciertas modificaciones extra que mejoraron los resultados obtenidos hasta la fecha.

El Informer introdujo cuatro nuevos cambios:

- El primero, es que emplea una codificación de posicionamiento relativo, pero *especial*. Y es que, el orden de los datos en una serie temporal es mucho más importante que un texto. No solo porque en un texto se pueden cambiar el orden de las palabras sin alterar el significado de la frase —e.g. coloca la silla a la derecha— sino que además el posicionamiento en una serie temporal conlleva un *timestamp*. Este, aporta información adicional —segundo, minuto, hora, día, etc— al Transformer y por ello es conveniente tenerlo en cuenta para la predicción. Con esta codificación, el Informer consigue implementar los dos puntos recientemente mencionados: contexto local y codificación “aprendible”²⁵.
- El segundo cambio tiene que ver con reducir la complejidad de atención, haciendo que cada

²⁴Se puede pensar en las preguntas que se le hacen a ChatGPT; de media, suelen ser pocas palabras respecto a los cientos y miles de datos que suelen formar una serie temporal.

punto de la serie temporal preste atención solo a algunos otros —y no a todos—. Esto se hace con el algoritmo *ProbSparse self-attention*²⁶. La manera en la que este algoritmo calcula los datos a los que cada punto debe prestar atención tiene mejores propiedades matemáticas que los métodos empleados por los dos anteriores (Child et al., 2019) (Li et al., 2019). Con esto, se consigue reducir la complejidad de atención a $\mathcal{O}(L \ln L)$.

- El tercer cambio es que, a diferencia del Transformer original, en el que la dimensión de entrada y salida entre capa y capa —tanto en el *Encoder* como en el *Decoder*— era constante siempre. Informer, consigue ir comprimiendo la información para reducir tanto la dimensión de las matrices de salida a la mitad²⁷ (véase la Figura 2.20) como para ir reduciendo la dimensión de estas capas —i.e. números de estas matrices, h en el Transformer original (véase la Figura 2.19)—. El objetivo de todo esto es reducir la memoria del algoritmo y reducir aún más —junto con *ProbSparse self-attention*— el coste computacional sin perder información relevante.
- Como última mejora, Informer hace que el proceso de predicción del *Decoder*, en vez de ser secuencial, se haga de una sola vez. Esto, con el objetivo que evitar errores acumulados que suceden en las predicciones secuenciales (véase la Figura 2.19).

Véase a continuación la arquitectura del algoritmo y lo explicado recientemente.

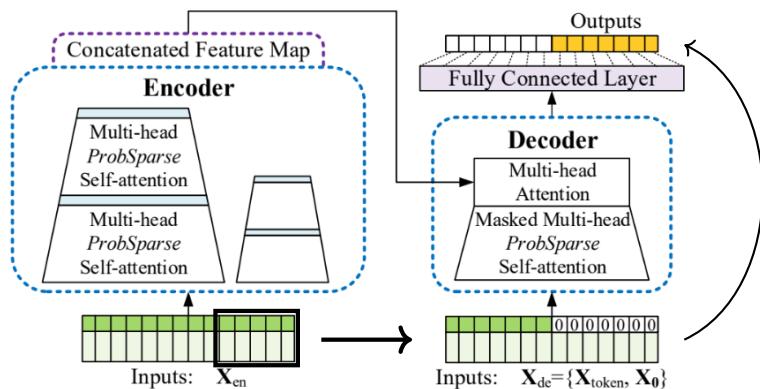


Figura 2.19: Arquitectura Informer. Fuente: (H. Zhou et al., 2021)

²⁷Se deja en el *apéndice*^{A7.1} más información sobre la codificación que realiza el Informer en los datos de la serie temporal.

²⁸Se deja en el *apéndice*^{A7.2} una breve explicación del este algoritmo.

²⁹Esto se consigue con un MaxPooling de 2×2 .

En la Figura 2.19, nótese la forma de pirámide, indicando la reducción de dimensiones (h) en cada capa (el flujo es hacia arriba). También, nótese como el *Decoder* recibe como entrada los últimos n datos de la serie temporal³⁰ (X_{token}) concatenado con un vector de todos ceros (X_0). Este vector X_0 , su dimensión, representa la cantidad de puntos a predecir en el futuro por el Informer.

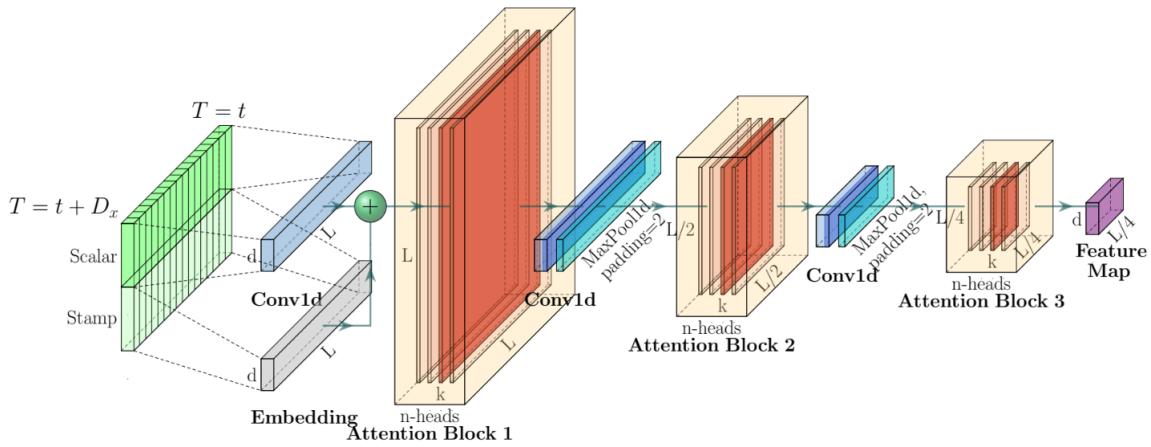


Figura 2.20: Input y tres primeras capas del Encoder en el Informer. Nótese cómo se va reduciendo la dimensionalidad (L) de las capas de atención (rojo) a la mitad. Esto se consigue con un MaxPooling de 2×2 . Fuente: (H. Zhou et al., 2021)

Adicionalmente, se deja en el *apéndice*^{A7.3} una tabla comparativa de este algoritmo con respecto a los que eran el estado del arte hasta el momento.

TCN

Las TCN (*Temporal Convolutional Networks* (Bai et al., 2018) se dan en un contexto similar al de los Transformer. Sus autores, también conscientes de los problemas de las RNN comentados en la sección anterior (secuenciales, gradiente inestable, memoria, etc.), decidieron crear un algoritmo para la predicción de secuencias³¹.

Algo que cabe remarcar, es que no se debe confundir con la otra TCN (Lea et al., 2016) que se aplican para la segmentación de imágenes. Por tanto, siempre que se mencione el algoritmo TCN en este trabajo, se estará haciendo referencia al algoritmo de secuencias (Bai et al., 2018).

Lo particular y diferente de este algoritmo, es que emplea la operación de **convolución** para realizar la predicción. Véase la Figura 2.21.

²⁹ n es un hiperparámetro, i.e. se puede elegir.

³¹Aunque el nombre del algoritmo pueda incitar a pensar que es un algoritmo para series temporales, no es así; es para

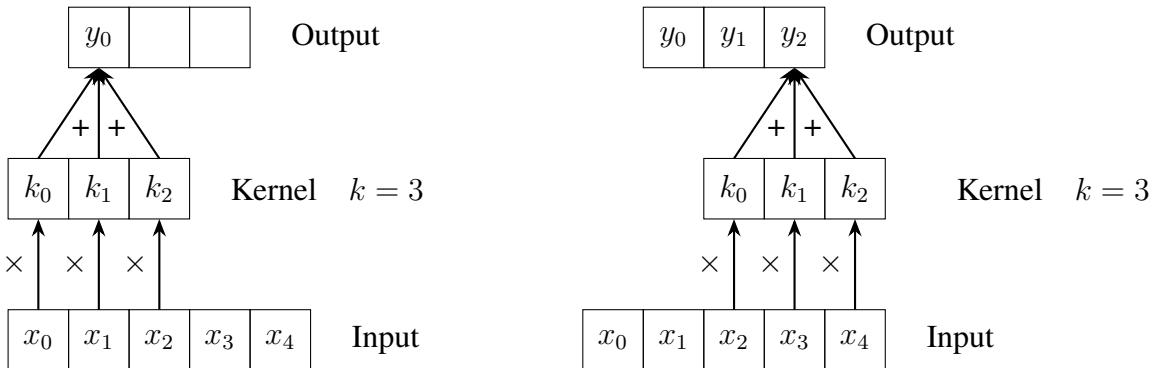


Figura 2.21: Ejemplo entre una secuencia input y cómo se obtiene la secuencia de output a través de la operación de convolución.

En la Figura 2.21 se emplea un *kernel* de tamaño $k = 3$ y se hace la suposición de que la secuencia de entrada es un vector unidimensional³².

Las TCN son capaces de mantener relaciones entre datos muy distantes entre sí. Es decir, tienen sustancialmente más memoria que las RNN y su familia (LSTM, GRU, RNN encoder-decoder, etc.). Adicionalmente, son “paralelizables” y sus gradientes son más estables —al igual que los Transformer—. Tienen dos características principales, aunque como se comentará más adelante, no son suficientes. Estas características son:

- La longitud de secuencia de entrada y de salida coinciden.
- No se usa información futura en la predicción de la secuencia. Es decir, dada una entrada de datos x_0, \dots, x_t , y se quiere predecir una secuencia de salida y_0, \dots, y_t , para la predicción de un dato y_i con $i \in [0, t]$ no se usa información de un dato x_j | $j > i$. Esta idea es la misma que la de enmascarar la información en el *Decoder* de los Transformer.

En la Figura 2.21 no se cumplen las dos propiedades que recién se han comentado. La dimensión de entrada ($X \in \mathbb{R}^5$) no es la misma que la de salida ($Y \in \mathbb{R}^3$). Y también, para el cálculo de y_0 , por ejemplo, se han utilizado datos futuros (x_1 y x_2). Lo mismo para y_1 e y_2 . Para solventar ambos problemas basta con añadir lo que se conoce como (*left*) *zero-padding*. En este caso, de dos dimensiones —i.e. $k - 1$ —, pero más adelante se verá que cambia. Véase la Figura 2.22.

secuencias. Esto engloba muchos más campos que las series temporales.

³²Podría darse el caso de que la entrada tuviera varios canales. Por ejemplos, una imagen de color es tridimensional. En los ejemplos siguientes se va a suponer que la entrada tiene un canal, es decir, la entrada es un vector unidimensional.

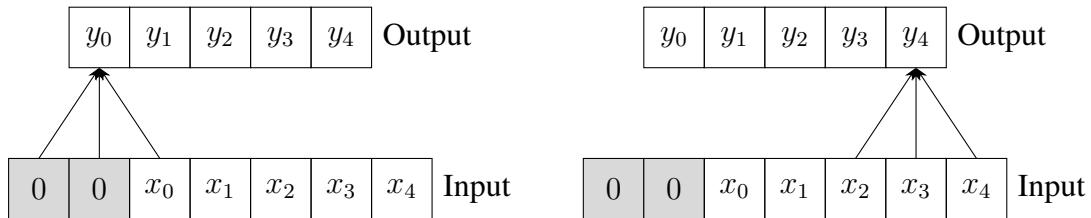


Figura 2.22: Convolución con zero-padding. Se obvia la convolución para hacerlo más claro.

De esta manera, ya se cumplen las dos primeras características mencionadas anteriormente. A este tipo de convolución se le conoce como **convolución causal**. Sin embargo, esto, como ya se ha dicho, no es suficiente. En el apartado anterior de los Transformer, se comentó que los datos temporales pueden tener relación con datos muy distanciados entre sí. Es decir, una propiedad deseable a tener por el algoritmo, es que un punto y_i sea capaz de tener información de x_0, x_1, \dots, x_i , i.e. contexto global. El procedimiento que se está siguiendo, a medida que se van aumentando las capas, no es viable. Véase la Figura 2.23.

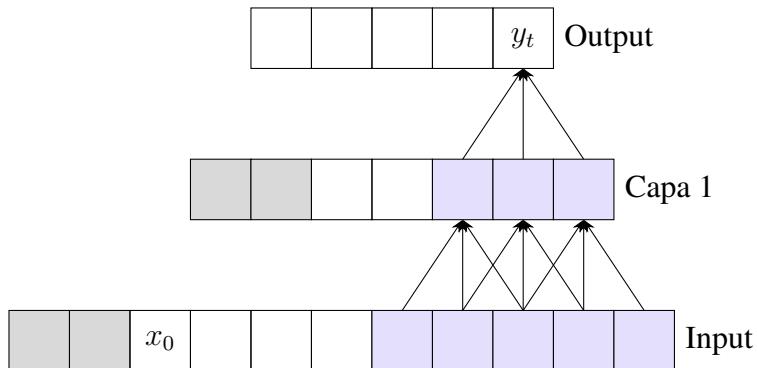


Figura 2.23: Ejemplo de inviabilidad al aumentar capas.

En la Figura 2.23 se puede ver claramente, que si la entrada fueran cientos de datos, muchas capas intermedias iban a ser necesarias para que la información de x_0 le llegue a y_t . Esto es porque el campo receptivo entre capa y capa crece de manera lineal.

Para solventar esto, se introduce el concepto de **dilatación** (Oord et al., 2016). Esto quiere decir que, según se van aumentando las capas, el campo receptivo se va expandiendo de una manera exponencial. Una forma muy común de establecer esta dilatación d es a través de elegir una base $b = 2$ y elevarla a i : siendo i en número de capas en que esté operando la dilatación. De esta manera, $d = b^i$. Véase la Figura 2.24.

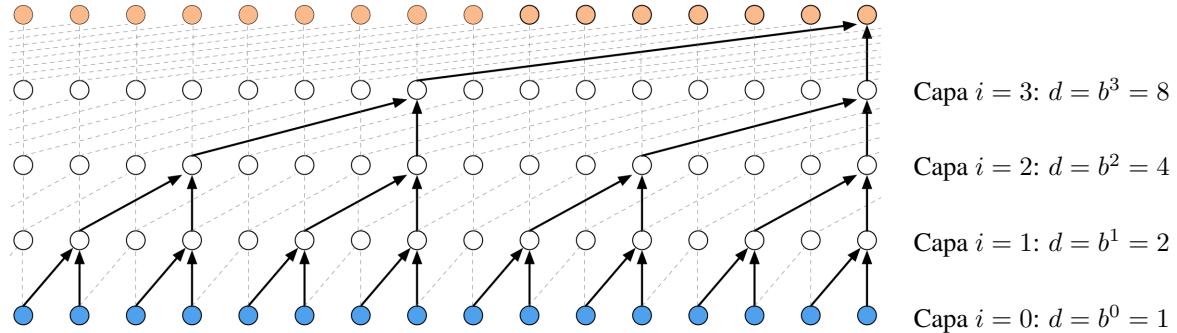


Figura 2.24: Dilatación. Fuente: (Oord et al., 2016)

Cabe remarcar que no cualquier valor de b es válido y compatible con el tamaño del *kernel* k (Lässig, 2019). También, la introducción de la dilatación hace que el (*left*) *zero-padding* ya no sea constante.

Por último, debido a la gran profundidad de la red neuronal, se emplea un bloque residual por capa. Este bloque residual (He et al., 2016) tiene la misma finalidad que el visto en los Transformer: mejorar la eficiencia del algoritmo. No obstante, es algo distinto. Véase la Figura 2.25.

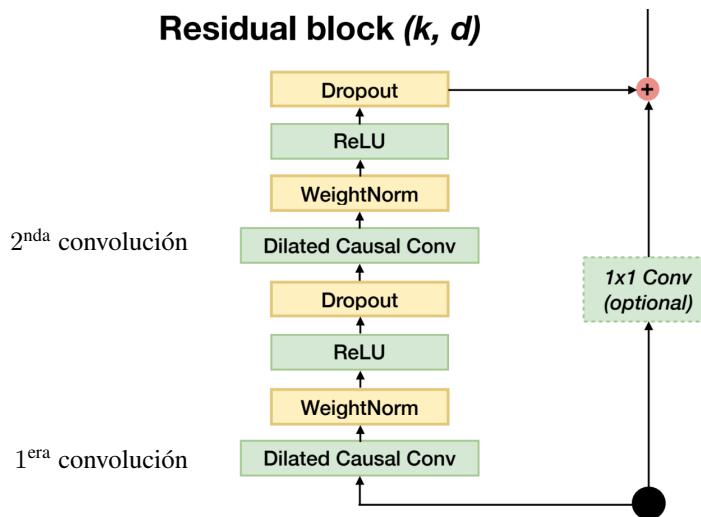


Figura 2.25: Bloque Residual TCN. Fuente: (Bai et al., 2018)

Es decir, si antes (Figura 2.24), en cada capa i , se aplicaba una convolución con un tamaño de *kernel* $k = 3$ y una dilatación d , ahora se aplican dos. Además, después de cada convolución, se aplica una normalización de los datos (Salimans & Kingma, 2016)³³ para simplificar los valores con los que la red entrena y, por tanto, reducir la complejidad de los cálculos; lo que implica mayor rapidez de entrenamiento.

Después, se aplica una función de activación ReLu (Nair & Hinton, 2010), para introducir no linealidad a los datos y que la TCN sea capaz de aprender patrones más complejos. Además, como medida para evitar *overfitting*³⁴ se aplica la técnica de *Dropout* (Srivastava et al., 2014). Esto último, desactiva aleatoriamente neuronas en cada capa, con el fin de que la neurona sea más robusta en el aprendizaje. Por último, el bloque residual suma el *input* de la capa con el *output* obtenido después de haber hecho todo el proceso. Opcionalmente, en caso de que la entrada no tenga el mismo número de canales (dimensión) que la salida, se aplica un *kernel* para igualar ambas dimensiones y, por tanto, que se puedan sumar ambos vectores.

Después de haber visto la arquitectura general de una TCN, se puede deducir una ventaja adicional a las ya comentadas al principio de esta sección. Esta ventaja es su flexibilidad a la hora de modificar parámetros, permitiendo alterar los valores de k , d , b para encontrar una combinación correcta. Esto permite tener un gran control sobre el modelo.

2.2 Trabajos relacionados

2.2.1 Aplicación de Business Intelligence en una pequeña empresa mediante el uso de Power BI

En este trabajo (Rivera Resina et al., 2018), su autor se propone crear un *dashboard* con el objetivo de tratar de mostrar información valiosa a partir de los datos de una Pequeña y Mediana Empresa (PyME). Uno de los motivos más importantes para realizar este trabajo es la alta competencia que hay entre PyMEs en España y para poder destacar entre toda la competencia, el autor realza la importancia de conocer los datos de un negocio propio.

Con ello, el autor se propone analizar los datos y mostrarlos para tratar de sacar conclusiones significativas acerca de la empresa que provee los datos. Y que, de esta manera, se pudiera ayudar a entender la situación de esta y mejorarla. Este trabajo, pues está enfocado exclusivamente en el *Business Intelligence*.

La herramienta que se elige para la creación del *dashboard* es Power BI, la misma que en este trabajo. Además de lo comentado anteriormente, el autor también trata de demostrar la facilidad de uso de

³⁴Este tipo de normalización (*weighted normalization*) funciona mejor que la clásica *batch normalization* en modelos generativos (Salimans & Kingma, 2016).

³⁵Una RRNN sufre de *overfitting* cuando aprende los datos de entrenamiento demasiado bien y luego es incapaz de generalizar y, por tanto, no obtiene buenos resultados con datos que no ha 'visto'.

esta herramienta, asegurando que no se tienen que tener conocimientos previos de programación. Con ello, el autor realza la importancia de que los gestores de una empresa conozcan y sepan usar esta herramienta para poder sacarle el mayor partido posible a sus datos y, consecuentemente, a su negocio.

2.2.2 Transformers in Time Series: A Survey

Debido al gran atractivo que presentan los Transformer para las series temporales, en un periodo muy breve de tiempo, se han desarrollado diversas arquitecturas, cada una atacando un problema distinto, o el mismo desde una perspectiva diferente. Este artículo (Wen et al., 2022) resume de una manera muy clara las mejores arquitecturas de Transformer, así como sus ventajas y limitaciones. La mayoría de cosas que se comentan, se han mencionado ya en la sección 2.1.3. Cabe destacar que en este artículo se mencionan dos arquitecturas Transformer posteriores al Informer para series temporales univariantes³⁶, y con mejores resultados: Autoformer (H. Wu et al., 2021) y FEDFormer (T. Zhou et al., 2022). El motivo por el que se ha elegido Informer es porque es *open-source* facilitando la aplicación de este modelo. Una de las conclusiones más interesantes de este artículo, es que muestra cómo los distintos Algoritmos de Transformer obtienen mejores resultados con bajo número de capas. Es decir, de alguno manera no se está explotando todo el potencial de los Transformer para series temporales; lo que indica que todavía queda mucha investigación en este campo, pues todavía no se ha encontrado una respuesta definitiva a la pregunta: ¿Cuál es el diseño apropiado de una arquitectura Transformer para el análisis de series temporales?

³⁶Para series temporales multivariantes, hay otras opciones que han mostrado mejores resultados como Spacetimeformer o TST (*Time Series Transformer*).

3 Aspectos metodológicos

Las técnicas y metodologías aplicadas en este trabajo tienen su porqué. En este apartado se explicará, en primer lugar, las metodologías aplicadas y, posteriormente, el porqué se han elegido ciertas técnicas, procedimientos y metodologías en lugar de otras. El desarrollo de este trabajo tuvo una planificación y una selección de algoritmos previos a su realización. Las tecnologías, sin embargo, se fueron descubriendo a medida que se avanzaba en el desarrollo.

3.1 Metodología

Respecto a la metodología aplicada en este trabajo, debido a que se ha hecho en solitario, no se ha llevado a cabo ninguna metodología ni pautas a la hora de desarrollar el trabajo en concreto. Sin embargo, sí que se ha tratado de aplicar metodologías ágiles en la medida que se ha podido; concretamente **Scrum**.

Las metodologías ágiles vienen a sustituir a sus predecesoras: metodologías pesadas. Estas últimas dividían un proyecto software en distintas fases, y no se procedía a la siguiente fase antes de acabar por completo la actual. Esto las convertía en metodologías muy rígidas y poco accesibles ante cambios —algo muy común en proyectos software—.

Las metodologías ágiles, sin embargo, son todo lo contrario: están pensadas para que cambios en el proyecto supongan el menor inconveniente posible. Scrum concretamente, es una metodología que se basa en un desarrollo de software iterativo e incremental que utiliza ciclos de vida cortos llamados *Sprints*. En ellos hay una reunión entre el equipo desarrollador y el cliente para *feedback* y refinamiento, modificación o adición de requisitos. En el caso concreto de este trabajo, el autor era el “desarrollador de software” y el tutor el “cliente” quien aportaba opiniones y modificaciones en reuniones periódicas para el seguimiento del trabajo.

3.1.1 MLOps

Adicionalmente a Scrum, otra metodología incorporada en el trabajo ha sido *Machine Learning Operations* (MLOps) para el entrenamiento, almacenamiento y seguimiento de los modelos generados, así como para la automatización de estos pasos recientemente mencionados. MLOps surge a partir de la idea de DevOps, una metodología de trabajo en proyectos software que mejora y agiliza la comunicación entre los equipos de *Developers* y *Operations* tratando de automatizar en la medida de

lo posible distintas fases del ciclo de vida de un proyecto de software. De esta manera se consigue lo que muy comúnmente se conoce como *Continuous Integration/Continuous delivery* (CI/CD).

MLOps, por tanto, algo más centrado en el campo aprendizaje automático, son prácticas que facilitan el desarrollo, almacenamiento y monitoreo de modelos de aprendizaje automático en producción de una manera eficiente y automatizada. Esto principalmente involucra el versionado de modelos, la monitorización del rendimiento y guardar aquella información relevante del modelo entrenado. Para conseguir esto, las dos herramientas fundamentales aplicadas en este trabajo han sido **Prefect** para la automatización y **Miflow** para el almacenamiento de modelos, y versionado e información de estos. Estas herramientas son explicadas con más detalle en el siguiente apartado.

3.2 Técnicas aplicadas

Detrás de todo algoritmo de inteligencia artificial hay una componente matemática fundamental. Esta rama de la ciencia es tan amplia que puede dar soluciones con procedimientos distintos a un mismo problema. Por esto mismo no hay uno, sino varios algoritmos de aprendizaje automático a disposición del programador para poder ser aplicados al problema en cuestión. Esta variedad de posibles elecciones deja la puerta abierta a la selección de varios algoritmos y su posterior análisis y comparación. Por esto mismo se han elegido no uno, sino cuatro algoritmos de aprendizaje automático descritos en la sección anterior: XGBoost, LightGBM, TCN e Informer.

En primer lugar, las principales razones por la que se ha elegido XGBoost son por lo versátil que es; siendo un algoritmo válido para distintos problemas de aprendizaje supervisado, así como su popularidad en concursos de Kaggle (Chen & Guestrin, 2016). Esto lo convirtió en un algoritmo muy atractivo de aprender y aplicar en este trabajo.

LightGBM por otra parte, al ser un algoritmo que surge a raíz de mejorar la eficiencia de XGBoost (Ke et al., 2017), se consideró oportuno una vez elegido aplicar XGBoost, aplicar también LightGBM. Como dos últimos algoritmos aplicados están TCN e Informer. Su característica común es que son redes neuronales, pero lo que los hace especiales —y por lo que se ha decidido aplicarlos en este trabajo— es que el primero es específico de series temporales y el segundo tiene la arquitectura de red neuronal más potente y llamativa que hay hasta día de hoy.

3.3 Tecnologías aplicadas

Para poder llevar a cabo el desarrollo de este trabajo, hay una serie de tecnologías que se han aplicado en este trabajo.

3.3.1 Entorno y librerías

El trabajo se ha desarrollado en un entorno virtual de Anaconda. La ventaja principal de crear un entorno virtual es que se instalan únicamente las librerías y dependencias necesarias para realizar la tarea que se quiera hacer. El lenguaje de programación junto con sus librerías y versiones son los siguientes:

- Python 3.12.0
- Pandas 2.2.2
- Numpy 1.26.4
- Matplotlib 3.8.4
- Keras 3.3.3
- Tensorflow 2.16.1
- Torch 2.3.0
- Darts 0.29.0
- Xgboost 2.0.3
- Lightgbm 4.3.0
- Dagshub 0.3.27
- Mlflow 2.12.2
- Prefect 2.19.2
- Azure-storage-blob 12.20.0

3.3.2 Herramientas Software empleadas

Las herramientas software, además del entorno virtual y sus librerías aplicadas en el trabajo son las siguientes:

3.3.2.1 Git y Github

Se tratan de dos herramientas software muy comunes en cualquier ámbito del software. Su principal objetivo es tener un control de las versiones de un proyecto de software, siendo pues, fundamentales a lo largo del ciclo de vida de este.

3.3.2.2 Mlflow

Esta herramienta es utilizada en ámbitos de inteligencia artificial y permite llevar un seguimiento de los distintos modelos generados a lo largo del tiempo. Contribuye a hacer unas buenas prácticas de MLOps, en donde se tiene un seguimiento constante de todos los modelos generados, con cualquier información que se pueda considerar relevante acerca de estos como pueden ser sus parámetros, métricas obtenidas y casi cualquier información adicional, e.g. una gráfica de su entrenamiento. Todo esto, Mlflow lo organiza en lo que él llama “experimentos”.

3.3.2.3 Dagshub

Dagshub es un proyecto de software que tiene como objetivo hacer la vida más fácil a los desarrolladores de inteligencia artificial. Se trata de un repositorio, similar a Github, pero enfocado en datos, permitiendo a sus colaboradores hacer versionado de modelos, compararlos y guardar ahí los experimentos de Mlflow.

3.3.2.4 Prefect

Prefect es una herramienta para la automatización y orquestación de flujos de trabajo. De manera simple, se puede ver como un *cron job* para ejecutar código de manera automatizada, pero es mucho más potente. Se pueden determinar parámetros de entrada para estos archivos de código, además de que no están restringidos a tener que estar en el ordenador propio, sino que pueden estar alojados en un repositorio Github —como ha sido el caso de este trabajo— o en algún servicio en la nube. Dos características adicionales y fundamentales de Prefect es que se puede decidir dónde ejecutar el código y llevar un seguimiento de esta ejecución. Es decir, saber si ha fallado, en qué momento ha fallado y el porqué.

Otra herramienta que se estudió aplicar, complementaria a Prefect, es Airflow. Esta herramienta es mucho más potente y mucho más compleja con funcionalidades que Prefect no tiene. Pero, para lo necesario de este trabajo, Prefect era suficiente.

3.3.2.5 Power BI

Como herramienta de visualización de datos se optó por elegir Power BI. Se trata de una herramienta de Microsoft para poder crear gráficos interactivos de una manera simple desde casi cualquier fuente de datos.

Otra herramienta muy conocida y que se contempló utilizar para este trabajo fue Tableau. Se descartó sin ninguna razón en concreto; no se conocía ninguna de las dos herramientas con anterioridad y se eligió Power BI.

3.3.2.6 Azure

Para el almacenamiento de los datos se ha utilizado la nube de Azure Microsoft. Concretamente, se han empleado los servicios de:

- **Azure Blob Storage:** Se trata de una *datalake* en donde se almacenan dos archivos: un `.csv` con los datos en "crudo" y un `.parquet` con los datos transformados y con las respectivas predicciones hechas.
- **Azure SQL Database:** Esta es la base de datos SQL de donde Power BI leerá los datos. Asimismo, esta base de datos está gestionada con Azure SQL Server y con la herramienta SQL Server Management Studio (SSMS).
- **Azure Data Factory:** Se utiliza para crear, programar y orquestar flujos de trabajo de datos. Permite a los usuarios mover y transformar datos de diversas fuentes a diferentes destinos, facilitando la creación de *pipelines* de datos que pueden automatizar los procesos de ETL (Extracción, Transformación y Carga). Para este trabajo en concreto, se utiliza para mover los datos de Azure Blob Storage a Azure SQL Database.

Para poder visualizar mejor la arquitectura del trabajo y cómo estas herramientas están conectadas entre sí, véase la Figura 4.1.

4 Desarrollo del trabajo

En esta sección se va a proceder a explicar el desarrollo de este trabajo. Se va a dividir en seis partes:

- Preprocesamiento de los datos.
- Procesamiento de los datos: Creación de modelos.
- MLOps: monitorización y automatización del proceso.
- Comparativa de modelos.
- Almacenamiento de los datos.
- Visualización de los datos.

Para tener los pasos mejor pautados, véase la Figura 4.1 en donde se muestra la arquitectura del proyecto. También se puede ver de una manera más clara y visual cómo están conectadas las tecnologías explicadas en el apartado anterior.

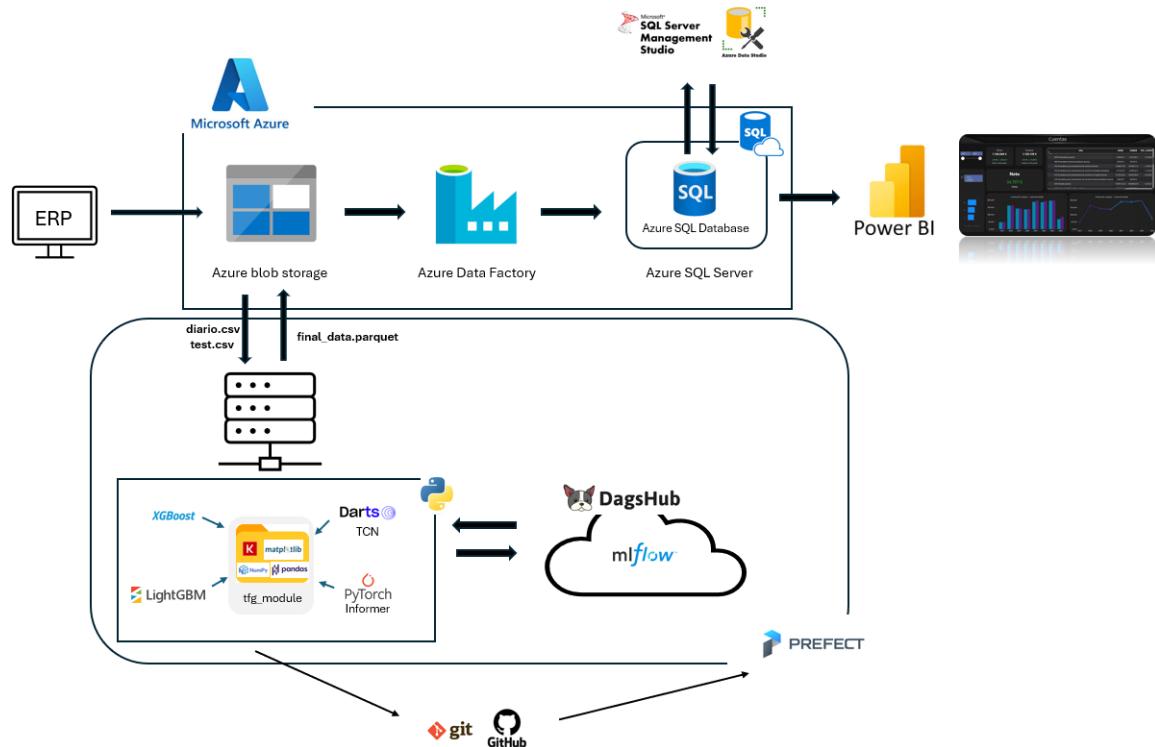


Figura 4.1: Arquitectura software del proyecto.

Cada sección de la Figura 4.1 se va a explicar más en detalle a lo largo de este apartado. No obstante,

se ve apropiado comentarla por encima. Todo empieza en Azure Blob Storage donde los archivos (.csv en este caso) están almacenados. Estos archivos contienen datos contables y se dividen de esta manera de tal forma que `diario.csv` serán los datos contables empleados para entrenamiento y validación y `test.csv` para testeo utilizados por los algoritmos de aprendizaje automático.

Una vez establecido el punto de partida, un servidor (en caso de este trabajo lo simula el propio ordenador) recoge estos archivos .csv para preprocesarlos, procesarlos y generar predicciones con los cuatro algoritmos mencionados en la sección 2. Estas predicciones son a partir de series temporales de Compras y Ventas, obtenidas en el preprocesamiento de los datos.

Los modelos generados a partir de los algoritmos son almacenados en Mlflow. Concretamente en un repositorio de Dagshub en la nube. De esta manera se tiene un control y seguimientos del proceso de aprendizaje automático. Una vez se tienen los modelos ahí, se eligen a los mejores modelos de cada algoritmo para poder compararlos y elegir al mejor y utilizar sus predicciones para posteriormente poder mostrarlas. Esto se resume en el archivo `final_data.parquet` que contiene los datos a utilizar por Power BI.

Este `.parquet` se envía en nuevo a Azure Blob Storage, en donde Azure Data Factory, se encargará de copiar estos datos a una tabla en una base de datos SQL. Será de este origen de donde Power BI extraiga los datos para posteriormente crear las visualizaciones.

Algo muy importante a mencionar es que todo el desarrollo y de más relacionado con este trabajo está alojado en un repositorio de [Github](#)³⁷. Ahí se puede ver e indagar más en detalle en el desarrollo de este trabajo; más allá de lo que se va a comentar en esta sección. Además, contiene un README.md que se recomienda leer para un entendimiento más profundo del contenido del repositorio; así como de claves en el desarrollo de este trabajo.

4.1 Preprocesamiento de los datos

Los algoritmos de aprendizaje automático necesitan un preprocesamiento de los datos previo para hacer un entrenamiento eficiente y, de esta manera, obtener un resultado lo más preciso posible. Como punto de partida, véase la Figura 4.2.

³⁷<https://github.com/JCOQUE/TFG-ingeneria.git>

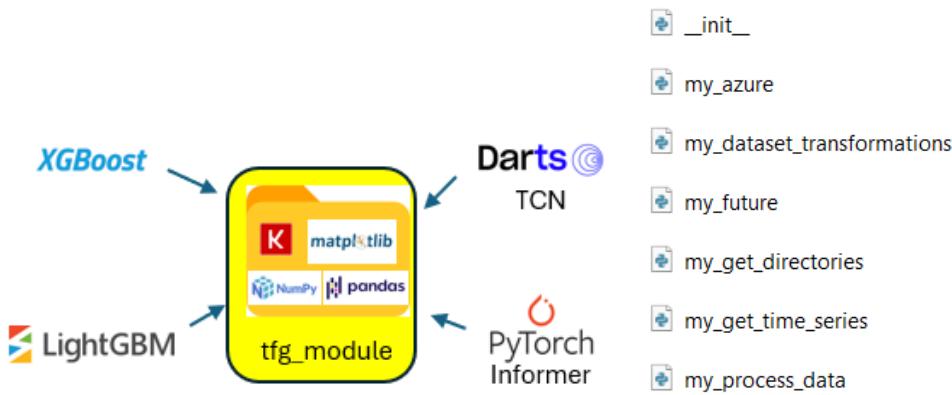


Figura 4.2: Carpeta tfg_module (izquierda) junto a su contenido (derecha).

En la Figura 4.2 se puede ver a la izquierda la carpeta "tfg_module". Esta se trata de un módulo de Python creado por el autor para el desarrollo de este trabajo. A la derecha de la Figura 4.2 se puede ver el contenido de este módulo. En él hay un archivo `__init__.py`. Esto se trata de una nomenclatura del propio lenguaje de programación de Python que indica que esta carpeta es un módulo de Python y que, por tanto, tiene un comportamiento distinto al de una carpeta normal. El resto del contenido son archivos `.py`, cada uno con un objetivo distinto.

Este módulo hace muchas cosas, pero las *más importantes* son:

- Extraer (*pull*) y enviar (*push*) archivos de Azure Blob Storage vía API (`my_azure.py`).
- Preprocesa los datos para añadir columnas (e.g. Compras y Ventas) a partir de las ya existentes (`my_dataset_transformations.py`).
- Una vez preprocesados los datos, se encarga de transformarlos a una serie temporal de Compras (o de Ventas, según se indique). Esto quiere decir que se obtiene un *DataFrame* únicamente con una columna `Fecha` y otra `Compras`. De esto se encarga `my_get_time_series.py`.
- También es encarga de generar las fechas futuras sobre las que se va a predecir, así como de almacenar estas predicciones en un *DataFrame* para posteriormente poder almacenarlas en un `.csv`. De esto se encarga `my_future.py`.

Todas estas funcionalidades son comunes a los algoritmos de aprendizaje automático empleados en el trabajo. Además de lo comentado recientemente, este módulo aporta una característica fundamental al desarrollo de código: abstraer funcionalidad para cada distinto algoritmo de aprendizaje automático. Con esto se consigue un código más limpio por tres motivos: el primero es porque, de esta manera,

el código de los algoritmos se centra en eso: en los algoritmos. Toda la parte de preparación de datos se lleva por detrás en este módulo. El segundo es que, debido a que cada algoritmo pertenece a una librería distinta (XGBoost, LightGBM, Darts y PyTorch), cada uno tiene sus peculiaridades y su manera de preprocesar los datos, entrenarlos y obtener resultados. Con este módulo lo que se consigue es que el código de cada algoritmo sea muy similar al del resto, pues estas diferencias de librerías se tienen en cuenta en el módulo y no en el código de los algoritmos en sí. En otras palabras, **se consigue estandarizar**, en la medida de lo posible, el código de los cuatro algoritmos de aprendizaje automático. Por último, se consigue una reutilización de código eficiente, cumpliendo así con un principio de programación fundamental: *Don't repeat yourself* (DRY).

Para ver un ejemplo visual de lo que se acaba de mencionar, véase la Figura 4.3 en donde `my_process_data.py` del módulo creado tiene en cuenta si tiene que procesar³⁸ los datos para el algoritmo Informer o cualquier otro.

```
def create_features(df, target = None, informer = False):
    ...
    This function is in charge of splitting the time series DataFrame
    (which contains two columns: date and target) into input features
    for training (i.e. converting date, the input into hour, dayofweek,
    etc.) and output values (the target itself). Since the Informer
    model utilizes pytorch, and, therefore Tensors instead of pandas
    DataFrame, an additional argument is needed to treat each model
    in the proper way.
    ...
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day

    if target:
        if informer:
            x = torch.tensor(df.drop(columns=['date', target]).values, dtype = torch.float32)
            y = torch.tensor(df[target].values, dtype = torch.float32).unsqueeze(1)
            return x,y
        else:
            x = df.drop(columns=['date', target])
            y = df[target]
            return x, y
```

Figura 4.3: Ejemplo abstracción código con tfg_module.

Para emplear los respectivos archivos `.py` de este módulo en los archivos `.py` de los algoritmos, basta con poner las líneas de código mostradas en la Figura 4.4:

³⁸Este módulo principalmente se encarga de preprocesamiento de los datos, pero también cumple con algunas funcionalidades de procesamiento.

```
from tfg_module import my_get_time_series as mcts
from tfg_module import my_process_data as mpd
from tfg_module import my_future as mf
from tfg_module import my_get_directories as mgd
```

Figura 4.4: Importación tfg_module³⁹.

4.2 Procesamiento de los datos: Creación de modelos

Una vez los datos están preprocesados debidamente, se puede empezar a procesar estos para poder ser entrenados. La diferencia entre preprocesar los datos y procesarlos radica en que en el primero se transforman los datos en "crudo" en nuevos datos a partir de los existentes, y se extraen las características necesarias para poder entrenar los modelos. El procesamiento de los datos es el paso siguiente en el que, una vez se tienen los datos para ser entrenados, hay que decidir cómo entrenarlos.

El módulo explicado en el apartado anterior también se encarga, en cierta parte, de procesar los datos. Esto se lleva a cabo en `my_process_data.py`. Este archivo cumple con dos funciones principales: crear las características de *input* y *output* para el entrenamiento y testeo de los algoritmos —como se vio en la Figura 4.3—. También se encarga de la división de datos en entrenamiento, validación y testeo para el algoritmo de TCN, pues este usa la técnica de evaluación *train-test-split* como se comentará más en profundidad posteriormente en este mismo apartado.

Una vez ya están los datos preprocesados y procesados, se puede empezar a entrenar los algoritmos y generar los respectivos modelos. Todo este proceso se lleva a cabo en la siguiente sección de la arquitectura: véase la Figura 4.5.

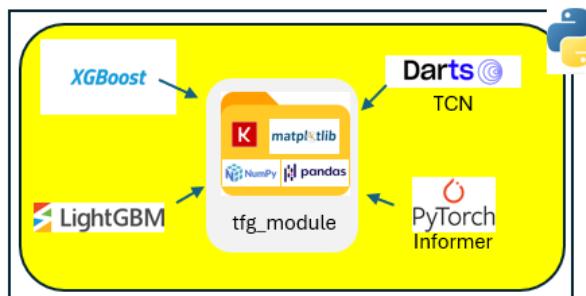


Figura 4.5: Procesamiento de los datos y creación de los modelos.

³⁹Para poder importar el módulo, la ubicación de este y de los archivos `.py` debe ser correcta. Véase el repositorio de Github para ver la estructura de carpetas.

4.2.1 Creación de modelos

Los cuatro algoritmos se han entrenado con los mismos datos. Estos datos de entrenamiento son el archivo `diario.csv` (véase Figura 4.1) con su respectivo preprocesamiento. El objetivo de la predicción será estimar las futuras compras y ventas de la empresa a partir de los datos contables. Pese a que el método de optimización de hiperparámetros para los cuatro algoritmos ha sido el mismo (GridSearch) el procesamiento de los datos ha sido algo distinto para el algoritmo TCN que para el resto.

4.2.1.1 Técnicas de evaluación

Existen enfoques distintos a la hora dividir los datos para poder entrenarlos. A esto se le conoce como técnicas de evaluación. El enfoque más común, por su sencillez de aplicación y su rapidez de ejecución, es la división en *train-test split*. Esto consiste en dividir los datos en un conjunto de entrenamiento (i.e. entrenamiento + validación) y otro conjunto de testeo para, posteriormente, poder evaluar el modelo. Esta es la manera en la que se ha entrenado el algoritmo TCN. Véase la Figura 4.6.



Figura 4.6: División entrenamiento TCN.

Por otra parte, los algoritmos de XGBoost, LightGBM e Informer han sido entrenados utilizando *K-fold Cross-Validation* (CV). Esta técnica de evaluación es más robusta que la anterior, pero computacionalmente más costosa. La ventaja de CV respecto a la anterior, es que todos los datos son parte del conjunto de entrenamiento y de testeo en algún punto del entrenamiento. Véase la Figura 4.7.

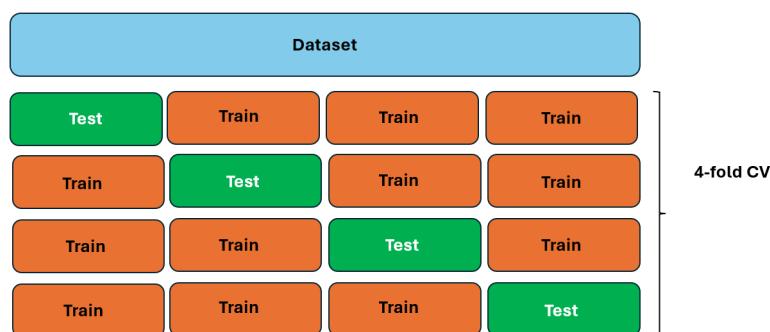


Figura 4.7: División entrenamiento XGBoost, LightGBM e Informer.

Cabe mencionar que el archivo `diario.csv` supone entrenamiento y validación en *train-test split*, mientras que en CV, es el *dataset* entero de la Figura 4.7.

Por último, es importante a recalcar que el uso de distintas técnicas de evaluación para distintos algoritmos es debido a cómo están implementadas las librerías de estos algoritmos.

4.2.1.2 Optimización de hiperparámetros

Para encontrar la mejor combinación de hiperparámetros posibles, se ha optado por utilizar el algoritmo de GridSearch. Este algoritmo es computacionalmente muy costoso pero muy eficaz a la hora de encontrar el mejor modelo posible. Recibe un conjunto de posibles valores que se desea probar para cada hiperparámetro (e.g. *learning rate*, número de épocas y otros específicos de cada algoritmo de aprendizaje automático). GridSearch lo que realizará es un entrenamiento para cada posible combinación de valores de hiperparámetros y devolverá la combinación de estos que mejor resultado ha obtenido en base a una métrica de monitorización.

Para el caso de series temporales, se ha optado por utilizar dos métricas de monitorización típicas de este problema de aprendizaje supervisado: *Mean Absolute Error* (MAE) y *Root Mean Squared Error* (RMSE). De esta manera, por cada vez que se entrene un algoritmo, se guardarán dos modelos: el modelo que mejor métrica MAE haya obtenido; así como el modelo con mejor métrica RMSE. Estos modelos no tienen por qué ser los mismos. De esto se hablará más en profundidad a continuación.

4.2.1.3 Obtención de mejores resultados

Una vez el algoritmo se ha entrenado, GridSearch devolverá el mejor modelo encontrado, con sus respectivos parámetros, en base a la métrica (o métricas) estipuladas. Puesto que se aplicarán metodologías MLOps (véase apartado siguiente), se guarda el modelo con su respectiva información relevante. Esta información es:

- El mejor modelo para cada métrica.
- Parámetros del modelo.
- Métrica MAE y RMSE del modelo (tanto para el modelo con mejor MAE, como para el modelo con mejor RMSE).
- Un `.csv` con las predicciones del modelo (próximos doce meses, aunque únicamente se utilizarán tres).

- Un .png con las predicciones realizadas por del modelo.

Para este trabajo, esta es la información que se ha considerado relevante almacenar para cada modelo. No obstante, no quiere decir que otro tipo de información adicional no fuera posible. Esta información de cada modelo quedará registrada en la nube, pero esto se comenta más en detalle en el siguiente apartado.

4.3 MLOps: monitorización y automatización del proceso

Como se ha mencionado en secciones anteriores, MLOps supone un conjunto de prácticas en el ámbito de aprendizaje automático que permite tener un control y seguimiento del ciclo de vida de los modelos. Todo esto, a ser posible, de manera automatizada. De esta manera, se consigue tener a disposición siempre el mejor modelo generado hasta el momento. También se facilita la labor de poder comparar modelos generados en un futuro con los modelos actuales para poder ir actualizándolos de manera correcta. Esto es una de las esencias del ciclo de vida de *machine learning*.

4.3.1 Monitorización de los modelos

Tener un control del ciclo de vida de los modelos es esencial para tener un versionado correcto e información acerca de ellos más allá del propio modelo. De esta tarea se va a encargar principalmente Mlflow.

Desplegar un servidor en Mlflow en local es una tarea relativamente sencilla y que se podría haber llevado a cabo en el desarrollo de este trabajo —pues no es un entorno colaborativo—. De todas maneras, con el objetivo de enfocar este trabajo lo más similar posible a un **entorno colaborativo real**, se ha optado por alojar el servidor de Mlflow en Dagshub. Véase la Figura 4.8 para situar estas tecnologías en la arquitectura de la Figura 4.1.



Figura 4.8: Mlflow alojado en Dagshub.

El único motivo de haber utilizado Dagshub, como se ha dicho, es el de poder tener un servidor de

Mlflow en la nube, simulando un entorno colaborativo real. Dagshub, a un nivel alto de abstracción, se puede ver como una extensión de Github pensada para alojar código de inteligencia artificial. De hecho, una de las maneras de iniciar un repositorio en Dagshub es vinculándolo con el repositorio de Github, como se ha hecho en este trabajo. Véase la Figura 4.9.

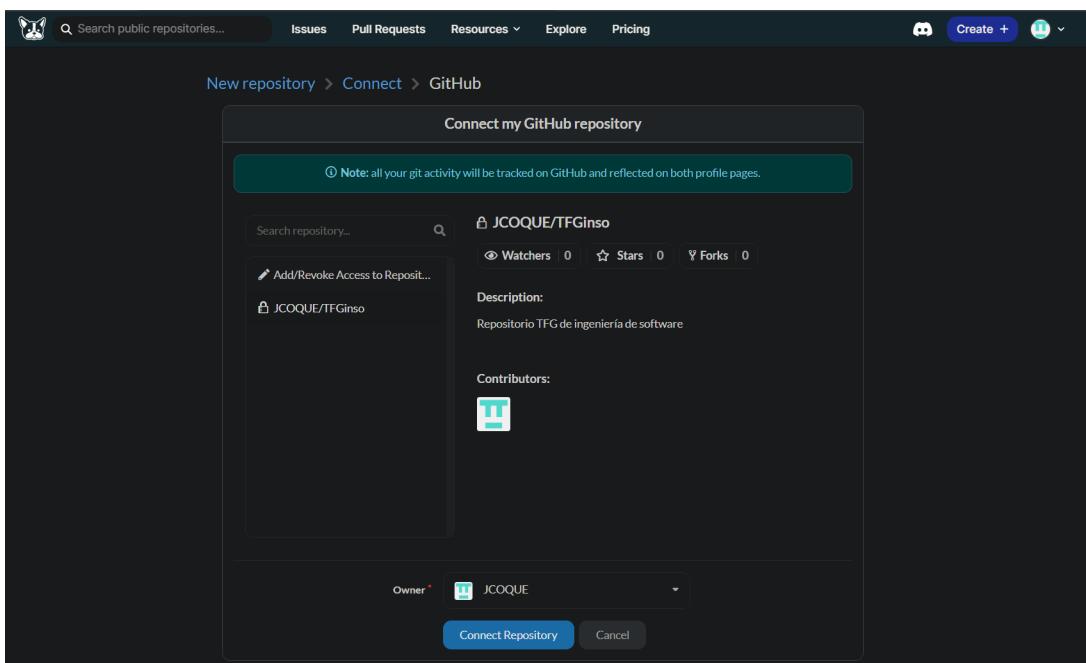


Figura 4.9: Proceso de vinculación entre el repositorio de Github con el Dagshub⁴⁰.

Por temas de seguridad, mientras que el repositorio de Github es público, el de Dagshub es privado. De nuevo, simulando un entorno colaborativo real en el que se deja el código de proyecto abierto en Github para poder recibir sugerencias de mejora en el código por parte de gente externa (*pull-requests*), mientras que el versionado de modelos se maneja de manera privada.

Una vez hecho el paso de vincular el repositorio de Github en Dagshub, uno puede ir al botón verde **Remote** donde allí encontrará la información necesaria para poder conectarse tanto al repositorio de Dagshub desde código, como al servidor de Mlflow. Esto se puede ver de manera más visual en la Figura 4.10, donde, además, también se puede ver un pequeño ejemplo de cómo iniciar Mlflow y de cómo se guardan los parámetros y métricas de un modelo.

⁴⁰Aunque en la Figura 4.9 aparezca el nombre de **TFGinso** como el nombre del repositorio de Github, este nombre de repositorio se cambió posteriormente a **TFG-ingeneria**. Este último es el nombre del repositorio actual en Github.

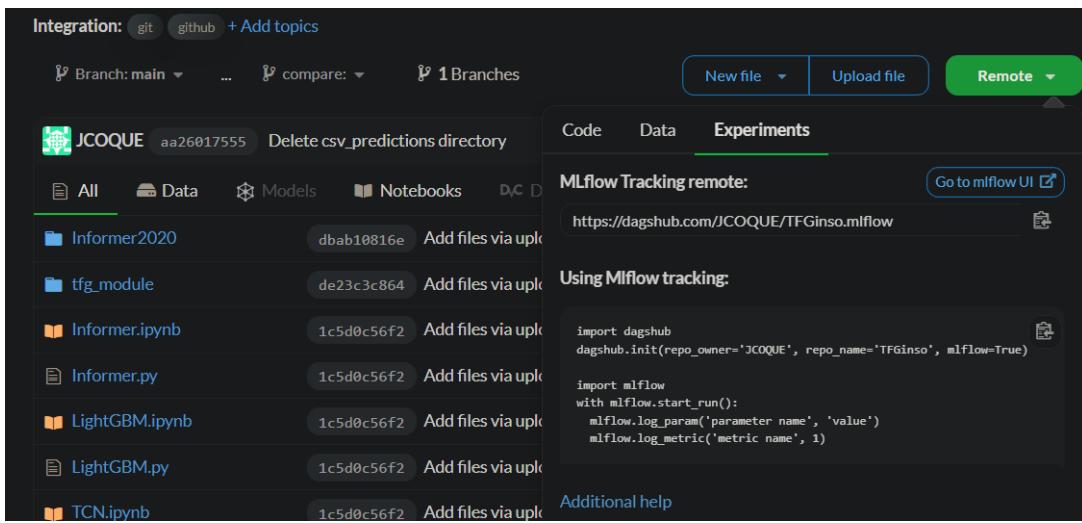


Figura 4.10: Comandos para conectarse a Dagshub y alojar modelos en Mlflow⁴¹.

En la sección de *Experiments*, tras pulsar el botón verde de **Remote**, uno puede ver la sección de:

- **Mlflow Tracking remote:** url necesaria para conectarse al servidor de Mlflow desde código.
- **Using Mlflow tracking:** comandos necesarios para inicializar el repositorio de Dagshub desde código Python.

En el caso particular de este trabajo, por cada entrenamiento de cada algoritmo, se han almacenado en Mlflow dos modelos: el modelo con mejor métrica de MAE y el modelo con mejor métrica de RMSE —como se ha comentado con anterioridad—. Adicionalmente a los modelos en sí, se han almacenado:

- Parámetros de cada modelo.
- Métricas de cada modelo.
- Esquema de características de entrenamiento (*inputs*) de cada modelo.
- Esquema de características de salida (*outputs*) de cada modelo.
- Gráfica con las predicciones a doce meses de cada modelo.
- .csv con las predicciones y su respectiva fecha.

Toda esta información irá a parar a lo que Mlflow denomina como “Experimentos”. Por lo general, un Experimento representa un conjunto de modelos entrenados y almacenados de un algoritmo en

⁴¹Este repositorio será eliminado al entregar el trabajo, por lo que la información proporcionada en la Figura 4.10 no será útil.

concreto. En este trabajo, al tener cuatro algoritmos y dos series temporales (una para Compras y otra para Ventas) se tienen un total de ocho Experimentos.

Una vez en el código se haya guardado la información pertinente de cada modelo, uno se puede ir a la interfaz gráfica de Mlflow (en el botón azul *Go to Mlflow UI* de la Figura 4.10), y visualizar allí sus respectivos experimentos. Además, Mlflow permite comparar distintos modelos (comúnmente de un mismo Experimento, pero se pueden hacer de múltiples) de una manera sencilla. Véase la Figura 4.11 donde se muestran dos modelos del Experimento “Compras LightGBM”, donde por cada modelo, aparece su esquema de datos de entrada (*Dataset*), métricas, *tags* y otra información relevante.

Run Name	Created	Dataset	Duration	User	Models	MAE	RMSE	Time	Tags
best_RMSE	3 days ago	dataset (f9e7eeb7) Training Features	13.1s	jcoque	lightgbm	7845.42	10041.46	15:24:24 03/...	LightGBM_b...
best_MAE	3 days ago	dataset (f9e7eeb7) Training Features	18.3s	jcoque	lightgbm	7836.12	10082.01	15:24:24 03/...	LightGBM_b...

Figura 4.11: Ejemplo de modelos almacenados (best_MAE y best_RMSE) para el Experimento “Compras LightGBM”.

Dentro de cada modelo, se puede encontrar información adicional acerca de este. Véase la Figura 4.12.

Model schema

Name	Type
hour (required)	integer
dayofweek (required)	integer
quarter (required)	integer

Outputs (1)

Compras (required)	float
--------------------	-------

Make Predictions

```

import mlflow
from pyspark.sql.functions import struct, col
logged_model = 'runs:/5e2f4a6d16b54d8cb22e003ac6913abb/LightGBM_Compras_best_RMSE'

# Load model as a Spark UDF. Override result_type if the model does not return double values.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model, result_type="double")

# Predict on a Spark DataFrame.
df.withColumn('predictions', loaded_model(struct(*map(col, df.columns))))

```

```

import mlflow
logged_model = 'runs:/5e2f4a6d16b54d8cb22e003ac6913abb/LightGBM_Compras_best_RMSE'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

```

Figura 4.12: Modelo almacenado en Mlflow, junto con la gráfica, datos de predicciones y el respectivo link para recuperarlo.

En la Figura 4.12, se puede ver subrayada una sección de Artefactos. Esta sección es común a todos los modelos. En ella se encuentra el modelo almacenado y cómo recuperarlo (cuadrado en rojo de abajo a la derecha). Además, también aparece subrayado la imagen .png de las predicciones y el .csv con las predicciones en valores numéricos junto con sus respectivas fechas. Por último, en el centro de la imagen, aparece en **Model schema** donde se pueden ver las características de entrada (*inputs*) que está recibiendo el modelo, así como las características de salida *output* de este mismo modelo.

Como se dijo al principio de este apartado, todo este proceso de seguimiento y monitorización del ciclo de vida de los modelos de MLOps es conveniente implementarlo y es lo que se haría en un entorno real. Pero además, se busca que todo este proceso explicado hasta ahora, desde el preprocesamiento de los datos hasta la generación y almacenamiento de los modelos, sea de manera **automatizada**. Esto se explica en el siguiente apartado.

4.3.2 Automatización del proceso

Tener el proceso descrito hasta ahora, desde el preprocesamiento de los datos hasta el almacenamiento de los modelos en MLflow, de manera automatizada no solo ahorra tiempo, sino que también posibles errores humanos. La contraparte de tener un proceso automatizado, es que se pierde cierto control en el flujo si no se utiliza una herramienta adecuada. Prefect, es la herramienta perfecta para esto, pues ofrece automatización, pero también un seguimiento del flujo de tareas de manera sencilla e intuitiva. Véase la Figura 4.13 para poder posicionar a Prefect en la arquitectura.

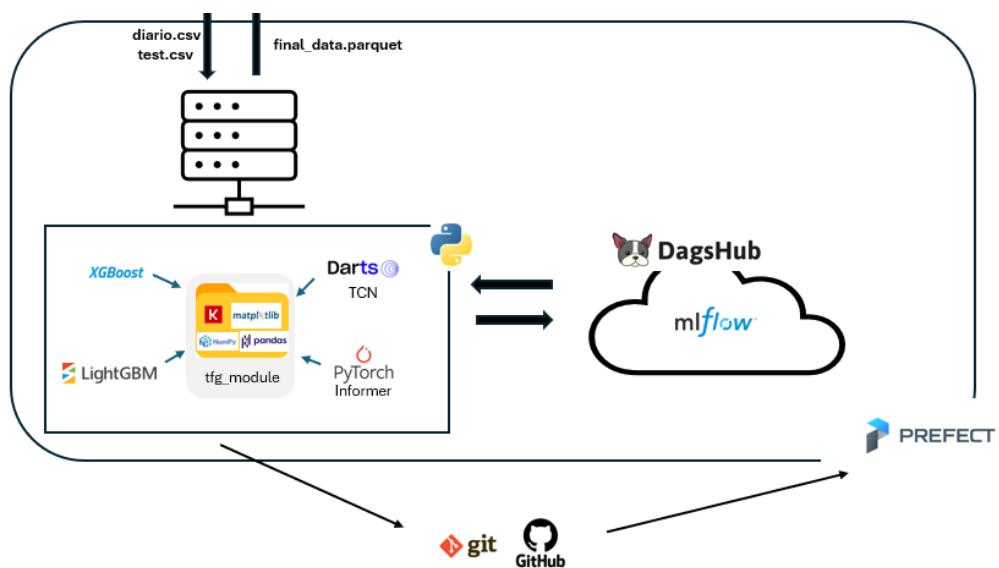


Figura 4.13: Prefect en la arquitectura del proyecto.

La Figura 4.13 trata de representar cómo todos los pasos explicados hasta ahora en esta sección los engloba Prefect. Es decir, que están automatizados por esta herramienta. Además, el código que ejecuta Prefect no lo extrae de manera local, sino del repositorio de Github. De nuevo, tratando de simular un entorno colaborativo real.

Para poder continuar, hay que entender algunos conceptos fundamentales de Prefect:

- **flow**: El código que se ejecuta.
- **task**: Divide el *flow* en trozos más pequeños para tener un seguimiento mejor de la ejecución.
- **deployment**: Proceso que permite ejecutar el código de manera automatizada y desde una localización remota (e.g. repositorio Github, como es el caso de este trabajo).
- **work pool**: Instancia donde se almacenan los flows para poder ejecutarlos.
- **worker**: Proceso encargado de ejecutar los flows (i.e. el código).

Respecto al *flow* y *tasks*, Prefect ofrece los decoradores `@flow` y `@task` que deben ser asignados únicamente a funciones (no pueden ser asociados a métodos de clase y, por eso, esta parte del código en los `.py` de los algoritmos es algo redundante, pues se crean funciones llamadas *X* que llaman a los métodos de clase de cada algoritmo con nombre *X*). Véase la Figura 4.14 para poder ver estos decoradores en el código y tener una mejor visualización acerca de estos y de cómo están implementan.

```
@task(task_run_name = 'Save results to mlflow', log_prints = True, retries = 2)
def save_mlflow(lgbm):
    print('Saving to mlflow...')
    lgbm.save_mlflow()

@flow(flow_run_name='LightGBM {target}', retries = 2)
def run(target):
    my_lgbm = MyLightGBM(target = target)
    set_attributes(my_lgbm)

    best_model_lgbm = train(my_lgbm)
    results = get_results(my_lgbm, best_model_lgbm)
    save_best_results(my_lgbm, results)

    predictions_mae = make_predictions(my_lgbm, 'best_MAE')
    save_predictions_to_csv(my_lgbm, predictions_mae, 'best_MAE')

    predictions_rmse = make_predictions(my_lgbm, 'best_RMSE')
    save_predictions_to_csv(my_lgbm, predictions_rmse, 'best_RMSE')

    init_mlflow_repository(my_lgbm)
    mlflow_connect(my_lgbm)
    save_mlflow(my_lgbm)

    return None
```

Figura 4.14: Decoradores de Prefect en el código.

En la Figura 4.14 se puede ver en la función `save_mlflow()` la redundancia de la que se ha hablado antes. Por el momento Prefect no ha implementado, al menos de manera sencilla, el utilizar sus decoradores en métodos de clase y esta fue la solución que se encontró. Respecto a los decoradores en sí, se puede ver como la función `run()` tiene asociado el decorador `@flow`. Esta función se encarga de llamar a otras funciones, cada una encargada de una cosa. Estas funciones a las que llama, cada una tiene un decorador `@task` asignado. Además, también se puede apreciar en la Figura 4.14 algunos parámetros que se han introducido a estos decoradores, como puede ser el nombre o el número de intentos que Prefect debe realizar en caso de que algo vaya mal en la ejecución del código. También está establecido en estos parámetros que los *prints* se guarden en la ejecución de Prefect.

La ejecución de un *flow* entero se vería de la siguiente manera en la interfaz gráfica de Prefect.



Figura 4.15: Flow ejecutado con Prefect.

En la Figura 4.15 se puede ver la ejecución de un *flow* entero. Este *flow* está dividido en tareas. Si uno se fija bien, las tareas mostradas con la Figura 4.15 son las mismas que las funciones a las que se llama en la función `run()` en la Figura 4.14. También, en la Figura 4.15 se pueden ver algunas flechas que salen de algunas tareas y se dirigen a otras. Esto viene a indicar la **dependencia** entre unas tareas y otras. Por ejemplo, hay una flecha que sale de la tarea *Train* y apunta a la tarea *get_results*, indicando que para ejecutar esta segunda tarea, se debe haber ejecutado la primera antes.

Algo importante a mencionar, y que puede resultar confuso, es que el orden de las tareas está indicado de izquierda a derecha. La altura a la que está cada tarea en la Figura 4.15 no indica nada.

Por último, la Figura 4.15 viene muy bien para explicar lo que se ha mencionado al principio de este apartado: Prefect, además de ser una herramienta de automatización, con ella se puede tener un control

de la ejecución del código. Y es que, en el caso de la Figura 4.15, todas las tareas aparecen en verde; indicando que todo ha ido bien. En caso de que algo hubiera ido mal, se podría saber en qué tarea el código ha fallado. Además de que Prefect proporciona un mensaje de salida con el tipo de error que ha surgido durante la ejecución. Esto para depurar el código es una ventaja y alivio para el programador.

Los otros tres conceptos, *deployment*, *work-pool* y *worker* están muy ligados entre sí. Tienen una relación muy parecida a la que puede tener un sistema pub/sub. Véase la Figura 4.16.

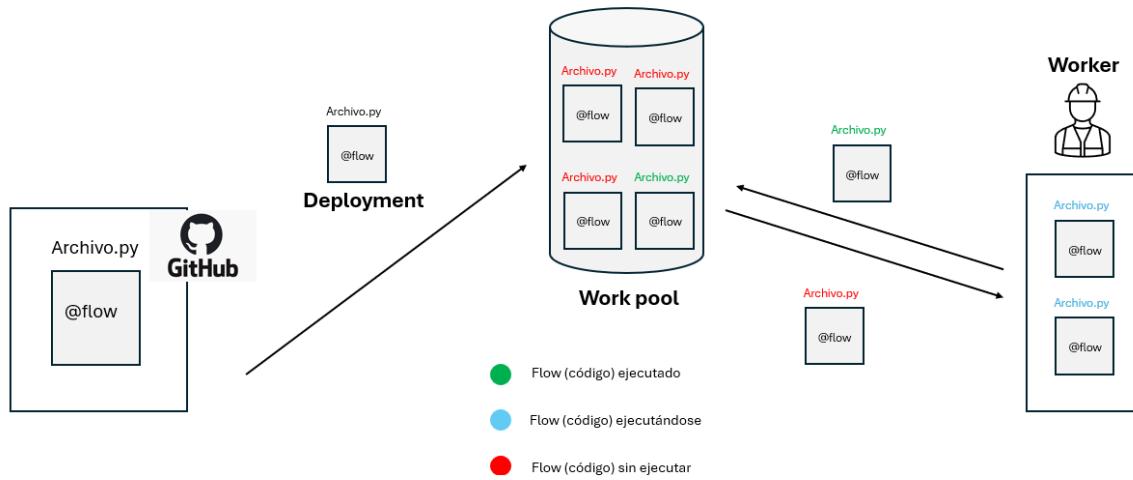


Figura 4.16: Conceptos deployment, work-pool y worker de manera visual.

Hasta ahora, se ha explicado cómo funciona la ejecución de un código con Prefect: a través de sus decoradores `@flow` y `@task`. Pero esto no es suficiente para poder automatizar esta ejecución. Para ello está el *deployment*. El *deployment* es una característica de Prefect que lleva el código a ejecutar a un escalón por encima. Esto implica que:

- El código se pueda ejecutar de manera automatizada.
- Se pueda leer el código a ejecutar desde donde sea; no necesariamente en local. En el caso de este trabajo, el código a ejecutar se extrae de Github. Se quiere dejar esto claro para que se pueda entender mejor la Figura 4.1.
- Se pueda ejecutar este código en un entorno que se desee. Idealmente en un servidor con capacidad de entrenar varios modelos al mismo tiempo de manera eficiente. En el caso de este trabajo, se ha utilizado el propio ordenador.

Para esto último, es necesario asociar el *deployment* con un *work-pool*. La Figura 4.16 trata de representar cómo el *deployment* extrae el código de Github y lo encola en su *work-pool* asociado. En

este *work-pool* es donde se aloja el código (los *flows*) que el *worker* descarga, ejecutará y devolverá el resultado de la ejecución.

Para más información de los comandos empleados para el despliegue de la arquitectura mostrada en la Figura 4.16, están todos explicados en el [README.md](#) de Github en la sección *Commands and other important info to keep in mind*.

Nota

Aunque no se ha hecho por simplificar contenido de trabajo, se podrían haber duplicado todos los archivos `.py` de los algoritmos y haber configurado los *deployments* para ejecutar, de manera programada, los algoritmos para entrenar la serie temporal de Compras y, por otra parte, para entrenar la serie temporal de Ventas. En caso de que se requiriese, sería muy sencillo implementarlo.

4.4 Comparativa de modelos

En este apartado se explicará la comparativa de modelos. Antes que nada, mencionar que debido a la falta de tiempo, pero sobre todo de recursos, únicamente ha sido posible comparar los algoritmos de LightGBM y TCN. Era de esperar que XGBoost e Informer, ambos por su complejidad de arquitectura, iban a ser los que más tiempo iban a demandar. Estos algoritmos se tuvieron que dejar de entrenar cuando llevaban más de dos días entrenándose y su consumición de recursos provocaban lentitud en el ordenador, dificultando la realización de otras tareas. En un entorno con una máquina más potente, esto no hubiera supuesto ningún problema.

Para evaluar los modelos, se hace uso del archivo `test.csv`. Esto es para poder comparar datos reales con datos que el modelo no ha visto —en el entrenamiento—, pero sí ha predicho.

Tanto para Compras como para Ventas se han comparado cuatro modelos en total:

- LightGBM best_RMSE
- LightGBM best_MAE
- TCN best_RMSE
- TCN best_MAE

En ambos casos, Compras y Ventas, TCN ha obtenido un rendimiento mayor, llegando a ajustarse muy

bien a los datos de testeo en la serie temporal de Compras. En Ventas, por otra parte, los resultados no han sido tan buenos.

4.4.1 Compras

Véase en la Tabla 4 para poder ver las métricas de los modelos para Compras.

Compras		
LightGBM best_RMSE	RMSE	10041.46
	MAE	7845.42
LightGBM best_MAE	RMSE	10082.01
	MAE	7836.12
TCN best_RMSE	RMSE	6317.86
	MAE	6756.17
TCN best_MAE	RMSE	7589.41
	MAE	4976.97

Tabla 4: Comparativa de métricas entre distintos modelos para Compras.

Las métricas, tanto de RMSE como de MAE, no indican si un modelo es bueno o no. Es decir, un MAE de 0.01, por ejemplo, puede ser mucho o poco dependiendo de los datos del problema. Lo mismo ocurre RMSE. Sin embargo, estas métricas sí que sirven para poder determinar, *a priori*, si un modelo es mejor que otro. Estas métricas, al contrario que ocurre con otras muchas, se trata de buscar que su valor sea lo mínimo posible. Con esto mencionado, a partir de los datos mostrados en la Tabla 4, a primera vista se puede ver que los modelos generados por este algoritmo TCN parecen mejores que los algoritmos generados por este algoritmo de LightGBM.

Se hace un inciso en que esta comparativa no es definitiva, pues también se deben contrastar los datos reales con los datos predichos en una gráfica. Esto es porque las series temporales tienen fluctuaciones, tendencias y estacionalidades. Lo que se trata de buscar con las predicciones de un modelo, es que este modelo haya “aprendido” estas características de la serie temporal dada y lo haya sabido plasmar

en sus predicciones. De esta manera, no solo un modelo es mejor que otro si obtiene, en este caso, unas métricas con un valor menor, sino también si ha sido capaz de aprender de una manera más eficaz estas propiedades mencionadas recientemente de la serie temporal. Véase la Figura 4.17 para ver de manera gráfica los datos de testeo (reales) con las predicciones realizadas por cada modelo.

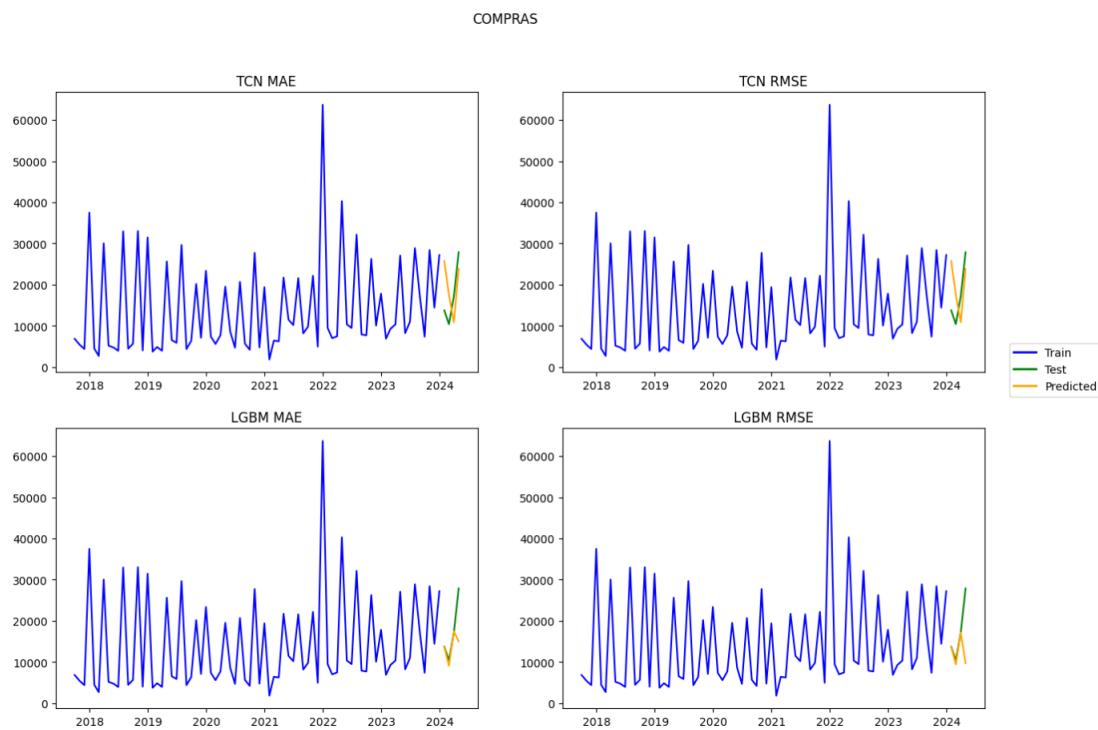


Figura 4.17: Datos de testeo junto con las predicciones de cada modelo de Compras visualizado de una manera gráfica.

En la Figura 4.17 se puede ver de manera mucho más clara que TCN, **en este caso**, predice mejor que el algoritmo LightGBM. No solo obtiene mejores métricas —como se vio en la Tabla 4—, sino que también se ajusta de manera mucho más precisa a los datos de testeo. Además, es interesante ver cómo, para la serie temporal de compras, TCN ha aprendido muy bien los patrones y tendencias de la serie temporal. Las fluctuaciones las ajusta muy bien.

También cabe mencionar que, al ser las series temporales la consecuencia de un proceso estocástico, más modelos deberían generarse, con sus respectivas comparativas para poder afirmar que el algoritmo TCN es superior a LightGBM en este caso. Las conclusiones de la comparativa, por tanto, son que para los modelos generados, el algoritmo TCN se ajusta mejor que el de LightGBM. Una comparativa más exhaustiva suponía un tiempo de computación inviable con los recursos de los que se disponía.

Debido a que ambos modelos de TCN son muy parecidos (en cuanto a parámetros y métricas obtenidas), las predicciones las realizan de manera muy similar. Se podrían emplear ambos algoritmos para generar una única predicción. En el caso de este trabajo en concreto, se ha optado por elegir al modelo **TCN best_RMSE**, ya que se ajusta ligeramente mejor a los datos de testeo mostrados en la Figura 4.17. Las predicciones realizadas por este algoritmo serán las predicciones mostradas de manera visual en el último apartado de esta sección.

4.4.2 Ventas

La misma lógica de comparativa que se ha explicado en Compras, también se aplica para Ventas. De nuevo, se van a mostrar en una tabla las distintas métricas para cada modelo entrenado —que son los mismos⁴² que en el apartado anterior, pero para la serie temporal de Ventas—, así como una gráfica que mostrará los datos de testeo junto con las predicciones realizadas por cada modelo. Para ver las métricas de cada modelo entrenado en este trabajo, véase la Tabla 5.

Ventas		
	RMSE	4320.92
LightGBM best_RMSE	MAE	3442.7
	RMSE	4321.27
LightGBM best_MAE	MAE	3442.56
	RMSE	3086.34
TCN best_RMSE	MAE	4354.17
	RMSE	3096.51
TCN best_MAE	MAE	2407.42

Tabla 5: Comparativa de métricas entre distintos modelos para Ventas.

De nuevo, el algoritmo de TCN, en una primera instancia, parece haber sido más eficaz a la hora de

⁴²Los modelos en sí son distintos. Se han entrenado con series temporales distintas. Lo que es igual son los algoritmos obtenidos y el nombre de sus modelos.

estimar la variable objetivo: Ventas en este caso. Sin embargo, este ejemplo viene muy bien para recalcar lo que se ha mencionado en el apartado anterior: MAE y RMSE no son métricas que indiquen si un modelo es bueno o no. En este caso, el valor de las métricas es más bajo que para la variable objetivo de Compras del apartado anterior. En una primera instancia, uno podría pensar que estos modelos habrán obtenido mejores resultados; pues sus métricas son mucho más bajas. Pero, como se verá a continuación, para nada es así. Véase la Figura 4.18 para ver gráficamente los datos de testeo junto con las estimaciones realizadas por cada modelo.

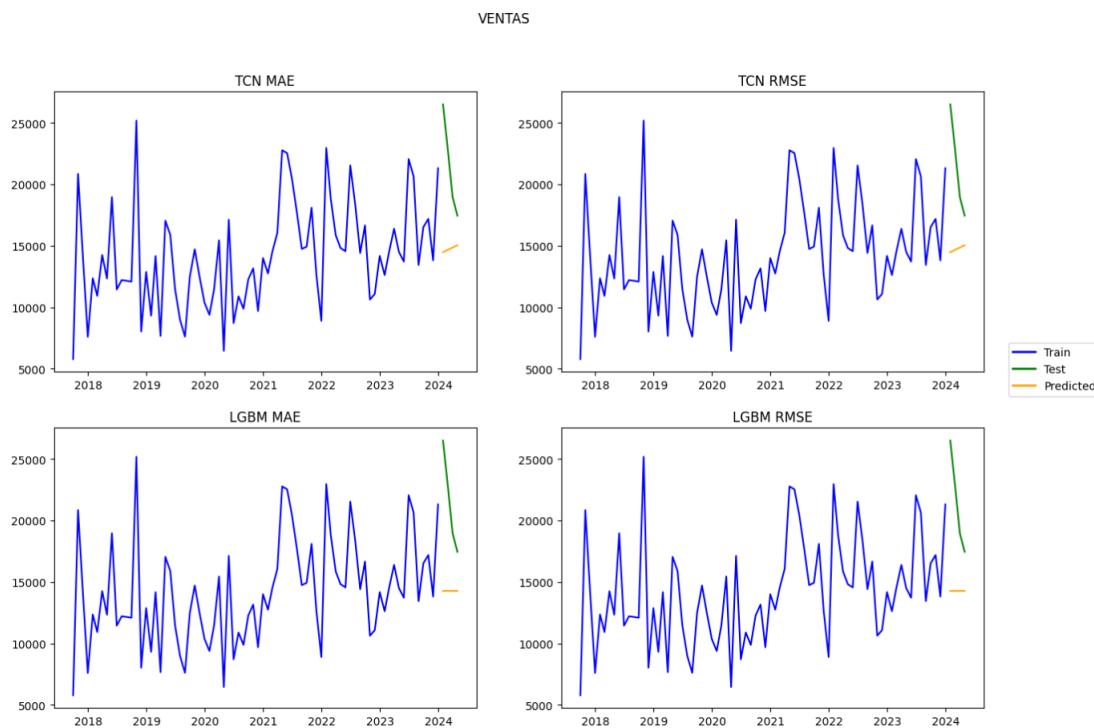


Figura 4.18: Datos de testeo junto con las predicciones de cada modelo de Ventas visualizado de una manera gráfica.

En este caso, por lo que muestran las gráficas, ningún algoritmo ha sido capaz de aprender bien las fluctuaciones, tendencias y estacionalidad de la serie temporal. Si uno compara ambas series temporales, esta segunda es mucho más compleja de tratar de predecir: no tiene un patrón tan claro como la serie temporal de Compras. Esto se deba seguramente a cómo se lleva internamente la contabilidad dentro de la empresa y de la manera en que se hacen los cobros y pagos en ella.

Se podrían estudiar los modelos más en profundidad, cómo mejorarlo, tratar de averiguar por qué no han predicho tan bien como en Compras, plantear otros enfoques de entrenamiento como puede ser Optuna, etc. Esto es un trabajo de semanas, y en este trabajo, por falta de tiempo, no se ha cubierto.

Debido a las métricas obtenidas, se ha elegido el modelo **TCN best_MAE** como modelo candidato a que sus predicciones sean las que aparezcan en las visualizaciones finales.

4.4.3 Datos finales

Una vez se han elegido los modelos candidatos de Compras y Ventas para mostrar sus estimaciones, lo que se hace es juntar toda la información, la presente y la futura, en un **.parquet** que se envía a Azure Blob Storage. Estos datos finales suponen la concatenación de toda la información disponible hasta el momento. Esto es el **diario.csv**, el cual contiene la mayoría de datos. El archivo **test.csv** que tiene el mismo formato y misma estructura que el archivo anterior, pero con datos más actuales. Y, finalmente, después de concatenar estos dos archivos mencionados, se añaden las predicciones de Compras y Ventas de los respectivos modelos elegidos en los apartados anteriores.

Estos datos concatenados forman el archivo **final_data.parquet**. Este archivo supone los datos a almacenar en Azure SQL Database y, por ende, los que utilizará Power BI en sus visualizaciones. El formato **.parquet** se ha elegido por ser un formato que almacena los datos de manera más eficiente y, por tanto, ocupa menos que un **.csv** u otras extensiones; perfecto para entornos de *Big Data*.

4.5 Almacenamiento de los datos

En este apartado se explicarán los servicios utilizados en este trabajo para el almacenamiento de los datos. Todo se ha llevado en Azure. De esta manera, los datos están alojados en un lugar seguro y accesible desde cualquier parte del mundo para una persona con los permisos adecuados. De nuevo, al igual que se ha mencionado en apartados anteriores, **simulando un entorno real colaborativo**. Véase la Figura 4.19.

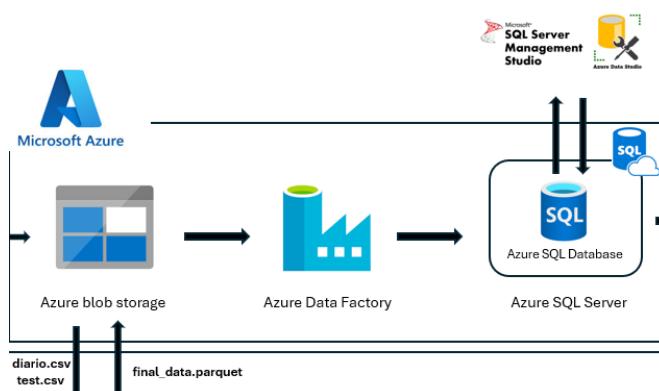


Figura 4.19: Arquitectura Azure.

Aunque no se haya mencionado hasta ahora, todo empieza aquí: en Azure Blob Storage. Se trata de un *datalake* en donde se puede guardar casi cualquier tipo de dato; estructurado, semi-estructurado o no estructurado. Es de este origen de donde el archivo `my_azure.py`, explicado en el apartado de preprocessamiento, extrae los datos (*pull*) y, asimismo, envía los datos (*push*) una vez estos ya contienen la información necesaria para poder ser visualizados (como se mencionó en el apartado anterior). Véase la Figura 4.20 en donde se muestra una imagen con el contenido de Azure Blob Storage.

The screenshot shows the Azure Blob Storage interface for the container 'containertfginso1'. The left sidebar contains navigation links: Overview, Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Access policy, Properties, and Metadata. The main area displays a table of blobs:

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
<input type="checkbox"/> diario.csv	5/16/2024, 12:40:01 ...	Hot (Inferred)		Block blob	2.8 MiB	Available
<input type="checkbox"/> final_data.parquet	6/3/2024, 6:38:03 PM	Hot (Inferred)		Block blob	619.99 KiB	Available
<input type="checkbox"/> test.csv	5/31/2024, 8:33:43 PM	Hot (Inferred)		Block blob	283.94 KiB	Available

Figura 4.20: Contenido de Azure Blob Storage.

Estos archivos se alojan dentro de un contenedor llamado “containertfginso1” —como se puede ver arriba a la izquierda de la Figura 4.20—. Esto es importante tenerlo en cuenta para secciones posteriores, en donde se hará referencia a este contenedor para seleccionar la información que se desea copiar desde Azure Blob Storage a Azure SQL Database (`final_data.parquet` en este caso).

Ahora, aunque se discutirá más en profundidad en la siguiente sección, por el momento basta con saber que a la aplicación Power BI le conviene que los datos vengan de una base de datos SQL y no de un *datalake*. Esto es porque de esta manera se actualizan los datos de manera más eficiente. Para poder llevar los datos desde Azure Blob Storage en formato `.parquet` a una tabla de base de datos SQL, se utiliza Azure Data Factory. Este servicio ofrece múltiples **actividades** para crear *pipelines* y transportar datos de un origen a un destino. La actividad de interés para este trabajo es *Copy Activity* que se encarga de copiar datos desde Azure Blob Storage hasta Azure SQL Database. Para ello es necesario crear lo que Azure llama *Linked Services* y *Datasets* —tanto para Azure Blob Storage como para Azure SQL Database—. Por tanto, una arquitectura de Azure que muestre de manera más precisa lo que está ocurriendo es la siguiente; véase la Figura 4.19:

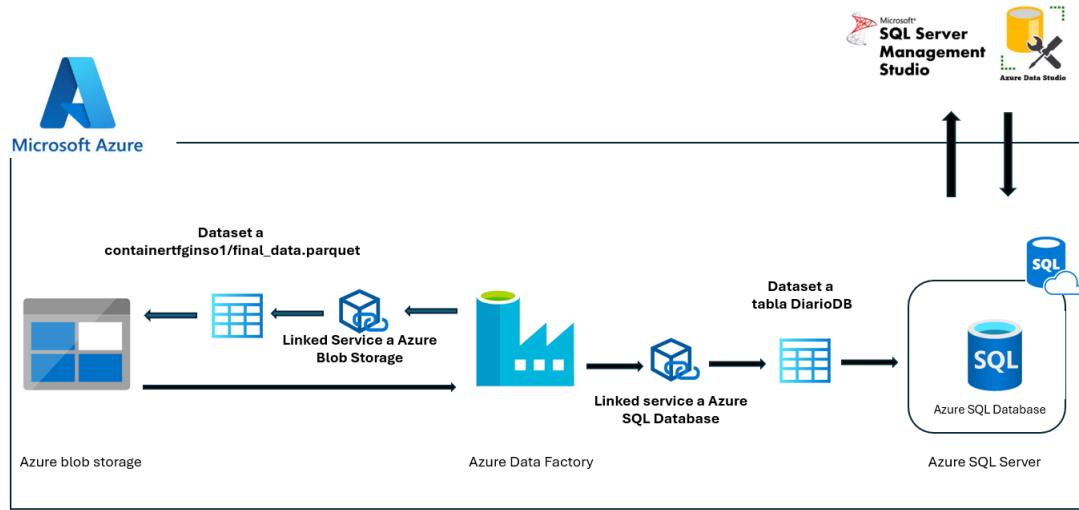


Figura 4.21: Arquitectura pura de Azure.

Puesto que Azure Data Factory se puede conectar a varios orígenes de datos, los *Linked Services* le indican a qué origen de datos debe conectarse y cómo hacerlo. En la Figura 4.21 hay dos *Linked Services* distintos: uno para Azure Blob Storage y otro para Azure SQL Database.

Una vez hecho este paso, se debe indicar a Azure Data Factory qué datos interesa recopilar del origen y hacia dónde interesa llevarlos en el destino. Para ello están los *Datasets*. El *Dataset* de Azure Blob Storage le indica a Azure Data Factory que debe coger el archivo `containertfginso1/final_data.parquet` de Azure Blob Storage. Por el contrario, el *Dataset* de Azure SQL Database le indica a Azure Data Factory hacia dónde debe mover estos datos: en este caso a la tabla `diarioDB` de la base de datos SQL creada a través de la herramienta *SQL Server Management Studio*. En la Figura 4.22 se muestra la configuración de cada uno de los *Datasets* empleados en este trabajo.

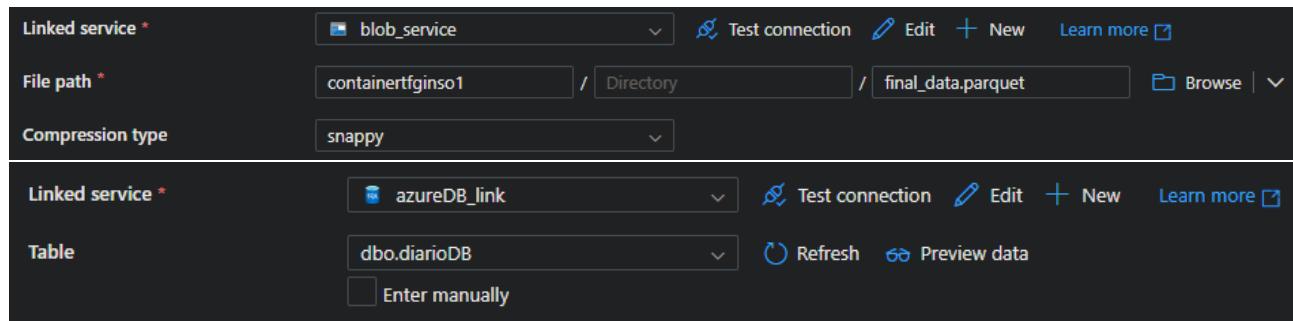


Figura 4.22: Configuración Datasets.

En la Figura 4.22 se puede ver cómo cada *Dataset* tiene su propio *Linked Service* asociado. Así como también están especificados los orígenes y destinos de los datos de una manera más concreta.

Dos últimas cosas a tener en cuenta es la configuración de “mappeo” entre los datos de origen y destino —cerciorarse de que están bien—. Esto se tuvo que modificar en el trabajo. Véase la Figura 4.23.

The screenshot shows the 'Mapping' section of the Azure Data Factory interface. It displays a table with four columns: Source, Type, Destination, and Type. The Source column lists various columns like ID, Fecha, Cuenta, etc. The Destination column lists corresponding columns in a destination dataset. The 'Type' column shows the data types for both source and destination. There are also icons for 'Import schemas', 'Preview source', 'New mapping', 'Clear', 'Reset', and 'Delete' at the top.

Source	Type	Destination	Type
ID	123 INT64	ID	123 int
Fecha	123 datetime	Fecha	123 datetime
Cuenta	123 UTF8	Cuenta	123 varchar
NoCuenta	123 UTF8	NoCuenta	123 varchar
Debe	123 DOUBLE	Debe	123 decimal
Haber	123 DOUBLE	Haber	123 decimal
Entidad	123 UTF8	Entidad	123 varchar
Compras	123 DOUBLE	Compras	123 decimal
Ventas	123 DOUBLE	Ventas	123 decimal
NoGrupo	123 UTF8	NoGrupo	123 varchar
type	123 UTF8	type	123 varchar

Figura 4.23: Mappeo entre origen y destino.

Por otra parte, la configuración a introducción de datos en la tabla SQL en modo **upsert** (*update + insert*) —para solo copiar datos nuevos y/o actualizar los que ya había—. Para determinar si un dato es nuevo (y por tanto debe introducirse) o, por el contrario, ya está en la base de datos (y por tanto debe actualizarse), se le indica a Azure Data Factory que debe fijarse en la columna ID de los datos. De nuevo, esta configuración está pensada para entornos *Big Data* en los que hay una gran cantidad de datos y se trata de optimizar este proceso lo máximo posible. Véase la Figura 4.24.

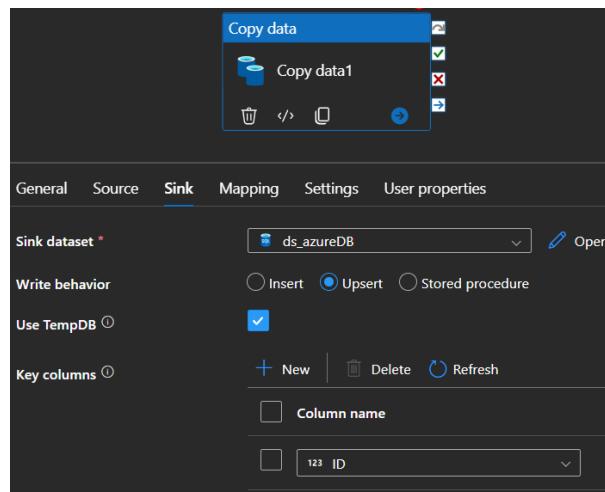


Figura 4.24: Configuración de upsert con su respectiva columna ID.

Con todo esto, añadido a la *Copy Activity*, se zure SQL Database. En la Figura 4.25 se muestra un esquema con el resultado de realizar esta actividad de copia.

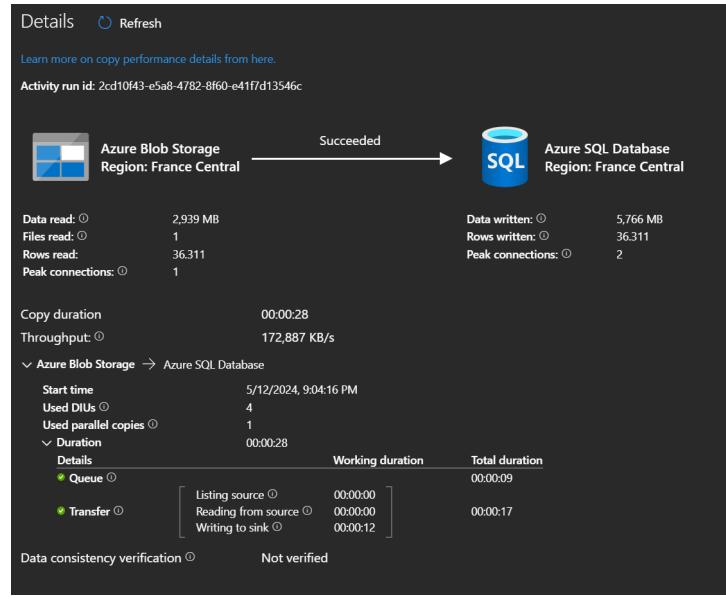


Figura 4.25: Esquema de la Copy Activity realizada.

Esta actividad se ha configurado para que se realice de manera concurrente diariamente. Es decir, hay otra capa de automatización en el proyecto.

4.5.1 SQL Server

Hasta ahora se ha hablado de Azure SQL Database pero no de Azure SQL Server. La función de Azure SQL Server es gestionar y alojar una o incluso varias bases de datos SQL (en el caso de este trabajo una). Otra utilidad muy importante que proporciona un *endpoint* para poder conectarse a la base de datos; tanto desde Azure Data Factory, como desde SQL Server Management Studio, como desde Power BI. Véase la Figura 4.26.

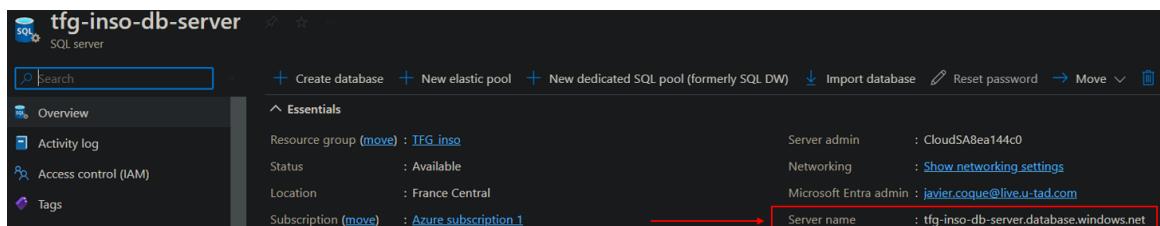


Figura 4.26: SQL Server endpoint.

Por último, y no por ello menos importante, mencionar el uso de la herramienta —ya mencionada con

anterioridad— SQL Server Management Studio. Con ella y comandos básicos de SQL se ha creado un usuario, y la tabla en bases datos para el proyecto. Para más información de los comandos empleados en Github ir a Desarrollo/SQL_scripts.

Nota

Para no alargar en exceso este apartado se dejan referencias de configuración de todo lo mencionado hasta ahora: Azure Data Factory, Azure Blob Storage, etc. Para más información, por favor diríjase al [README.md](#) de Github en la sección *Tools/Azure*.

- Documentación extra-oficial: (Tech, 2024a), (Tech, 2024b), (Tech, 2024c), (Tech, 2024d), (TechBrothersIT, 2024), (Knowledge, 2024).

Una vez los datos ya están en el lugar desde donde Power BI los extraerá, únicamente queda recogerlos y mostrarlos.

4.6 Visualización de los datos

Para la visualización de los datos se ha empleado la herramienta de Power BI por su sencillez y su interactividad entre gráficas. Con esta herramienta se ha realizado el *dashboard* final. Véase la 4.27 para posicionar esta herramienta dentro de la arquitectura del proyecto mostrada en la Figura 4.1.



Figura 4.27: Última sección de la arquitectura del proyecto: Power BI.

Power BI se conecta a tres fuentes de datos: la tabla SQL mencionada en el apartado anterior y dos Excel adicionales para aportar cierta inteligencia temporal y de cuentas contables (Calendario y PGC respectivamente). Esto es, por ejemplo, para ordenar los meses correctamente en los datos (Power BI los ordena alfabéticamente: abril, agosto, etc.). Respecto a las cuentas bancarias, para poder relacionar la cuenta 100 con Financiación Básica y demás.

Para añadir dinamismo a las gráficas y que el usuario tenga cierto control sobre qué gráficas quiere ver, se han introducido lo que Power BI llama como *parámetros*. Además, para poder sacarle provecho a

los datos, se han incluido lo que Power BI llama *medidas* para obtener ciertos resultados deseados a partir de los datos existentes. Estas medidas están escritas en código DAX (*Data Analysis Expressions*). Para poder ver mejor cómo están relacionadas unas tablas con otras y los parámetros utilizados, véase la Figura 4.28.

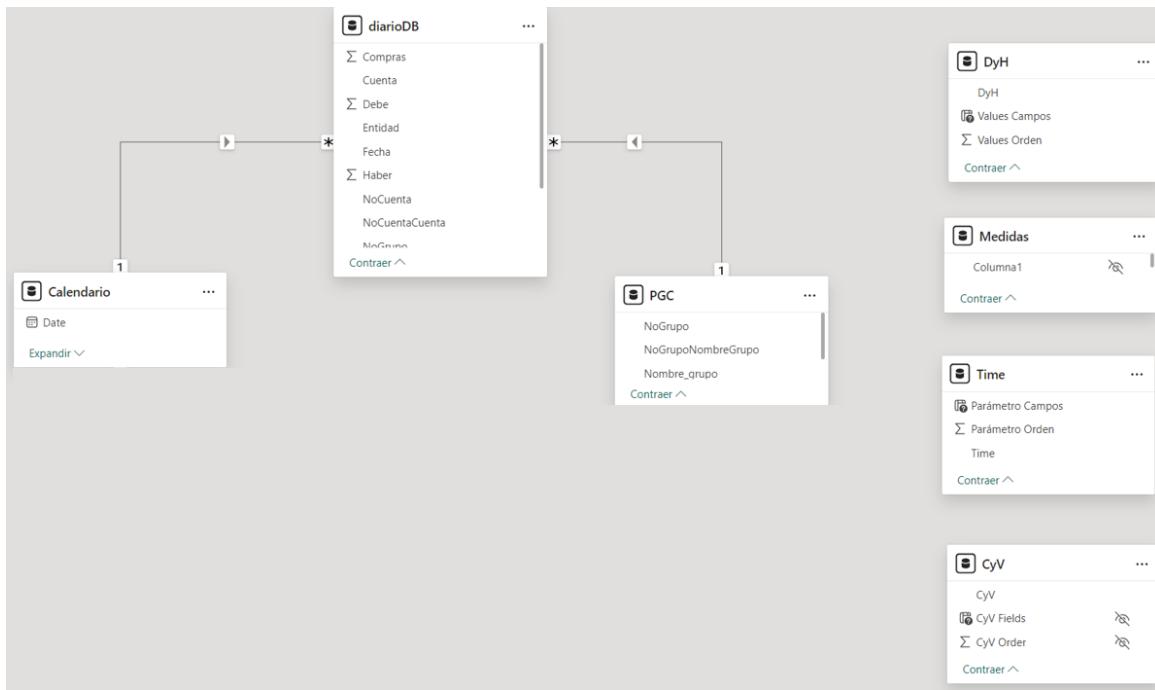


Figura 4.28: Esquema relacional Power BI.

En la Figura 4.28 se muestran cuatro tablas: **diarioDB**, **Calendario**, **PGC** y **Medidas**. Esta última se trata de una tabla *dummy* que contiene una columna vacía. Esto es una práctica común en Power BI, con el único objetivo de tener todas las medidas creadas con DAX en una misma tabla —y así no tenerlas repartidas por distintas tablas—. Además, en la Figura 4.28 se puede ver cómo las tablas **Calendario** y **PGC** están relacionadas con la tabla **diarioDB**. Por ser un poco más concretos, la tabla **Calendario** está relacionada con la columna **Time** de **diarioDB**, mientras que la tabla **PGC** con la columna **NoGrupo**.

4.6.1 Actualización incremental

La actualización incremental —o *incremental refreshing* en inglés— es una característica que ofrece Power BI para únicamente actualizar datos nuevos. Los datos clasificados como nuevos se definen en una política de configuración de la propia actualización incremental. Esta característica, en un entorno *Big Data* con una inmensa cantidad de datos, es **esencial**.

El exclusivo motivo por el que se ha decidido **implementar una base de datos SQL** en el desarrollo de este trabajo es porque para poder hacer una **actualización incremental**, el sistema de alojamiento desde donde Power BI extrae los datos a utilizar, debe permitir una operación llamada *query folding*. Esta operación se trata de una técnica de optimización en el contexto ETL (Extracción, Transformación y Carga de datos), en la que el filtrado de datos se hace en el sistema de origen en lugar de en el sistema de alimentación. Es decir, que el filtrado de datos que se pueda realizar como consecuencia de la actualización incremental, se hace —en el caso de este trabajo— en la tabla SQL, en lugar de en Power BI. Esta operación no está soportada por todos los sistemas de almacenamiento de datos; pero por bases de datos SQL sí.

Para realizar una actualización incremental sobre una tabla de Power BI, hay que definir previamente unos parámetros de tiempo en una columna de tipo *Time* en la tabla `diarioDB` para el caso concreto de este trabajo (Microsoft, 2024). Una vez hecho esto, se puede proceder configurar esta actualización incremental. Véase la Figura 4.29.

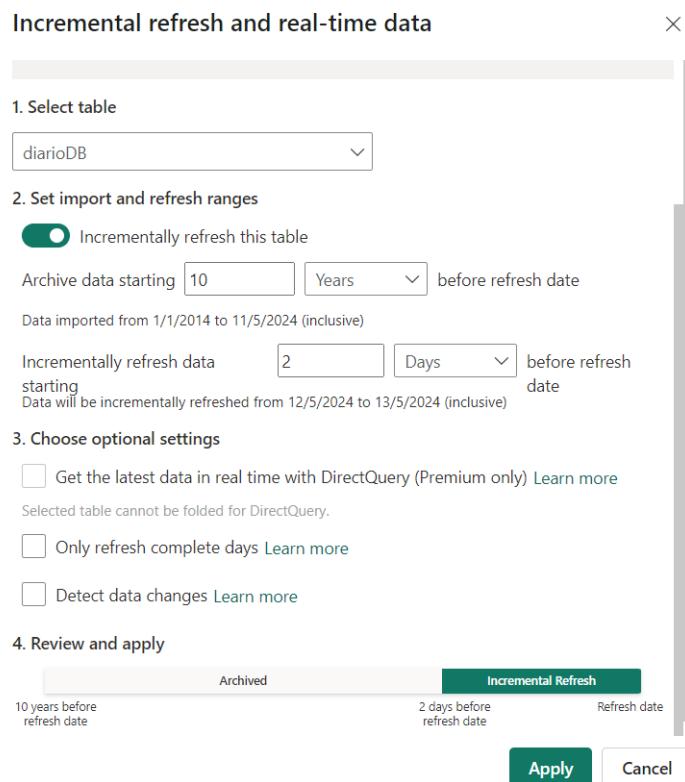


Figura 4.29: Política de configuración de actualización incremental.

En la Figura 4.29, justo encima de los botones *Apply* y *Cancel* se puede entender muy bien la

configuración establecida. Se van a leer datos de los últimos diez años y a cargar únicamente aquellos datos que se hayan modificado en los últimos dos días; manteniendo el resto de datos intactos. Esta configuración, en un entorno real debería configurarse según la situación, tipo de datos, arquitectura implementada, etc.

4.6.2 Resultado final

El resultado final son cuatro pestañas; cada una con un *dashboard* distinto. El primer *dashboard* es uno general que, adicionalmente a las tablas mencionadas con anterioridad, hace uso del parámetro **CyV** mostrado en la Figura 4.28. El parámetro **CyV** permite al usuario que decida sobre si las gráficas muestran información de Compras o de Ventas de la empresa. Véase la Figura 4.30.

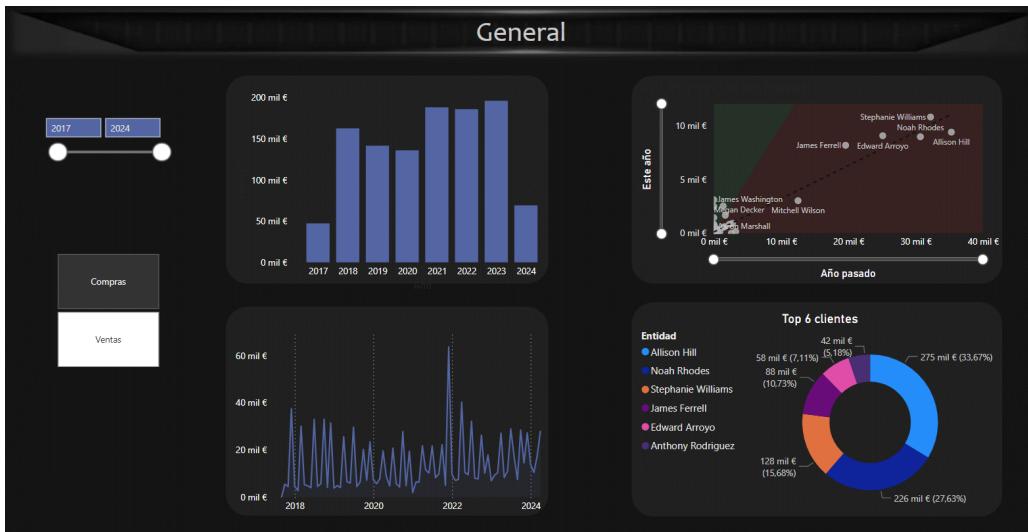


Figura 4.30: Pestaña General dashboard.

En la Figura 4.30 se trata de mostrar un *dashboard* general con información respecto a Compras o Ventas; algo que se puede especificar en los botones de la izquierda. Además de esto, el usuario también puede elegir entre el periodo de tiempo que más le interese. El color del *dashboard* elegido es **negro**, pues añade elegancia y simplicidad a las gráficas.

En cuanto a las gráficas, se ha optado por elegir dos clásicas, de barras y de línea, para mostrar el parámetro seleccionado por el usuario de manera cuantitativa. Adicionalmente a estas gráficas, se ha considerado interesante mostrar una gráfica de tipo *scatter* en la que se comparan las Compras o Ventas de los clientes un año en concreto, con respecto al año anterior. De esta manera, se puede ver si un año dado está haciendo clientes a los que se está comprando o vendiendo en mayor o menor cantidad que respecto al año anterior. Esta gráfica ayuda de manera sustancial para tener un mejor seguimiento de los

clientes, por si hubiera que llamar a alguno de ellos —o cualquier otra razón comercial que va más allá en este trabajo—. También los ejes de esta gráfica son modificables por el usuario. Por último, abajo a la derecha, se muestra una gráfica de tarta mostrando los *Top 6 clientes* del respectivo parámetro. De nuevo, por motivos comerciales principalmente, pero también para otros muchos ámbitos de la empresa.

El segundo *dashboard* ofrece una comparativa entre Compras y Ventas, así como una comparativa de cada una de estas medidas respecto al año pasado hasta el día actual. Por último, ofrece una visión rápida y eficaz de todas las diferentes cuentas, su Debe y Haber, para cada año. De esta manera, si el método de la partida doble se hubiera empleado mal en algún momento, es muy fácil de ver. Véase la Figura 4.31 en donde se muestra este segundo *dashboard*.



Figura 4.31: Pestaña Cuentas dashboard.

En la pestaña de la Figura 4.31 el usuario también puede elegir una ventana temporal sobre la que mostrar los datos contables. Además, abajo a la izquierda, se muestra una gráfica de preguntas y respuestas. Esta gráfica hace uso de inteligencia artificial incorporada por Power BI para tratar de resolver preguntas acerca de los datos. Arriba a la derecha de esta gráfica, se encuentran 3 tarjetas mostrando información acerca de Compras, Ventas y neto. Las dos primeras muestran adicionalmente una comparativa entre las Compras y Ventas del máximo año seleccionado respecto al año anterior. Esta comparativa es interesante porque, en caso de que se haga del año actual con respecto al año anterior, se tiene en cuenta el día del actual año para la comparativa con el año previo⁴³. Las dos gráficas inferiores, al igual que en la pestaña anterior, son gráficas de barras y línea. La diferencia

radica en que estas gráficas muestran las dos variables a la vez, permitiendo una comparativa entre ellas de una manera mucho más clara y efectiva. Por último, arriba a la derecha, se encuentra la gráfica más importante de este *dashboard*: la gráfica que muestra información de Debe y Haber de las cuentas contables de cada año. Esta gráfica es de inmensa utilidad para un contable, pues muestra de manera muy clara si se está cumpliendo con el método de la partida doble de contabilidad. Este, como se mencionó en la sección 2, es un principio fundamental de la contabilidad y debe cumplirse **siempre**.

El tercer *dashboard* utiliza gráficas de análisis avanzado no propias de Power BI, sino de Zebra BI. Este *dashboard* es el menos llamativo estéticamente debido a que la versión gratuita de Zebra BI ofrece pocas posibilidades de cambiar la estética de las gráficas. Aun así, las gráficas mostradas en esta pestaña ofrecen un análisis muy potente. Por un lado, las cuentas contables de un año en concreto con respecto al año previo, dejando al usuario elegir entre Debe y Haber con el parámetro **DyH** y la estacionalidad con el parámetro **Time** (Figura 4.28). Por otro lado, en las Compras y Ventas (según determine el usuario) y, de nuevo, con respecto al año previo a partir de un año seleccionado. Véase la Figura 4.32.



Figura 4.32: Pestaña Análisis dashboard.

En la Figura 4.32 se pueden ver las dos gráficas mencionadas anteriormente. También, se permite al usuario elegir el año sobre el que este quiere que le muestre la información. Esto, teniendo en cuenta que ambas gráficas mostradas realizarán una comparativa entre el año seleccionado y el anterior.

⁴³De esta manera, si se está a ocho de agosto, realiza una comparativa hasta el ocho de agosto del año pasado. Esto permite una comparación “ limpia” entre dos años si uno de ellos no ha terminado todavía.

Además, una característica importante de la gráfica de la derecha en la Figura 4.32, es que cuenta con un botón (solo se ve al pasar el ratón por encima) que permite mostrar otro tipo de gráficas analíticas.

Ambos gráficos de Zebra BI mostrados en este *dashboard* son muy potentes analítica y comparativamente. Estos gráficos son los más utilizados, por empresas que utilizan Power BI, a nivel de finanzas y ventas (Zebra BI, 2024). Este trabajo está restringido —a nivel de estética, pero sobre todo de aporte de mayor información— a que estas gráficas son de pago. De esta manera, modificarlas y obtener más información acerca de ellas, es una tarea que va más allá del contenido de este trabajo.

Por último, está el *dashboard* de BA que contiene las predicciones comentadas en el apartado *Comparativa de modelos* de esta misma sección del trabajo. Esta pestaña ofrece información acerca de las previsiones de compras y ventas en los siguientes tres meses por parte de la empresa. Esto es fundamental porque se puede prever un aumento de compras en los siguientes meses y que la empresa no tenga suficiente liquidación y tenga que anticiparse a la situación, ya sea con pagarés, adelantando el cobro a clientes que habían pagado con una letra, pedir un préstamo al banco, etc. Tener datos futuros permite a la empresa anticiparse a las distintas situaciones, yendo siempre un paso por delante. Véase la Figura 4.33.



Figura 4.33: Pestaña Predicciones dashboard.

Todos estos *dashboard* tienen en común que son **totalmente interactivos** y fáciles de usar. Además cuentan con características adicionales propias de Power BI muy interesantes, pero que no son mencionadas en este apartado por no ser un tema principal.

Para terminar, se ha visto interesante crear un diseño para móviles y publicarlo para que el *dashboard* pueda ser visualizado desde el teléfono móvil. De esta manera, se consigue que el *dashboard* sea más accesible, pudiéndolo consultar en un ámbito fuera de la oficina. Véase la Figura 4.34.



Figura 4.34: Resultado final del proyecto en teléfono móvil personal.

4.6.3 Publicación del dashboard en la nube

Suponiendo que se está trabajando en un entorno colaborativo con más personas o, que simplemente se quiere que las personas puedan ver el *dashboard* sin necesidad de mandárselo explícitamente cada vez que lo necesiten, se ha publicado el *dashboard* en la nube en un entorno de trabajo en el que estarían las personas interesadas y/o involucradas en el *dashboard*. Esta característica, junto con la posibilidad de ver del *dashboard* desde un teléfono móvil, convierte a esta aplicación lo más accesible posible.

Para hacer esto, es tan sencillo como darle al botón de "Publicar" en Power BI. Pero, si se quiere que desde la nube se actualicen los datos sin que uno tenga que hacerlo desde local, hay que configurar unos *gateways* que se conecten desde el servidor hasta las fuentes de datos para poder actualizarlos desde la nube. En el caso particular de este trabajo, hay dos *gateways*: uno hacia el ordenador del autor (para las tablas Calendario y PGC, mencionadas con anterioridad) y otra hacia la base de datos SQL

para la tabla `diarioDB`. Para ello, una vez en la nube, uno debe dirigirse al modelo semántico del *dashboard* (cuando se publica un *dashboard* en la nube se crea automáticamente el modelo semántico que son los datos en sí y el reporte que es el *dashboard*), tres puntos y ajustes. Una vez hecho esto, en la sección de *Gateways and cloud connections* se configuran. Véase la Figura 4.35.

△ Gateway and cloud connections

To use a data gateway, make sure the computer is online and the data source is added in [Manage Connections and Gateways](#). If you're using an On-premises data gateway (standard mode), please select the corresponding data sources and then click apply.

Gateway connections

Use an On-premises or VNet data gateway

On

Gateway	Department	Contact information	Status	Actions
 Personal Gateway			<input checked="" type="checkbox"/> Running on PCCOQUE	

Cloud connections

Data sources included in this semantic model:

 <code>SqlServer{"server":"tfg-inso-db-server.database.windows.net","database":"tfg-inso-db"}</code>	Maps to: <input type="button" value="Personal Cloud Connect"/>
---	---

Figura 4.35: Gateways creadas en el servidor de Power BI.

5 Conclusiones

Este trabajo principalmente tenía como objetivo crear un *dashboard* en el que se pudieran visualizar, analizar y comparar datos contables. Como se ha mostrado en el apartado 4.6.2, se ha desarrollado este *dashboard* que se comenta, en Power BI, con cuatro pestañas; cada una mostrando una información relevante distinta.

Además del *dashboard* en sí, el trabajo contaba con algunos objetivos adicionales que tenían en cuenta tanto el *dashboard*, como procesos previos a la hora de realizar las visualizaciones. En primer lugar, el proceso desde que se recogen y procesan los datos hasta que llegan a Power BI es totalmente automatizado; con una arquitectura compleja teniendo en cuenta siempre que se trabaja con una gran cantidad de datos. Un ejemplo de ello es la automatización incremental en Power BI explicado en la sección anterior.

En segundo lugar, los datos se guardan de manera segura en la nube y no en local; siendo accesibles desde cualquier lugar del mundo con acceso a los datos. Por otra parte, el *dashboard* no solo está también publicado en un servidor de Power BI en la nube, siendo accesible desde internet, sino que también se ha ido más allá y se ha añadido una funcionalidad adicional a lo propuesto en un principio: la opción de poder visualizar el *dashboard* desde el teléfono móvil implicando un diseño especial para este tipo de dispositivo. De esta manera se consigue que el producto final no solo sea accesible desde cualquier parte del mundo, sino que también no haga falta ni un ordenador para poder verlo: si se necesitase verlo, por alguna urgencia, en un ámbito sin ordenador, se puede ver desde el móvil.

Por último, el *dashboard* no solo muestra datos ya existentes, sino que también se pueden visualizar estimaciones futuras en su pestaña correspondiente. Estas predicciones, al haber involucrado MLOps, se asegura que siempre sean las más actualizadas posibles.

Como líneas futuras a este trabajo, estaría bien tener un servidor con capacidad masiva de entrenamiento de modelos de aprendizaje automático. En este trabajo se ha utilizado el ordenador propio por no tener otra opción; lo cual ha resultado una limitación a la hora de entrenar y comparar modelos. Así como para realizar estimaciones futuras entre los distintos algoritmos.

Respecto a esto último, una comparativa de modelos más rigurosa, con más modelos y más métricas hubiera sido ideal, pero este tema iba más allá de este trabajo pues llevaría meses poder hacerlo de bien de manera correcta. Por ejemplo, sería de gran utilidad incorporar gráficas con el valor de las

métricas a lo largo del entrenamiento; de esta manera se podría ver mejor si llegado a cierto punto, el modelo de aprender.

Por otro lado, debido a la facilidad que ofrece Power BI a la hora de importar nuevas tablas y relacionarlas de una manera sencilla, estaría interesante poder utilizar más fuentes de datos y tratar de mostrarlos con los que ya se tienen para conseguir conocimiento incluso mayor de la empresa.

También, en caso de que la cantidad de datos o complejidad del proyecto aumentase, sería interesante estudiar herramientas alternativas a las que se ha usado un poco más potentes como puede ser Airflow en lugar de Prefect. Para terminar, aunque no se haya profundizado mucho en Dagshub, esta herramienta es, por lo visto, una herramienta muy potente para combinarla con Mlflow. Para los objetivos que se habían planteado en un principio, la utilidad que se le ha dado a Dagshub es más que suficiente, pero se espera estudiar algo más en profundidad esta herramienta y ver cómo puede llevar este trabajo a un escalón incluso superior.

6 Apéndice

6.1 Procedimiento especulativo de cuenta divisionaria

Los activos no corrientes, cuando se compran, la contabilidad no los figura en la cuenta de compras, i.e. 6XX. Sino que lo hace en la cuenta de inmovilizados 2XX. Esto es porque un activo no corriente, como puede ser la compra de una furgoneta, no se ve como un gasto, sino como un aumento de activo (no corriente) de la empresa. En otras palabras, la empresa, al comprar la furgoneta, no ha perdido dinero, simplemente lo ha transformado⁴⁴.

Por otro lado, la compra de activos corrientes, por lo general, sí que se suelen contabilizar como gasto —como puede ser la compra de folios—. Pero ¿y la compra de existencias? El Plan General Contable tiene un grupo para estas: el 3XX. La compra de existencias se puede ver dos maneras: como compra/gasto, que, al venderla irá a parar a la cuenta de venta/ingreso (cuentas involucradas en esta operación son las 6XX y la 7XX). O, por otra parte, se puede ver como un aumento de activo (corriente) de la empresa, y cuando estas se vendan, disminuye dicho activo (cuenta involucrada en esta operación es la 3XX).

Bien, pues el procedimiento especulativo de cuenta divisionaria (el más utilizado por empresas, el empleado por el Plan General Contable, el empleado en este trabajo) indica que las compras de existencias se vean así: **como compra**. Y, por tanto, queden registradas en la cuenta 6XX.

Sin embargo, hay otros dos métodos —con alguna pequeña diferencia entre ellos— que interpretan la compra de existencias como que se debería registrar en la cuenta 3XX del PGC. Estos dos métodos son:

- El procedimiento administrativo.
- El procedimiento especulativo de cuenta única —pues la compra/venta de existencias afecta solo a la cuenta 3XX—.

⁴⁴El gasto de dicho vehículo, la empresa lo figura como amortización según vayan pasando los años.

6.2 Plan General Contable español: cuentas más importantes

1. Financiación Básica	1XX	4. Acreedores & deudores	4XX
Capital social (grupo personas)	100	Proveedores	400
Fondo social	101	Acreedores	410
Capital (individual)	102	Clientes	430
Resultado del ejercicio	129	Deudores	440
Deudas a l/p con entidades de crédito	170	H.P. IVA soportado	472
Proveedores de inmovilizado a l/p	173	H.P. IVA repercutido	477
Efectos a pagar a largo plazo	175	Seguridad Social trabajador	476
2. Inmovilizado	2XX	5. Cuentas financieras	5XX
Propiedad industrial	203	Proveedores de inmovilizado a l/p	523
Terrenos y bienes naturales	210	Caja	572
Maquinaria	213	Banco	572
Mobiliario	216	6. Compras y gastos	6XX
Elementos de transporte	218	Compra de mercaderías	600
Amortiz. acum. inm. material	281	Primas y seguros	625
3. Existencias	3XX	Sueldos y salarios	640
Comerciales	300	Amort. de inm. material	681
Materias primas	310	7. Ventas e ingresos	7XX
Material de oficina	328	Ventas de mercadería	700

6.3 Predicción intrapolar vs. extrapolar

Es común ver datos representados en gráficas bidimensionales, i.e. con eje x y eje y . Cada eje representa una variable de los datos. Como los datos son finitos, los valores de las gráficas x e y también lo son. A la hora de realizar la predicción, puede darse el caso que dicha predicción sea dentro de los límites de los valores del eje x ; en cuyo caso se dice que la predicción es **intrapolar**. Un ejemplo de ello es la predicción de ventas de helados basándose en la temperatura del día siguiente. Véase la Figura 6.1.

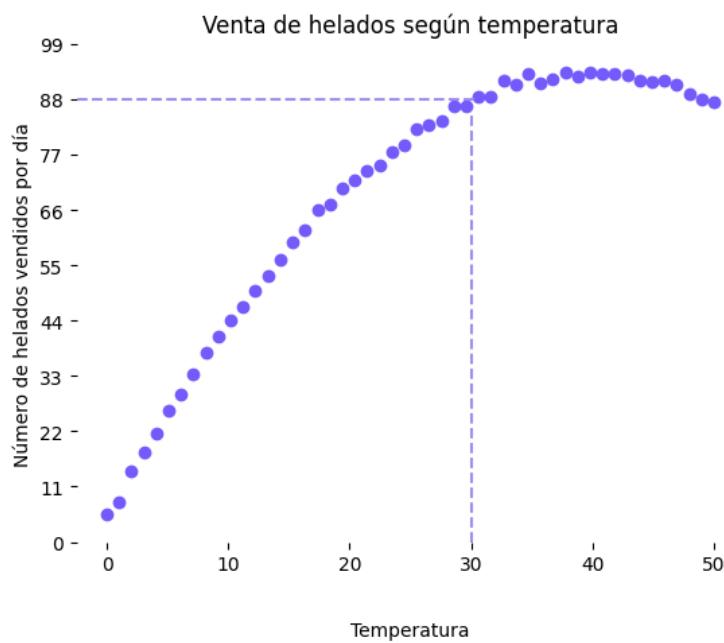


Figura 6.1: Ejemplo predicción intrapolar.

Sin embargo, en una gráfica de una serie temporal, el extremo derecho representa, por lo general, el tiempo presente o la fecha del último dato obtenido. Lo que se trata de predecir (tiempo futuro) va más allá del eje x de la gráfica, y por ende, es una predicción **extrapolar**.

6.4 Gradient Boosting

6.4.1 Descenso del gradiente

Un algoritmo de aprendizaje automático se puede ver como una función matemática de n variables. Estas n variables son los parámetros que dicho algoritmo emplea para realizar una determinada tarea

de predicción; esta puede ser regresión, clasificación binaria, clasificación múltiple, etc.

Son el valor de estos parámetros —también llamados pesos— los que el algoritmo debe aprender para generar una salida esperada a un *input* en concreto. En una primera instancia, el algoritmo no sabe el valor óptimo de estos parámetros y por ende, debe ir actualizándolos hasta encontrarlos. Para ello, muchos algoritmos de aprendizaje automático (RRNN, regresión lineal y logística, algoritmos de *Gradient Boosting*, etc.) emplean el **descenso del gradiente** para encontrar estos valores de parámetros que hacen al algoritmo generar un modelo que sea lo mejor posible para la tarea específica que este deba realizar.

Para explicar en qué consiste el descenso del gradiente, se ve apropiado poner un ejemplo muy simple de regresión lineal con **un solo parámetro**; pero, cabe remarcar que la misma explicación puede ser extendida a varios parámetros. Lo único, que en vez de tener una gráfica en dos dimensiones (véase la Figura 6.2) se obtendría una gráfica en $n + 1$ dimensiones, siendo n el número de parámetros que el algoritmo que se vaya a emplear utilice.

Supóngase que se tiene un problema de regresión lineal. Es decir, a partir de la Figura 6.2 se quiere obtener una recta que mejor se ajuste a los datos.

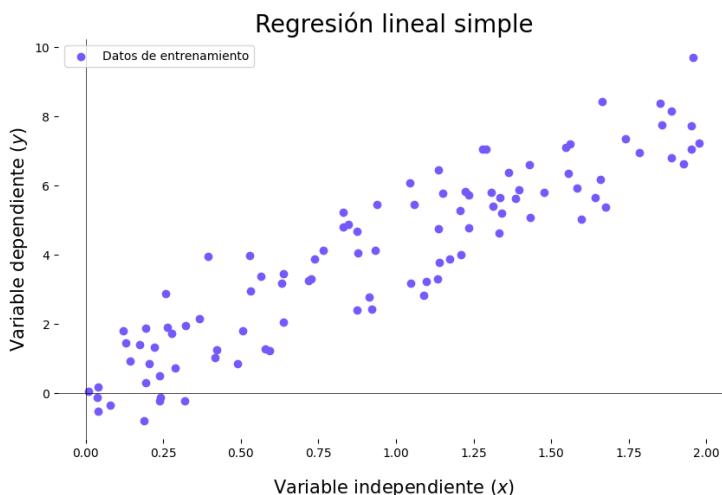


Figura 6.2: Problema regresión lineal.

Este problema de regresión lineal tendría dos parámetros: uno que indica el corte de la recta con el eje y (i.e. altura de la recta) y otro parámetro para indicar la inclinación de dicha recta. Esto en una fórmula matemática se ve de la siguiente manera:

$$y = \theta_0 + \theta_1 \cdot x \quad (6.4.1)$$

En la ec. 6.4.1, θ_0 indica el corte de la recta con el eje y , mientras que θ_1 indica la inclinación de la recta. Para simplificar más las cosas, se ha elegido un problema en el que $\theta_0 = 0$. Es decir, la recta de regresión es lineal y no afín (una recta lineal pasa por el origen de coordenadas $(0,0)$ y una recta afín no). De esta manera, la ec. 6.4.1 se simplifica de tal manera que solo hay un parámetro a determinar: θ_1 :

$$y = \theta_1 \cdot x \quad (6.4.2)$$

Como solo hay un parámetro, para simplificar las cosas, se va a renombrar a θ_1 : $\theta_1 \Rightarrow \theta$. Para cada valor de θ se obtiene una recta distinta. Véase la Figura 6.3.

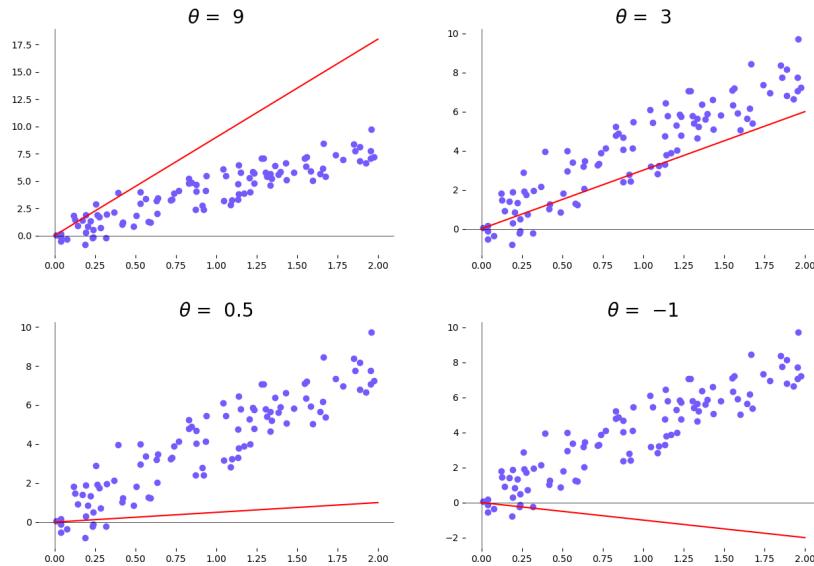


Figura 6.3: Recta de regresión para distintos valores del parámetro θ .

En la Figura 6.3 se puede ver de manera muy clara cómo distintos valores de θ ajustan de una manera mejor o peor la recta a los datos. Si este ajuste es bueno o malo, es determinado por una **función de coste** —también llamada función de pérdida— denotada generalmente con la letra J . Hay varias funciones de coste al alcance del programador, y varias de ellas oportunas para un mismo problema. La función de coste por excelencia para la regresión lineal es RMSE (*Root Mean Squared Error*). Esta función de coste —al igual que el resto— devuelve un número $C \in \mathbb{R}$ que indica cómo de bien se ajusta el modelo a los datos. En otras palabras, la función de coste es un indicativo de cómo de buenos son los actuales valores de los parámetros (θ en este caso). Por ejemplo, en la Figura 6.3, el valor de

$\theta = 3$ tendrá un valor C menor que con el que se obtiene con $\theta = -1$.

De esta manera, cada valor del parámetro θ tiene asociado un coste. Esto se puede ver en una gráfica de dos dimensiones, en la que el eje x sean los distintos valores de θ y el eje y su coste asociado. Véase la Figura 6.4.

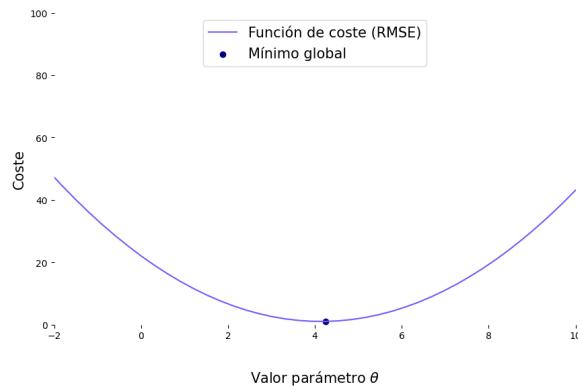


Figura 6.4: Función de coste para distintos valores de θ .

El objetivo del problema pues, pasa por encontrar el valor de θ que minimice la función de coste. Este valor es el **mínimo global** (en este caso) dibujado en la Figura 6.4.

Para llegar a dicho punto, se aplica lo que se conoce en matemáticas como **gradiente**. En una primera instancia, el algoritmo no sabe el valor óptimo de θ , y se inicia de manera aleatoria. Supóngase que este valor aleatorio es $\theta = 9$. Como se ve en la Figura 6.3, este valor no aproxima muy bien la recta a los puntos. El coste asociado a este valor, está lejos del mínimo. Véase la Figura 6.5.

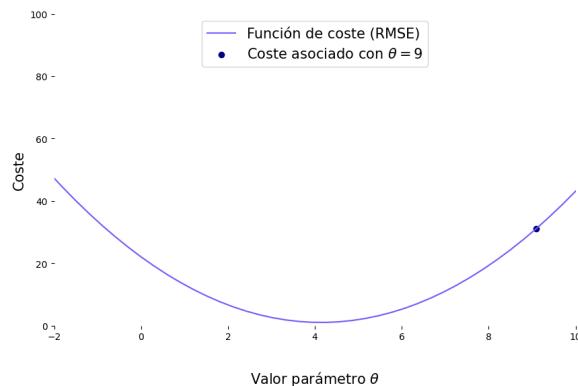


Figura 6.5: Coste asociado a $\theta = 9$.

El gradiente de una función ∇J (derivada) indica la dirección de máximo crecimiento de esta. Como

justo lo que se pretende es ir en la dirección contraria —dirección de máximo decrecimiento— para encontrar el mínimo global, se calcula la dirección contraria al gradiente. Para ello, basta con obtener el valor negativo del gradiente, i.e. $-\nabla J$.

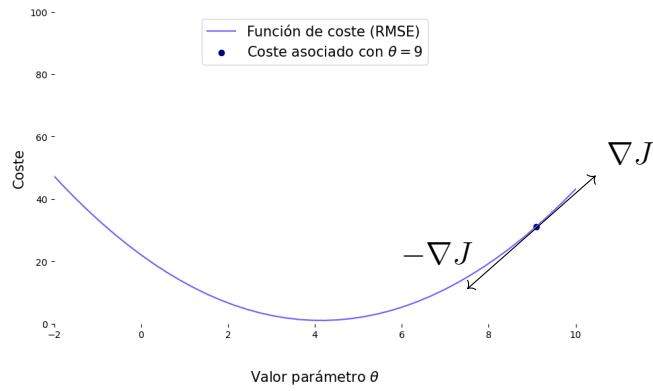


Figura 6.6: Gradiente.

Una vez se sabe a la dirección a la que hay que moverse, hay que decidir cuánto moverse. Esto lo determina el *learning rate* (α). Este *learning rate* no puede ser muy alto, porque provocaría saltos muy grandes con poco ajuste al mínimo global. Véase la Figura 6.7.

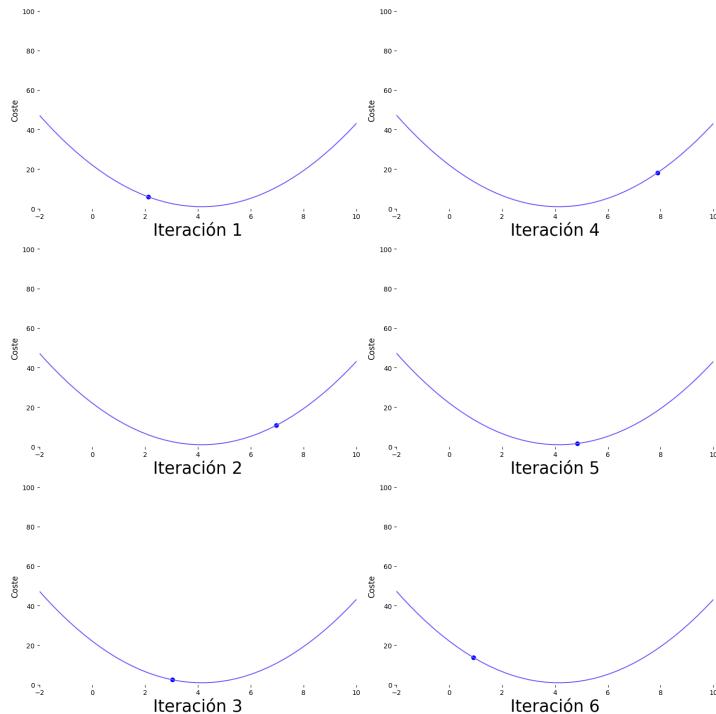


Figura 6.7: Ejemplo de mala práctica: Learning rate α demasiado alto.

Por el contrario, si el *learning rate* es demasiado bajo, provoca que el algoritmo aprenda muy despacio. Esto, aunque pueda parecer buena idea, no lo es porque los recursos son muy importantes en la informática. Esto implica que si el algoritmo aprende muy despacio, le llevaría mucho tiempo de entrenamiento poder llegar a un mínimo local. Este coste computacional es, por otra parte, lo que se trata de evitar también. De hecho, uno de los motivos de descarte de algoritmos en las comparativas de estos, es su tiempo de entrenamiento.

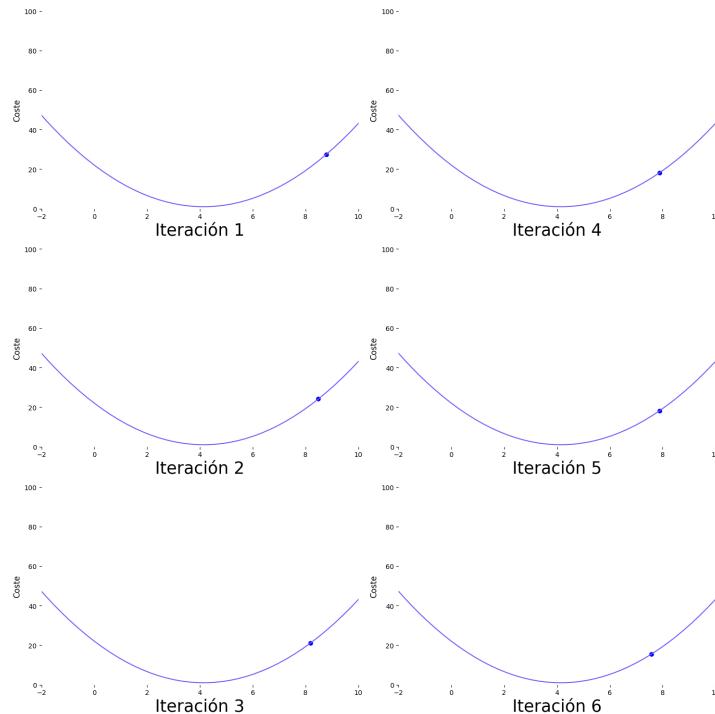


Figura 6.8: Ejemplo de mala práctica: Learning rate α demasiado bajo.

Como conclusión, establecer un correcto valor de *learning rate* en un algoritmo es fundamental tanto para la eficacia como para la eficiencia de este. Además de que, ese hiperparámetro es común en la mayoría de algoritmos de aprendizaje automático, no solo de redes neuronales.

Los valores más comunes y que son un estándar a día de hoy son valores como $\alpha = 0.001$, $\alpha = 0.01$, $\alpha = 0.05$, $\alpha = 0.1$.

Para poder ver un ejemplo visual de un *learning rate* correcto establecido, véase la Figura 6.9.

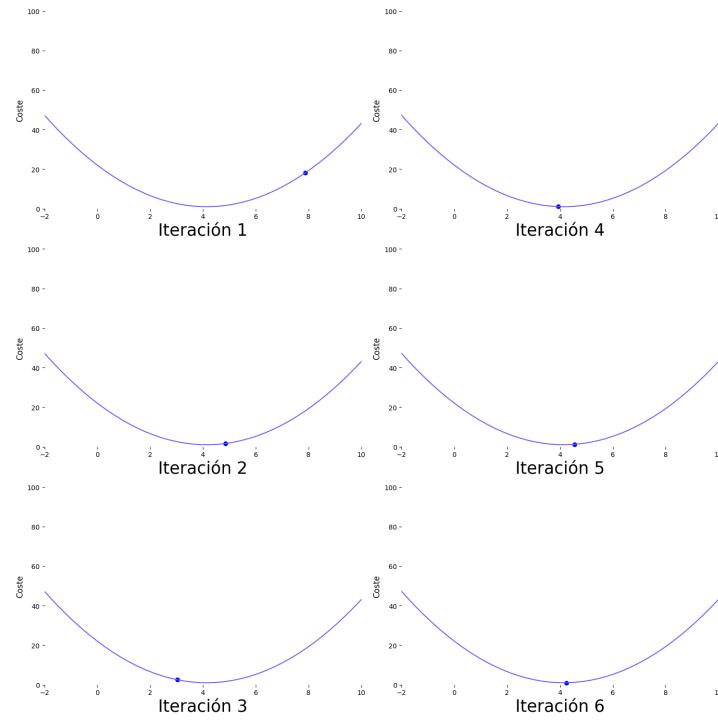


Figura 6.9: Ejemplo de un learning rate adecuado.

Se puede comprobar que si en la ec. 6.4.2, se establece el valor de θ_1 al valor del mínimo global de la función de coste ($x = 4.24$) se obtiene una recta de regresión que se ajusta muy bien a los datos. Véase la Figura 6.10.

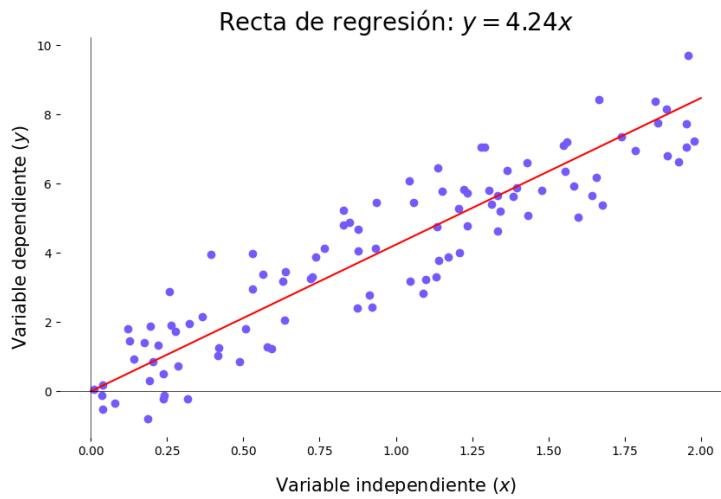


Figura 6.10: Solución problema recta de regresión.

En esto básicamente consiste el gradiente descendente. El algoritmo continúa hasta alcanzar un número

máximo de iteraciones (definido por el programador) o hasta alcanzar una diferencia muy pequeña en la función de coste entre una iteración y la anterior. Esto equivale a establecer un límite en el gradiente. Esto es porque cuanto más pequeño sea el gradiente, más cerca se está de un mínimo de la función (la pendiente será más plana), y por ende, la función de coste se actualizará de una manera muy leve.

```
def gradienteDescendente(funcionCoste_J, tasaAprendizaje, maxIter,
                           minValorGradiente):
    # Inicializar parametro (theta) con valores aleatorios o iniciales
    theta = rand_param_init()
    iteracion = 0
    # Inicializar gradiente infinito
    gradiente = float('inf')

    # Mientras no se alcance el numero maximo de iteraciones y gradiente
    # sea mayor que su limite de valor minimo
    while iteracion < maxIter and gradiente > minValorGradiente:
        # Calcular el gradiente de funcion de coste (theta) en el punto
        # actual
        gradiente = calcularGradiente(funcionCoste_J, theta)

        # Actualizar los parametros utilizando la regla de actualizacion
        # del descenso de gradiente
        theta -= tasaAprendizaje * gradiente

        # Incrementar el contador de iteraciones
        iteracion += 1

    if gradiente <= minValorGradiente:
        print("La convergencia ha sido alcanzada")
    elif iteracion >= maxIter:
        print("El algoritmo ha alcanzado el numero maximo de iteraciones
              sin converger")
    else:
        print("Algo ha ido mal")
```

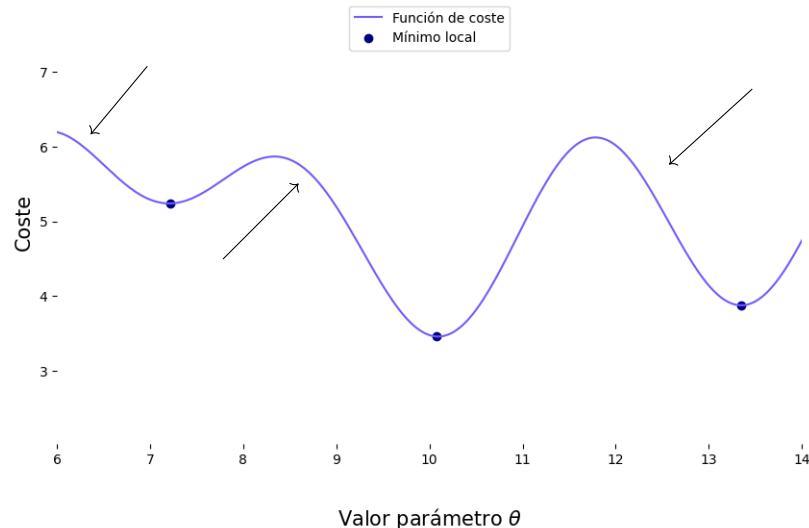
```

minCosteObtenido = funcionCoste_J(theta)
print(f"El coste de la función es {minCosteObtenido} con el valor del
      parametro theta = {theta}.")
return theta, minCosteObtenido

```

Listing 1: Pseudo-código algoritmo de descenso del gradiente.

Por último, caben a destacar dos cosas más. En el ejemplo dado, la función de coste de la Figura 6.4 se trataba de una función convexa. Una propiedad de estas funciones es que **siempre** tienen un mínimo global. Podría darse el caso en el que la función de coste que se obtenga sea no-convexa. Véase la Figura 6.11.

Figura 6.11: Ejemplo de función no-convexa con las flechas indicando supuestos valores iniciales aleatorios de θ .

En la Figura 6.11, se ve el problema de que la función sea no-convexa. Y es que dependiendo del valor inicial aleatorio del parámetro θ , el algoritmo del gradiente descendente “llevará” dicho punto a un mínimo; pero este mínimo ya no tiene por qué ser global. En otras palabras, puede darse del caso en el que el valor de función de coste obtenido no sea el más óptimo de todos. En la Figura 6.11, si los parámetros se inicializan aleatoriamente en las posiciones de la primera y tercera flecha ($x = 6.5, x = 12.2$ aproximadamente), dicho punto llegará hasta un mínimo local que no es el mínimo global de la función de coste. Esto implica que no se vaya a obtener el valor óptimo del parámetro θ .

en esos casos nunca. Además, el valor mínimo C que se obtenga, depende exclusivamente del valor aleatorio inicial de θ .

El algoritmo del descenso del gradiente **no asegura llegar al mínimo global de la función, sino a un mínimo local**. Para tratar de evitar mínimos globales “pobres” (con un valor muy alejado del mínimo global) hay distintas técnicas que se aplican en conjunto a este algoritmo. Algunas de estas técnicas pueden ser: calcular varios valores aleatorios iniciales para los parámetros, descenso de gradiente estocástico, etc. La explicación de estos conceptos va más allá del contenido de este trabajo.

La segunda y última cosa a remarcar, es que esta misma explicación se puede extender para dos parámetros (función en tres dimensiones), tres parámetros (función en cuatro dimensiones), etc. En estos casos, es importante remarcar que el **gradiente es un vector de tantas dimensiones como parámetros tenga la función**. Cada posición del vector gradiente hace referencia a cada uno de estos parámetros. Para el cálculo de dicho gradiente, se calcula la derivada parcial de cada parámetro respecto a la función de coste. Finalmente, al igual que se ha hecho en el ejemplo recientemente explicado, se calcula $-\nabla J$.

Se deja una función de coste en tres dimensiones, donde se puede ver de una manera sencilla que lo explicado hasta ahora es extensible a dimensiones mayores. Véase la Figura 6.12.

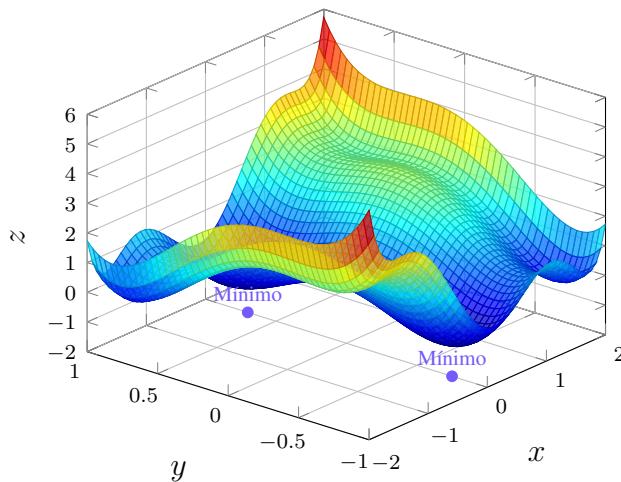


Figura 6.12: Six-Hump Camel function.

6.4.2 Regularización L1 y L2

La regularización son técnicas utilizadas en aprendizaje automático para evitar que un modelo sobre ajuste (*overfitting*) los datos de entrenamiento. Esto ocurre cuando el modelo predice, por lo general,

de manera correcta los datos de entrenamiento, pero es incapaz de hacer buenas estimaciones hacia datos con los que no ha entrenado. Si ocurre esto, implica que el modelo tiene una varianza alta (resultados muy distintos ante *inputs* distintos). Por tanto, lo ideal, es encontrar un punto, en el cual, el modelo generado sea capaz de predecir datos de una manera correcta (*low bias*) al mismo tiempo que sea robusto ante distintos datos, i.e. que generalice bien (*low variance*). A esto se le conoce como *Bias-Variance Trade off*. Véase la Figura 6.13.

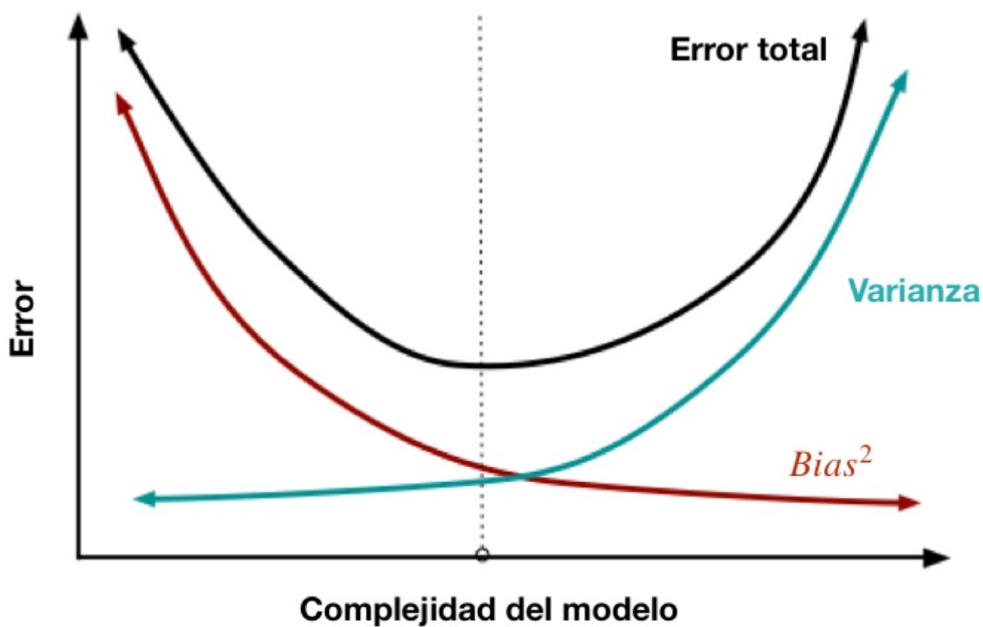


Figura 6.13: Error-Varianza. Fuente: <https://aprendeia.com/bias-y-varianza-en-machine-learning/>

Las técnicas de regularización buscan generar un modelo simple que sea capaz de generalizar bien. A esto se le conoce como principio de parsimonia o la navaja de Okham (Duignan, 2024).

Aunque haya varias técnicas de regularización, L1 y L2 en concreto, penalizan altos valores de pesos (parámetros) en el modelo generado. Esto es porque valores altos de estos pesos, no solo conducen a un modelo más complejo, sino que también a un modelo con mayor varianza⁴⁵. Estas dos condiciones conducen a un sobre ajuste en el modelo generado.

Por un lado, L1 (también conocido como LASSO: *Least Absolute Shrinkage and Selection Operator*) penaliza valores altos en los pesos del modelo, añadiendo la suma en valor absoluto de estos a la

⁴⁵Basta con imaginar dos pesos w_1 y w_2 con valores de 0.1 (bajo) y 100 (alto) respectivamente. Si se multiplican dos números A y B —cuales sean— a estos dos pesos, la diferencia entre $|w_1A - w_1B|$ siempre será menor que $|w_2A - w_2B|$, y por ende, w_1 (un peso bajo) conduce a una menor varianza.

función de coste:

$$J(\theta) = J'(\theta) + \lambda \underbrace{\sum_{j=1}^p |w_j|}_{\text{Regularización L1}} \quad (6.4.3)$$

En donde:

- $J'(\theta)$ es la función de pérdida definida para el problema, la cual se busca **minimizar**.
- λ es un factor $\in (0, 1)$ que indica la fuerza de la penalización: cuanto más cercano a 1 más fuerte es.
- p es el número total de pesos del modelo.
- w_j es el valor de un peso en concreto.
- $J(\theta)$ es la función de pérdida resultante.

Como los pesos se pasan a valor absoluto, siempre serán positivos y, por tanto, siempre añadirán valor a la función de coste $J'(\theta)$. Como esta función de coste, el objetivo es minimizarla, se están penalizando valores de pesos altos.

Por otro lado, L2 (también conocida como *Ridge*) también penaliza valores altos de los pesos del modelo, pero en vez de aplicar el valor absoluto, aplica el cuadrado de estos. De esta manera, al igual que con el valor absoluto, siempre son positivos, y por ende, el valor de los pesos siempre suma a la función de coste.

$$J(\theta) = J'(\theta) + \lambda \underbrace{\sum_{j=1}^p w_j^2}_{\text{Regularización L2}} \quad (6.4.4)$$

En la ec. 6.4.4, las variables tienen el mismo significado que en la ec. 6.4.3.

Se llaman L1 y L2 porque aplican las normas L1 (o norma Manhattan) y L2 (o norma euclídea) de matemáticas⁴⁶.

Mientras L1 tiende a asociar un valor de 0 a aquellos pesos irrelevantes en el modelo y mantener solo aquellos que realmente importan, L2, por el otro lado, tiende a mantener todos los pesos más o menos equilibrados con un valor próximo a cero. En algoritmos como regresión lineal, en donde cada peso

⁴⁶La regularización L2 es en realidad la norma L2 sin la raíz cuadrada.

está asociado a una variable del modelo, esto tiene una interpretación muy clara:

- Si se sospecha que varias variables del modelo son irrelevantes y/o no están “correladas” entre sí, se recomienda emplear L1 —pues asignará un valor de 0 a aquellos pesos asociados con variables irrelevantes—.
- Por el contrario, si se sospecha que varias variables influyen en el modelo y/o están “correladas” entre sí, se recomienda usar L2.

Sin embargo, en otros algoritmos, esta interpretación no es tan clara y es posible usar ambos a la vez: como hace **XGBoost**.

La razón por la que L1 conduce a determinados pesos a ser 0 y L2 a tener valores bajos, se puede ver muy bien si se grafican las funciones $|x|$, i.e. L1 y x^2 , i.e. L2.

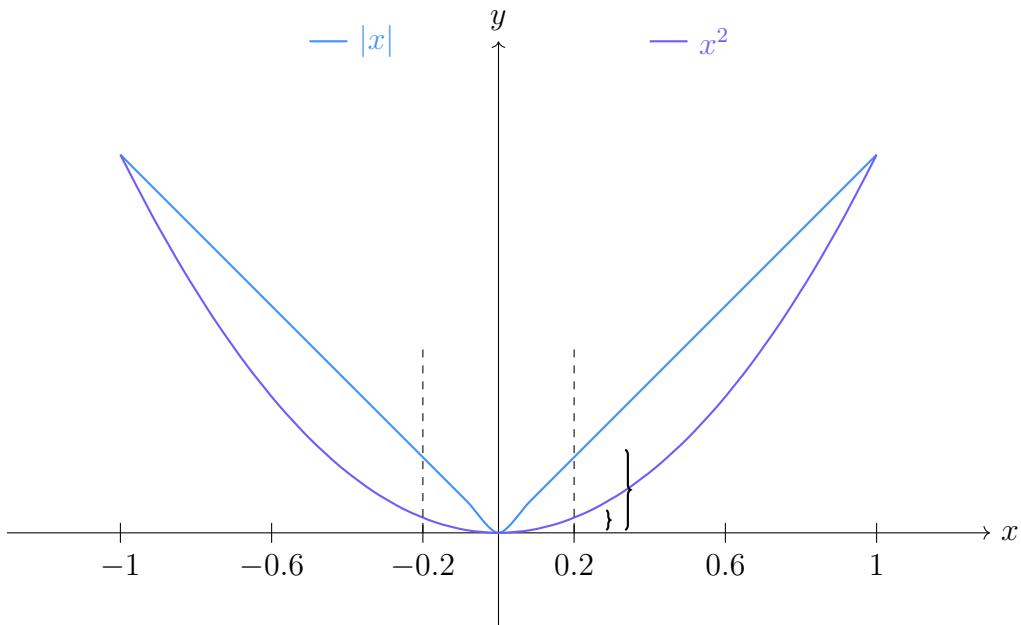


Figura 6.14: Gráficas $|x|$ y x^2 .

Dentro del contexto en el que los pesos están normalizados y, por tanto, sus valores $\in [-1, 1]$, en la Figura 6.14 se puede ver cómo incrementos en el eje x no tienen un impacto muy significativo en la imagen de x^2 ; pero, en cambio, sí que lo tienen en la imagen de la función $|x|$. Esto se traduce en que, mientras en L1 si hay una gran diferencia entre que un peso se aproxime mucho a cero —a si no lo hace—, en x^2 no importa tanto. En 6.14 se ha puesto el ejemplo de $x = \pm 0.2$, pero esto mismo se puede aplicar a cualquier $x \in (-1, 1)$.

6.5 Desvanecimiento del gradiente y gradiente explosivo

El mecanismo por el que las redes neuronales aprenden, es a través de la optimización de sus pesos. Estos pesos se actualizan después de un determinado número de entrenamientos (se especifica con el *batch size*). La optimización de estos pesos se hace mediante el algoritmo de **Backpropagation** (Rumelhart et al., 1986).

Este algoritmo revolucionó el campo de la inteligencia artificial. Se basa en la regla de la cadena para la optimización de pesos. De manera muy simplificada, esta regla de la cadena son multiplicaciones. En muchas ocasiones, la actualización de un peso implica muchas multiplicaciones⁴⁷.

Ahora, el problema del **desvanecimiento del gradiente** surge cuando muchos de estos valores que se están multiplicando $\in (-1, 1)$. Cuando se multiplican muchas veces (miles de veces) números $\in (-1, 1)$, el resultado que se obtiene es un número extremadamente pequeño. Luego este peso que se está tratando de actualizar, no lo hace de una manera eficiente —pues se actualiza en una cantidad ínfima—.

Por el contrario, el problema del **gradiente explosivo** se da cuando la mayoría de estas multiplicaciones $\notin (-1, 1)$. El resultado de multiplicar muchas veces números $\notin (-1, 1)$ da lugar a un número muy grande. Luego el peso que se está tratando de actualizar lo hace de manera muy brusca y no llega a converger a un valor óptimo.

Las RNN, por su naturaleza, pueden caer en estos problemas.

6.6 Transformers: origen y explicación

Como se ha mencionado, los Transformer tienen su origen en el campo del procesamiento del lenguaje natural. Este campo ha sido de especial interés a lo largo del siglo XXI, ya sea para generar texto, generar *chatbots*, traducción, resúmenes, etc.

Ahora, las redes neuronales no entienden de palabras, entienden de números. Para pasárlas a números, lo que se hizo fue *one-hot encoding*. Es decir, un vector de tantas posiciones como palabras tuviera el diccionario. Cada posición hace referencia a una palabra del diccionario. Entonces, cada palabra es un vector de todos ceros, menos un uno en su posición correspondiente.

⁴⁷miles de multiplicaciones.

Para comprimir toda esta información, este vector se convierte en otro de menor dimensionalidad. A esto se le conoce como *embedding* (Mikolov, Chen, et al., 2013) (Mikolov, Sutskever, et al., 2013). De esta manera, se obtenía información característica de una palabra en un vector mucho menor. Dicho vector es muy característico, pues palabras con relación entre sí (e.g. enero y febrero) tienen asignados vectores muy parecidos. Para poder visualizar esto, se puede comprimir esta información de cada palabra en un vector de tres dimensiones, y el resultado es el siguiente:

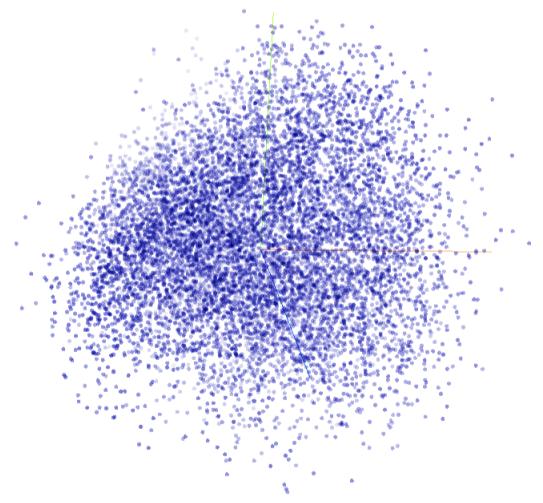


Figura 6.15: Palabras representadas en vectores de tres dimensiones. Cada punto es el final del vector, i.e. cada palabra. Fuente: <https://projector.tensorflow.org/>

Con las palabras expresadas en números, había que encontrar una red neuronal para entrenarla y las RNN⁴⁸, al ser redes neuronales con *memoria*, fueron la solución (Sutskever et al., 2014), (Cho, Van Merriënboer, Gulcehre, et al., 2014). Sin embargo, las RNN sufrían el problema de memoria a largo plazo, muy importante en las secuencias de palabras, como se puede ver en el siguiente ejemplo:



Ayer, un chico estaba realizando deporte de una manera bastante intensa hasta que su padre llegó.

Figura 6.16: Ejemplo, retención de memoria en texto. La palabra *su* hace referencia a una palabra muy anterior a ella (*chico*).

⁴⁸Cabe destacar que RNN también engloba a las redes neuronales LSTM y GRU.

Para solventar el problema de las RNN se añadió a estas el concepto de atención —i.e. RNN + atención—(Bahdanau et al., 2014). Este tipo de redes supusieron el estado de arte en el campo de NLP durante mucho tiempo (Luong et al., 2015) (Y. Wu et al., 2016) (Jozefowicz et al., 2016).

Pese a que se obtenían buenos resultados con estas redes neuronales, sufrían de que tardaban mucho tiempo en entrenarse. Esto es porque este entrenamiento se realizaba de manera secuencial. Es decir, se iba procesando palabra por palabra, lo que resultaba en un entrenamiento muy costoso.

En 2017, surge una nueva red neuronal: Los Transformer (Vaswani et al., 2017). Los creadores de esta red neuronal se dieron cuenta de que en el algoritmo mencionado anteriormente (RNN + atención), la RNN no hacía falta, y que con la atención era suficiente. Esta es la razón del nombre de la publicación: *Attention is all you need*. Esta red neuronal cambió el campo de NLP —y posteriormente muchos otros cuando utilizó los Transformer en ellos: e.g. imágenes—. No solo obtenían mejores resultados que sus antecesores, sino que también su entrenamiento no era secuencial; i.e. era “paralelizable”. Es decir, que esta red neuronal entrena con todos los datos disponibles al mismo tiempo, convirtiéndolo en un algoritmo más eficiente también. Su arquitectura es la siguiente:

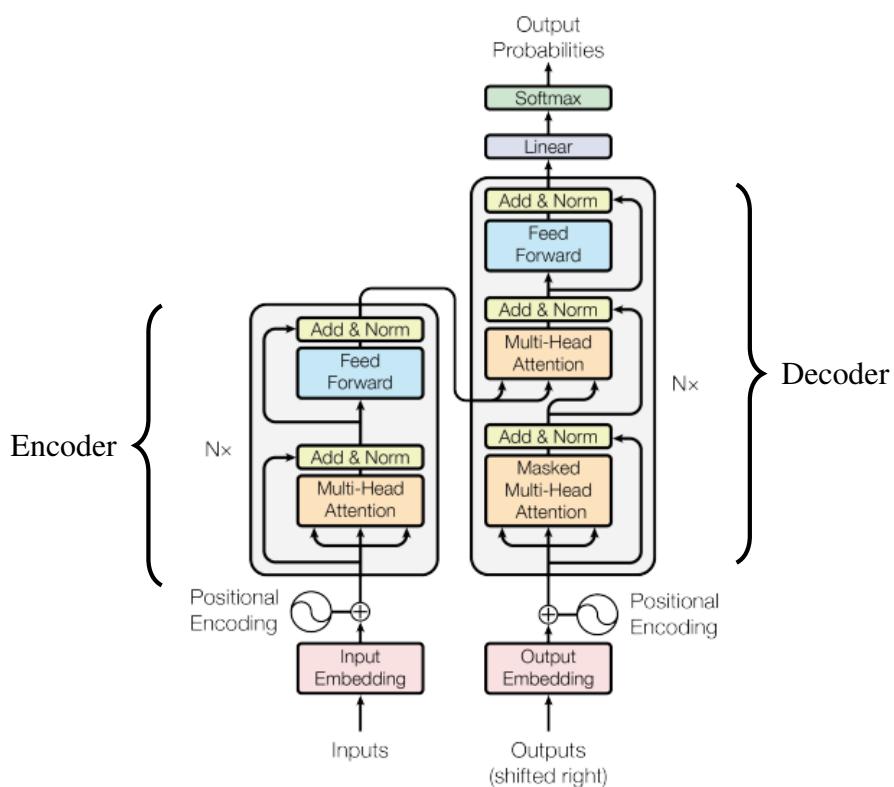


Figura 6.17: Arquitectura del Transformer original. Fuente: (Vaswani et al., 2017)

Sus autores denominan a esta arquitectura como “simple”. Por muy *trivial* que parezca, se va a proceder sus partes. Se va a empezar explicando la parte izquierda primero; luego la derecha es muy parecida.

6.6.1 Embedding y codificador posicional

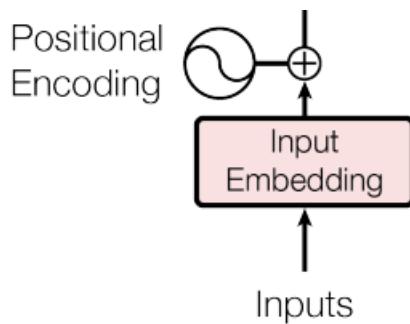


Figura 6.18: Parte izquierda: embedding y encoding posicional. Fuente: (Vaswani et al., 2017)

Los *inputs* (Figura 6.18), estando dentro el contexto de NLP, son secuencias (oraciones). Estas secuencias están formadas por palabras (denominadas como *tokens*⁴⁹). Las palabras, como ya se ha mencionado con anterioridad, es necesario transformarlas en valores numéricos para poder ser analizadas y procesadas por una red neuronal. A esta transformación se la conoce como *embedding* y su resultado es un vector numérico⁵⁰ con la información de la palabra. La dimensión de este vector será consistente a lo largo de la red neuronal, como se verá más adelante.

El **codificador posicional** (Figura 6.18), probablemente sea de lo más importante en la arquitectura. Como ya se ha dicho, esta red neuronal no es secuencial, y entrena con todas las palabras al mismo tiempo. Por esto, es necesario añadir algo de información al vector *embedding* para indicar la posición de la palabra en la oración —pues esta es muy importante⁵²—. Para ello, se genera un nuevo vector, de igual dimensión al *embedding*, que se sume al vector *embedding*. Ideas triviales, como la de que dicho vector sea todo ceros, menos un uno en la posición que ocupe la palabra en la oración, no sirven. Pues si la oración tiene muchas palabras, la suma daría lugar a un vector de valores muy altos, añadiendo complejidad al entrenamiento. Otra idea que tampoco es válida es la de asignar un

⁵⁰En realidad, un *token* puede ser una letra o alguna otra división de texto. Esto depende de la segmentación que se quiera hacer. Por lo general, será una palabra.

⁵¹En el *paper*, este vector tiene 512 dimensiones. $v \in \mathbb{R}^{d_{model}}$, $d_{model} = 512$.

porcentaje ($pos/pos_totales$) a la posición de la palabra; pues si dos oraciones, por ejemplo, tienen 2 y 4 palabras respectivamente, la primera palabra de la primera oración y la segunda palabra de la segunda oración tendrían el mismo porcentaje (0.5).

La solución pues, pasa por hacer que este vector sea sinusoidal⁵³, con una **frecuencia distinta para cada posición**. La idea intuitiva de representar un orden con frecuencia, se puede ver con la codificación binaria (Kazemnejad, 2019) (véase la Tabla 6 y la Tabla 7).

0:	0	0	0
1:	0	0	1
2:	0	1	0
3:	0	1	1
4:	1	0	0
5:	1	0	1
6:	1	1	0
7:	1	1	1

Tabla 6: Patrón en codificación binaria.

En la Tabla 6, se puede ver un patrón alternante entre 0's y 1's en cada columna. Si esto, se pasa del mundo discreto al mundo continuo, queda de la siguiente manera (véase tabla 7).

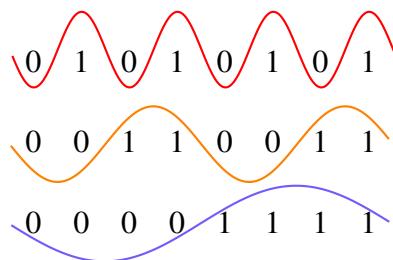


Tabla 7: Tabla 6 traspuesta: Frecuencia para número binarios.

⁵²No es lo mismo la oración: *Me río en el baño*; que la de: *Me baño en el río*.

En el artículo, esta es una fórmula matemática más compleja, utilizando senos para posiciones pares y cosenos para las impares, pero la idea es la misma: tener una frecuencia única para cada posición. Además, esta codificación tiene una ventaja, y es que los autores hipotetizan que, de esta manera, el modelo es más robusto a trabajar con longitudes de secuencias que no ha visto en el entrenamiento.

En resumen, esta parte de la arquitectura recibe la **secuencia de entrada**. Esta secuencia está compuesta de palabras y cada palabra se transforma en un vector de 512 dimensiones. Por tanto, lo que se pasa a la siguiente capa es una matriz E de dimensiones $[S, 512]$ siendo S el número de palabras de la secuencia y 512 la longitud del vector de cada palabra. Es importante remarcar que esta dimensión de salida es consistente a lo largo de todo el proceso.

6.6.2 Encoder

El *Encoder* se compone de seis capas. Estas capas son repeticiones de la anterior, i.e. son todas iguales. El objetivo de estas capas es que, dada una palabra de la oración, a qué otras palabras hay que prestar **atención** de la oración. A esto se le conoce como *self-attention*.

Esta información será vital para luego pasársela al *Decoder* —que es quien se encargará de generar la salida (texto) del modelo—.

6.6.2.1 Atención Encoder

Junto con la codificación posicional, esto es lo más importante del algoritmo.

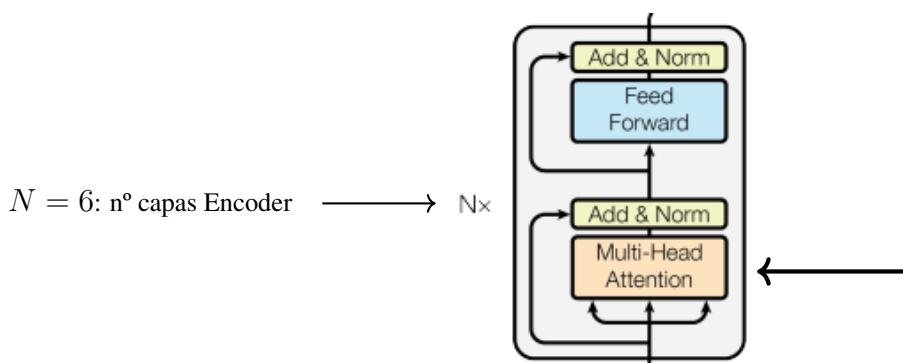


Figura 6.19: Atención múltiple. Fuente: (Vaswani et al., 2017)

La atención consiste en que el algoritmo sepa que, dada una palabra de la oración, a qué otras palabras debe prestarle atención. Por ejemplo, en la oración *El chico juega*, el Transformer, en el verbo, debe prestar atención a la palabra *chico* para poder conjugarlo bien.

Para conseguir esta atención, hay tres redes neuronales involucradas: *Query* (Q), *Key* (K) y *Value* (V). Las tres redes neuronales se van a encargar de generar un vector —con distintos propósitos— por cada palabra en la secuencia de entrada (todos estos vectores tienen la misma dimensionalidad que el vector *embedding*, i.e. $d_{model} = 512$).

Desarrollando un poco más esto, el vector de cada palabra que genera la red neuronal Q describe, por decirlo de alguna manera, las cualidades que debe cumplir otra palabra de la secuencia para que la palabra en cuestión le preste atención. El vector que genera K define una descripción de dicha palabra. En otras palabras, Q genera un vector de las características que busca una palabra para prestar atención a otra; mientras que K genera un vector de las características de cada palabra. Por tanto, las redes Q y K trabajan en conjunto para generar el vector de atención.

Una manera de ver esto de manera más visual es pensar como que cada palabra de la oración tiene una llave (vector generado por K) y un candado (vector generado por Q). Cogiendo el ejemplo de antes, esto se vería así. Véase la Tabla 8.

El				El
chico				chico
juega				juega

Tabla 8: Representación gráfica de vector *key* de cada palabra (columna de la izquierda) y el vector *query* de cada palabra (columna de la derecha).

Entonces, cada vector *query* de cada palabra, se compara con cada vector *key* de cada palabra. En decir, se está comparando las características que busca cada palabra (*query*) con la descripción del resto de palabras en la oración (*key*). Véase la Tabla 9.

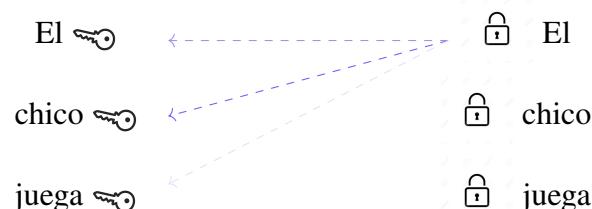


Tabla 9: Comparación entre características de la palabra “El” con la descripción del resto de palabras.

Esto se hace con todas las palabras de la oración. Quizá se vea de manera clara con un ejemplo del propio artículo. Véase la Figura 6.20:

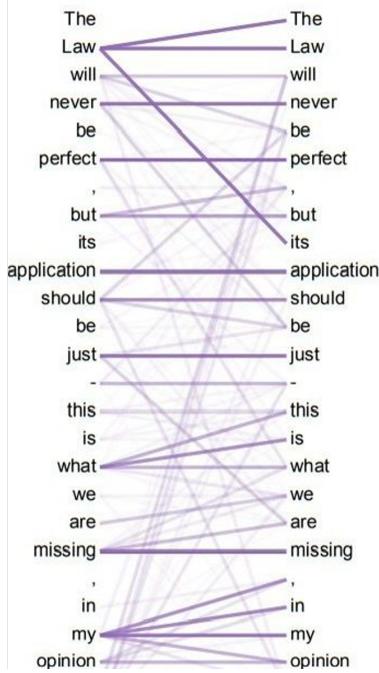


Figura 6.20: Ejemplo de *self-attention* entre todas las palabras de una secuencia. Cuanto más oscuro es el color, más atención le presta una palabra a la correspondiente. Fuente: (Vaswani et al., 2017)

De esta manera, si ambos vectores *query* y *key* son parecidos, quiere decir que las características que busca una palabra *A* son parecidas a la descripción de otra palabra *B*; lo que indica que la palabra *A* debe prestar atención a *B*. Para saber si dos vectores son parecidos, se emplea el producto escalar: $|Q| \cdot |K| \cdot \cos \theta$. Véase la Figura 6.21.

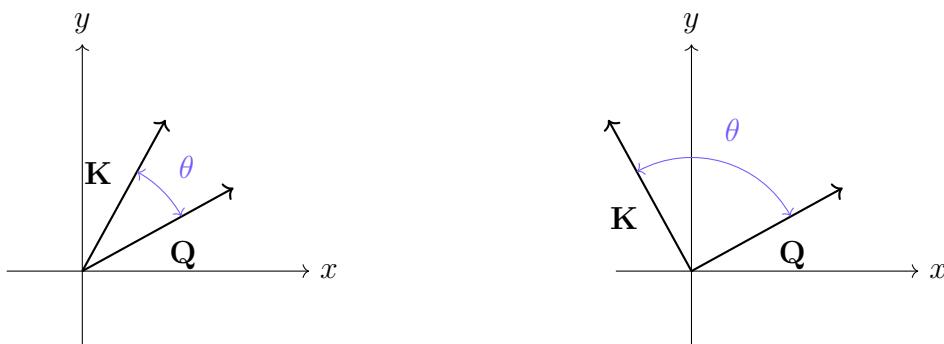


Figura 6.21: Visualización entre ángulo de vectores parejos (izquierda) y vectores dispares (derecha).

Como $\cos(\theta = 0^\circ) = 1$ y $\cos(\theta = 180^\circ) = -1$, de la Figura 6.21 se deduce que cuanto mayor sea este

producto escalar, más parecidos serán estos vectores.

Ahora, como el Transformer realiza todo esto de manera paralela, se puede pensar en Q y en K como matrices de dimensión $[S, 512]$, en donde cada fila de cada una de las matrices corresponde con el vector en cuestión de cada palabra. Las matrices Q y K son multiplicadas respectivamente por las matrices W^Q y W^K (ambas de dimensión $[512, 64]$) con el objetivo de proyectar Q y K a matrices Q' y K' de dimensión $[S, 64]$ ⁵⁴. Se procede pues, a multiplicar las matrices Q' y K' para obtener la **matriz de pesos de atención**. Para ello, hay que trasponer la matriz: $K \rightarrow K^T$:

$$\begin{array}{ccc} \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|} \hline & & \\ \hline \end{array} & = & \underbrace{\begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & \textcolor{purple}{\boxed{}} & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}}_{QK[S, S]} & \begin{array}{l} \text{Esta posición (p.e.) de la} \\ \text{matriz representa el peso de} \\ \text{atención que le da la primera} \\ \text{palabra (primera fila) a la} \\ \text{última palabra (última} \\ \text{columna) de la secuencia.} \end{array} \\ Q'[S, 64] & K'^T[64, S] & \end{array}$$

Matriz de pesos de atención

Esta matriz resultado (QK) se divide entre la raíz de la dimensión de las filas de Q' ⁵⁵ —i.e. $\sqrt{64} = 8$ — y se pasa por una función Softmax: $QK' = \text{Softmax}(QK/8)$. Esto último, hace que la suma total de atenciones que presta una palabra al resto, sea igual a uno. En otras palabras, la suma de los valores de cada fila de QK' da uno. El objetivo de esto es tratar con valores más simples y estandarizados.

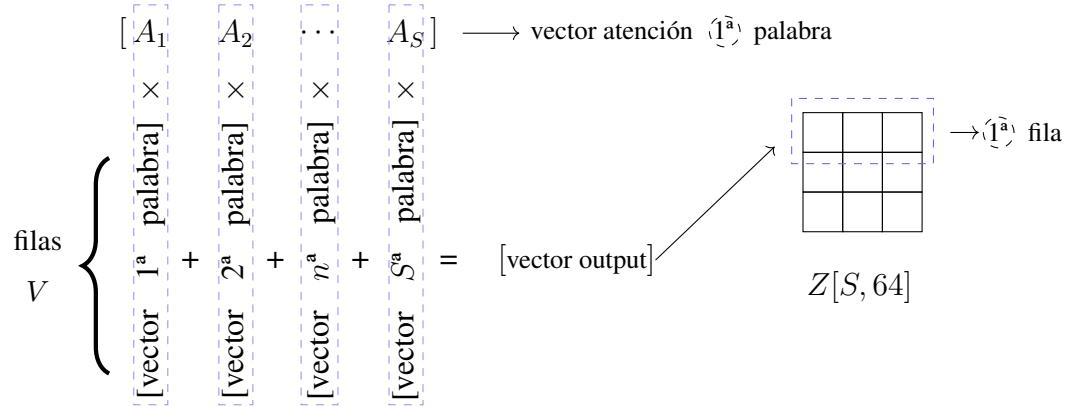
Una vez obtenidos los vectores de pesos estandarizados de cada palabra —filas de la matriz QK' —, habrá que multiplicar dichos pesos al vector valor (*value*) propio de cada palabra. La red neuronal V (*Value*) —que de la misma manera que Q y K , es una matriz de dimensión $[S, 512]$ — es la encargada de generar este vector *propio* para cada palabra en la secuencia. De nuevo, al igual que se ha hecho con Q y K , V se reduce (proyecta) a una dimensión $[S, 64]$ al ser multiplicada por una matriz de pesos $W^V[512, 64]$. De esta manera:

$$\begin{array}{ccc} \begin{array}{l} \text{vector atención} \\ \text{primera palabra} \end{array} & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} & = & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \\ QK'[S, S] & V'[S, 64] & \begin{array}{l} \text{vector } v \in \mathbb{R}^S \text{ de} \\ \text{la primera palabra} \end{array} & Z[S, 64] & \end{array}$$

⁵⁴Esto está mal explicado en muchos sitios, en donde explican que Q es el resultado de multiplicar la matriz E por la matriz W^Q . Lo mismo con las matrices K y V respectivamente. En el *paper* no se indica nada de eso.

⁵⁵Esto se hace para evitar problemas de explosión de gradiente, pues la multiplicación entre Q' y K' puede dar lugar a valores numéricos muy grandes.

Una manera más intuitiva de entender cómo están formadas las filas de la matriz QKV es la siguiente:



Hasta este punto, se ha hecho lo que se llama *atención de producto escalar escalado*. Ahora, una sola atención puede no ser suficiente; es decir, puede ser necesario —y lo es en la mayoría de los casos— prestar atención a más que simples palabras. Por ejemplo, en la frase *Me gusta la lasaña* hay que poner el foco en más de una atención (véase la Figura 6.22).



Figura 6.22: Ejemplo de atención múltiple.

En la Figura 6.22 se puede ver cómo hay relación entre las palabras *Me* y *gusta*; así como entre las palabras *la* y *lasaña*. Pero también hay una relación entre *Me gusta* y *la lasaña*. Es por esto que es necesario tener en cuenta varias dimensionalidades de la atención. Para ello, las matrices Q , K , V se repiten $h = 8$ veces en cada capa. Véase la Figura 6.23.

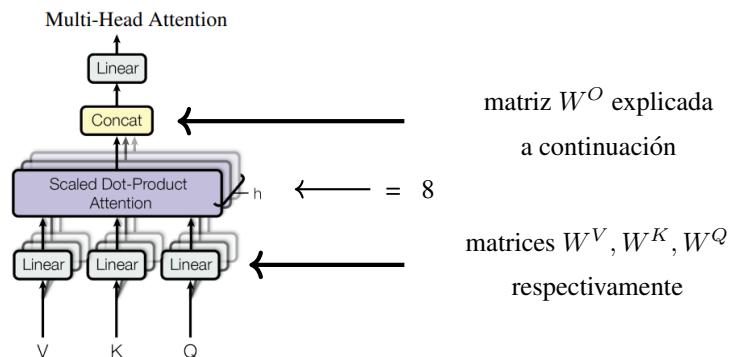


Figura 6.23: Atención llevada a más de una dimensionalidad. Fuente: (Vaswani et al., 2017)

De esta manera, se genera una nueva matriz Z por cada dimensión. Cada una de estas matrices se va concatenando a la anterior. Como esto se repite ocho veces, se acaba con una matriz $Z'[S, 64 \cdot 8]$. Esta matriz, con el fin de que tenga el mismo tamaño de entrada que la matriz de entrada en la capa de entrada —i.e. $[S, 512]$, tamaño de la matriz E —, se multiplica por otra matriz de pesos $W^O[64 \cdot 8, 512]$. Véase la Figura 6.24.

$$\begin{array}{c} \boxed{\text{ }} \\ \times \\ \boxed{\text{ }} \end{array} = \boxed{\text{ }} \\
 Z'[S, 512] \quad W^O[512, 512] \quad \text{MultiHead } [S, 512]$$

Figura 6.24: Matriz final de atención.

6.6.2.2 Bloque residual

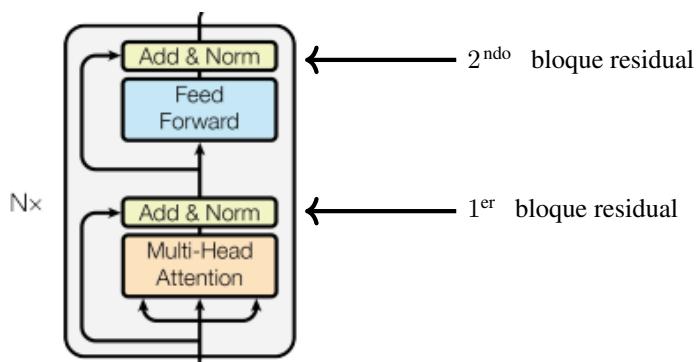


Figura 6.25: Ajuste de valores. Fuente: (Vaswani et al., 2017)

El bloque residual (He et al., 2016) se encarga de ajustar los valores para mejorar el rendimiento del entrenamiento y hacerlo más sencillo. Esto es necesario debido a la gran profundidad que tiene esta arquitectura. En la Figura 6.25, si uno se fija bien, le llegan dos *inputs*: el de la subcapa anterior y el de la anterior a esta. Es decir, al primer bloque residual —el de más abajo en la Figura 6.25— recibe la matriz de los *embeddings*⁵⁶ y la matriz *MultiHead* (Figura 6.24); ambas con la misma dimensión: $[S, 512]$. El segundo bloque residual, el de más arriba, recibe por otra parte el *output* de este primer bloque residual y el *output* de la red neuronal FFN (*Feed Forward Network*).

⁵⁶Cabe recordar que hay 6 capas en total como las de la Figura 6.25. Es decir, el bloque residual de más abajo de la

En cualquier caso, el ***output*** de todas las subcapas es siempre constante: una matriz de dimensiones $[S, 512]$. El bloque residual, por tanto, de lo único que se encarga es de sumar las dos matrices que recibe de los dos sub-bloques anteriores y las normaliza. Esto último acelera y simplifica el entrenamiento (Ioffe & Szegedy, 2015).

6.6.2.3 Red de retroalimentación

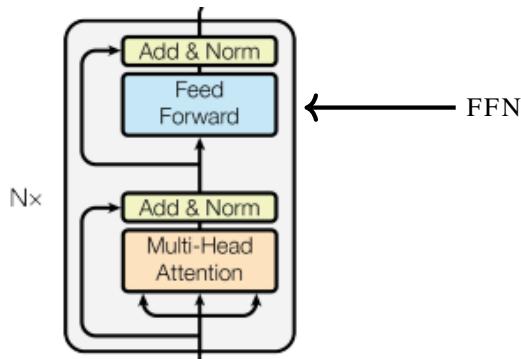


Figura 6.26: FFN. Fuente: (Vaswani et al., 2017)

Lo último por explicar del *Encoder* es la red de retroalimentación (FFN, por sus siglas en inglés). Esta red consiste en una MLP que es aplicada a cada palabra de la secuencia (oración) de manera separada. Esta red neuronal agrega no linealidad⁵⁷ al algoritmo, haciendo posible el aprendizaje de relaciones más complejas.

Cabe remarcar que los pesos de esta red neuronal FFN, al igual que el resto de redes neuronales —i.e. Q, K, V y las matrices de pesos W —, son distintos en cada capa.

6.6.3 Decoder

El *Decoder* se encarga de generar la secuencia de texto de salida. Dicha salida puede ser la traducción de un texto a otro idioma, generar un resumen, etc.

Su arquitectura es muy parecida a la del *Encoder* (Figura 6.17), con algunas leves modificaciones.

Figura 6.25 recibe la matriz de *embeddings* por ser la primera de las seis capas. En las siguientes, esto no ocurre; recibe el *output* del *Encoder* de la capa anterior.

⁵⁷Esto se consigue con las funciones de activación. En este caso se aplica una ReLu $\begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} = \max(0, x)$

Una diferencia importante entre ambos es que, el *Decoder* es secuencial. Es decir, va generando texto palabra a palabra (*token* a *token*) a partir del *input* de una manera secuencial. Para ello, la información del *Encoder* va a ser de inmensa ayuda para que el *Decoder* sepa a qué debe prestar atención según la palabra que esté procesando, con el fin de que el texto que genere sea lo más preciso posible.

Por último, el *Decoder*, al igual que el *Encoder*, tiene seis capas, una seguida de la anterior.

Nota

Las subcapas de bloque residual y de FFN no se van a explicar, pues funcionan igual que en el *Encoder* y ya han sido explicadas en dicha sección.

6.6.3.1 Outputs

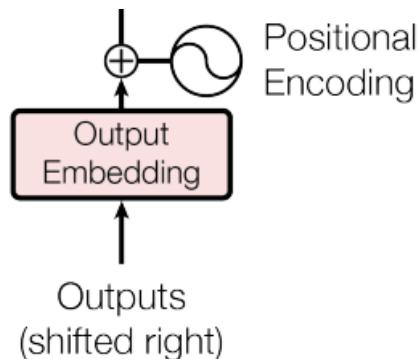


Figura 6.27: Outputs. Fuente: (Vaswani et al., 2017)

Algo muy importante a destacar es que el *Decoder* es algo especial, pues a diferencia del *Encoder*, este se comporta de una manera distinta en el entrenamiento a cómo se comporta luego a la hora de generar texto una vez ya se ha entrenado.

El *Decoder* en la **fase de entrenamiento** recibe como primer *input*⁵⁸ un *token* de iniciación⁵⁹. Este *token* le hace saber al *Decoder* que debe empezar a generar texto. Para predecir la primera palabra, como no tiene información de predicciones anteriores todavía, utiliza únicamente la información que recibe del *Encoder* (véase Figura la 6.30). A partir de aquí, el entrenamiento sigue una extensión de la técnica *teacher forcing* (Goodfellow et al., 2016, 10.2.1). El objetivo de esta técnica es que, junto con

el algoritmo de *backpropagation*, el *Decoder* sea capaz de aprender.

Partiendo del contexto que es la fase de entrenamiento y, por tanto, el *Decoder* tiene acceso al texto que **real** —i.e. el que tiene que tratar de predecir correctamente—, el *teacher forcing* consiste en que el *Decoder*, independientemente de la palabra que prediga, recibirá como siguiente *input* la palabra que debería haber predicho. Para ver esto de una manera más clara, se pone un ejemplo muy sencillo:

Ejemplo *teacher forcing*

Supóngase que el texto a predecir es Ayer hizo mucho frío. El primer *input* que recibe el *Decoder* ($t = 0$)⁶⁰ es un *token* de iniciación. A partir de este momento, la siguiente entrada que recibe, i.e. $t = 1$, será la palabra Ayer, independientemente de la predicción que haya hecho el *Decoder* en $t = 0$. De la misma manera, en $t = 2$, el *Decoder* recibe como *input* la palabra hizo —pues es la que debería haber predicho en $t = 1$ —.

t de tiempo.

El *Decoder*, en la arquitectura de Transformer, sigue una extensión de esto. Es decir, que los *inputs* que va recibiendo se van concatenando con los *outputs* que este va generando (véase la Figura 6.28).

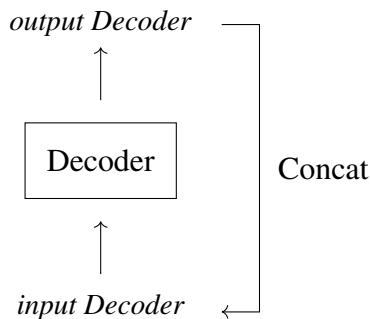


Figura 6.28: Concatenación Decoder.

De esta manera, en el ejemplo anterior, el *Decoder* primero recibe en *token* de iniciación ($t = 0$), luego la secuencia de *token* de iniciación + Ayer ($t = 1$), seguido de *token* de iniciación + Ayer + hizo ($t = 2$), etc. Nótese que, debido al *token* de iniciación, la secuencia está desplazada hacia la derecha. Eso es lo que quiere decir *shifted right* de la Figura 6.27.

⁵⁹En la Figura 6.27 viene la palabra *output* y no *input* para hacer referencia a que el *Decoder* genera la salida del modelo.

Se empleará la palabra *input* porque se cree que queda más claro de esta manera.

⁶⁰Este *token* también es conocido como <SOS> (*Start Of Sequence*).

El *Decoder* en la **fase de generación de texto** se comporta algo distinto, pues, por motivos obvios, no tiene acceso al texto *real* que debe predecir. La modificación aquí está en que ahora, en vez de ir recibiendo los *tokens* “reales”, lo que va concatenando son los *tokens* que va prediciendo cada instante t .

El otro aspecto importante a cubrir en esta sección es la **codificación posicional**. Se puede pensar que el *Decoder*, al ser secuencial, no necesita esta codificación. Pero, sí que la necesita porque, aunque sea secuencial en el sentido de que va generando palabras una a una, según las va generando, va recibiendo concatenaciones de palabras generadas hasta ese momento. Estas palabras que el *Decoder* recibe, las recibe de manera paralela —de la misma manera que pasaba con el *Encoder*—. Por tanto, la codificación posicional sigue siendo necesaria.

De la misma manera que con el *Encoding*, el vector de *embedding* de cada palabra que va recibiendo el *Decoder*, se suma a su vector posicional. De nuevo, estos vectores se concatenan en una matriz E' de dimensiones $[S', 512]$ siendo S' en número de *tokens* concatenados en un instante t_i de tiempo.

6.6.3.2 Enmascaramiento de atención

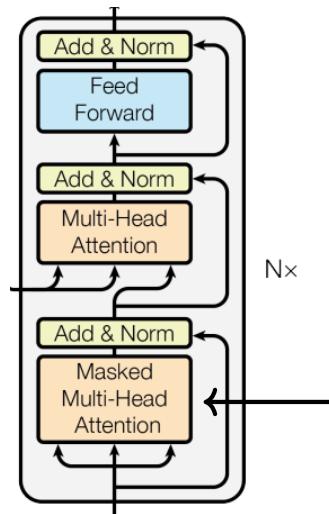


Figura 6.29: Subcapa de enmascaramiento. Fuente: (Vaswani et al., 2017)

El enmascaramiento tiene que ver con la propiedad que en el *paper* denominan auto-regresiva. Esto, lo que viene decir es que, para que el modelo genere un texto coherente, cada palabra concatenada que recibe el *Decoder*, no tiene acceso a información de palabras que estén a su derecha en la secuencia, i.e. palabras que hayan sido añadidas a la secuencia después de la palabra en cuestión.

En otras palabras, siguiendo con el ejemplo de antes, Ayer hizo mucho frío, supóngase que el *Decoder* está prediciendo bien. Hasta ahora ha predicho las palabras *Ayer* e *hizo*. Bien, pues el enmascaramiento, de lo que se va a encargar es de que la palabra *Ayer*, no tenga acceso a la información de que la palabra *hizo* se ha añadido a la secuencia, pues está a su derecha en la secuencia. En cambio, *hizo* sí que tiene acceso a la palabra *Ayer*, pues la palabra *hizo* está a la derecha de la palabra *Ayer* en la secuencia de *input* del *Decoder*.

En esta subcapa se genera la matriz Q . Es decir, un vector de **búsqueda de características** de cada palabra en la secuencia de *input*. Estos vectores que componen las filas de la matriz Q no son iguales al del *Decoder* por la restricción del enmascaramiento.

6.6.3.3 Atención Decoder

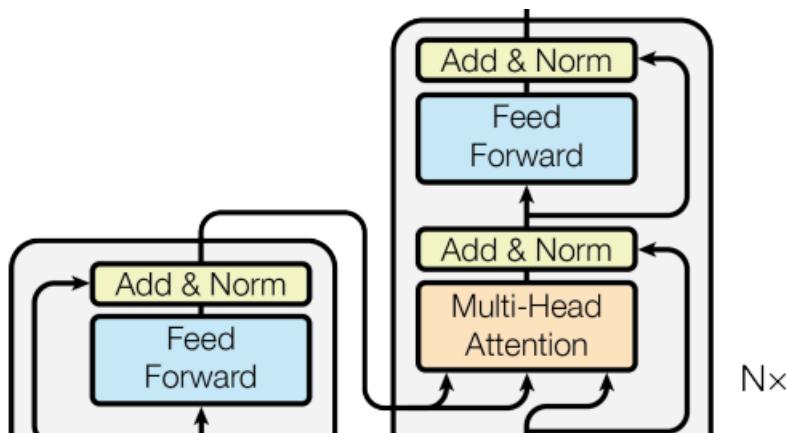


Figura 6.30: Atención Decoder. Fuente: (Vaswani et al., 2017)

El bloque de atención del *Decoder* funciona sutilmente distinto al que se ha visto con anterioridad. Esta subcapa recibe dos *inputs*:

- Por una parte, como se ve en la imagen 6.30, recibe información de la última capa del *Encoder*. Esta información en concreto, son las matrices K y V de esta capa del *Encoder*. Esto se hace con el motivo de aportar contexto de la palabra que se está procesado (y esto ocurre para las seis capas del *Decoder*). En cierta manera, esto permite que una palabra dada, e.g. Ayer, aunque no tenga acceso a información de palabras posteriores en su secuencia, el *Decoder* siga teniendo acceso a un contexto global calculado en el *Encoder*. Esta transmisión de información de contexto global por parte del *Encoder* al *Decoder* se denomina *Atención* y es distinto al concepto

de *self-attention* (sección 6.6.2).

- Por otra parte, esta subcapa recibe la matriz Q de la subcapa de enmascaramiento.

De esta manera, para predecir el *output* se hace lo mismo que se ha hecho en el *Encoder*. En este punto, el *Decoder* ya tiene la matrices Q , K y V de dimensiones $[S', 512]$ cada una de ellas. Se multiplican por unas matrices de pesos W^Q , W^K y W^V respectivamente para obtener una matriz Z de dimensión $[S', 64]$. Esta atención se repite $h = 8$ veces para obtener distintos grados de atenciones, y finalmente, al concatenar cada una de las matrices Z de cada dimensión de atención, se multiplica por otra matriz de pesos W^O de dimensión $[64 \cdot 8, 512]$ para obtener la dimensión consistente de salida $[S', 512]$.

Esta matriz resultado sigue el mismo proceso. Se pasa a un bloque residual que recibe este *input* y el *output* de la subcapa anterior. Luego se pasa por una FFN y posteriormente por otro bloque residual.

Este proceso se repite $N = 6$ veces, pues hay seis bloques de *Decoder*.

6.6.4 Salida de Transformer

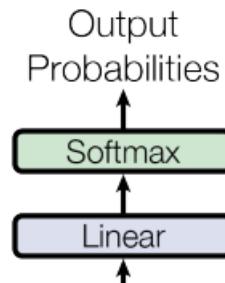


Figura 6.31: Salida de Transformer. Fuente: (Vaswani et al., 2017)

Finalmente, la salida de la última capa del *Decoder*, es una matriz de dimensión $[S', 512]$. Dicha matriz se pasa por una función lineal que se va a encargar de convertir esta información en un único vector de D dimensiones, siendo D el número de palabras del idioma al que se esté generando el texto. Es decir, que cada posición de este vector, corresponde a una palabra del idioma en cuestión —este idioma puede ser el mismo que el de la secuencia de entrada si, por ejemplo, la función del *Decoder* es generar un resumen; o, por otro lado, puede ser otro idioma si es un problema de traducción de texto—.

Finalmente, este vector se pasa por una función Softmax que pasa los valores de este vector a probabilidades, de tal manera que la suma de todos los valores de este vector sume uno. El output será aquella palabra que corresponda a la posición con mayor probabilidad del vector resultante de aplicar la función Softmax.

6.7 Informer

6.7.1 Codificación Informer

La codificación que hace el Informer en los datos de entrada es bastante completa. Y es que, una peculiaridad que tienen los datos temporales, es que su posición lleva asociado un *timestamp*, el cual indica el minuto, hora, día, mes, etc. en el que el dato se ha recogido. Todos estos *timestamps* aportan información adicional que un algoritmo puede exprimir.

El Informer hace tres codificaciones distintas; cada una aportando una información distinta.

- Como ya se ha comentado en la sección del Informer 2.1.3, el contexto local en series temporales es fundamental. Para aportar esta información local al algoritmo, Informer (H. Zhou et al., 2021) realiza una codificación fija (*fixed*), de la misma manera que se hace en el Transformer original, i.e. con senos y cosenos, para la posición de cada uno de los datos en la serie temporal.
- Por otra parte, emplea el *timestamp* para aportar un contexto global. Esto se hace a través de una codificación “aprendible” (*learnable embedding*).
- Por último, cada dato de la serie temporal se proyecta en un vector de d_{model} dimensiones (mismas dimensiones que los vectores de codificación anteriores) para poder añadir esta información a las dos codificaciones anteriores.

Véase la Figura 6.32 para poder ver esto de una manera más visual.

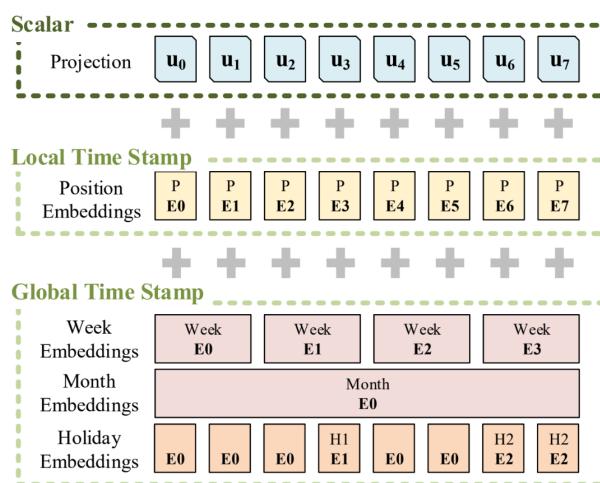


Figura 6.32: Codificación completa del Informer. Fuente: (H. Zhou et al., 2021)

6.7.2 Algoritmo ProbSparse

Este algoritmo surge de la necesidad de reducir la complejidad con la que se calcula la atención en el Transformer. La idea intuitiva detrás de *ProbSparse*, es que los autores se dieron cuenta de que solo algunos productos escalares entre las matrices Q y K proporcionaban la atención más relevante. Véase la Figura 6.33.

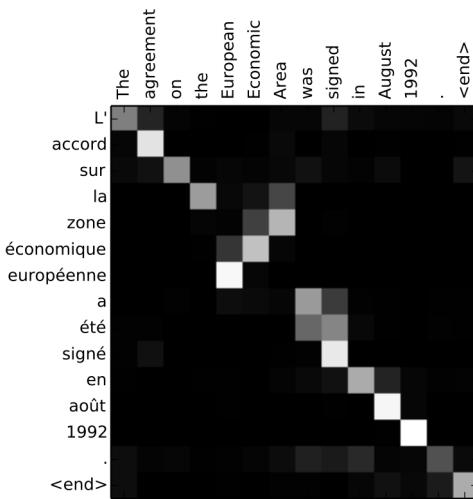


Figura 6.33: Ejemplo matriz de atención. Fuente: (Bahdanau et al., 2014)

En la Figura 6.33, cuanto más clara es una celda, se traduce en un producto escalar entre q_i y k_j con una alta atención. Es fácil ver que, en su gran mayoría, las celdas son más bien oscuras. Esto se puede ver de manera más rigurosa en la Figura 6.34.

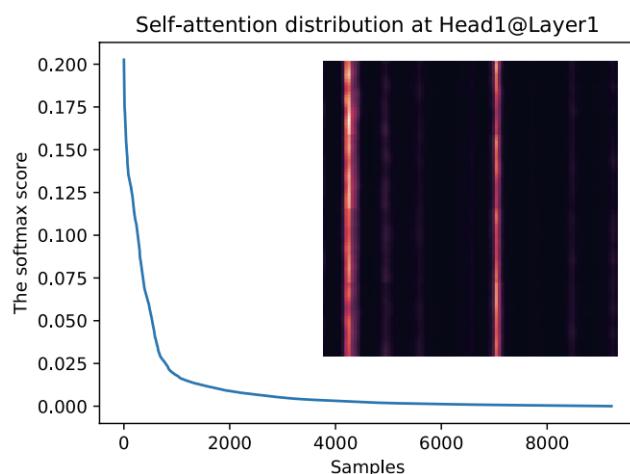


Figura 6.34: Distribución de las atenciones con forma de cola larga. Fuente: (H. Zhou et al., 2021)

En la Figura 6.34 se puede ver cómo solamente hay unos pocos productos escalares entre los vectores q_i y k_j que aportan una alta atención —i.e. un valor de Softmax alto—. Un producto escalar con una atención alta induce a que la distribución del vector q_i asociado a dicho producto escalar sea muy diferente a una distribución uniforme (H. Zhou et al., 2021). Esto es porque si un vector q_i sigue una distribución uniforme, se traduce en que no “busca” prestar atención a ningún dato en la secuencia en concreto, y por ende se puede no tener en cuenta para computar la atención.

Para determinar si la distribución de un vector q_i sigue una distribución que se asemeja a una distribución uniforme, se emplea la divergencia de Kullback-Leibler (Han, n.d.). Aunque su explicación va más allá del contenido de este trabajo, basta con saber que cuanto mayor sea q_i , más discrepancia hay entre su distribución y la distribución uniforme; lo que implica que dicho vector q_i es relevante a la hora de calcular la atención. A este proceso se le denomina *Query Sparsity Measurement* y está recogido en M en la Figura 6.35.

Como el objetivo es reducir la complejidad de la atención del Transformer original, seleccionando los q_i con mayor importancia no es suficiente, pues para calcular estos, hay que realizar el producto escalar entre Q y K , lo cual sigue siendo manteniendo la misma complejidad ($\mathcal{O}(L^2)$). Para ello, se seleccionan U vectores aleatorios k_j de la matriz K . Esta muestra se considera representativa. En la Figura 6.35 se puede ver un pseudo-código de estos pasos recientemente explicados.

Algorithm 1 ProbSparse self-attention

Require: Tensor $\mathbf{Q} \in \mathbb{R}^{m \times d}$, $\mathbf{K} \in \mathbb{R}^{n \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$

- 1: **print** set hyperparameter c , $u = c \ln m$ and $U = m \ln n$
- 2: randomly select U dot-product pairs from \mathbf{K} as $\bar{\mathbf{K}}$
- 3: set the sample score $\bar{\mathbf{S}} = \mathbf{Q}\bar{\mathbf{K}}^\top$
- 4: compute the measurement $M = \max(\bar{\mathbf{S}}) - \text{mean}(\bar{\mathbf{S}})$ by row
- 5: set Top- u queries under M as $\bar{\mathbf{Q}}$
- 6: set $\mathbf{S}_1 = \text{softmax}(\bar{\mathbf{Q}}\bar{\mathbf{K}}^\top/\sqrt{d}) \cdot \mathbf{V}$
- 7: set $\mathbf{S}_0 = \text{mean}(\mathbf{V})$
- 8: set $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_0\}$ by their original rows accordingly

Ensure: self-attention feature map \mathbf{S} .

Figura 6.35: Pseudo-código algoritmo ProbSparse. Fuente: (H. Zhou et al., 2021).

En el paso 2), se seleccionan U vectores aleatorios de K . Por lo general, m y n suelen ser ambos iguales: la longitud de la secuencia de entrada denotada como L . Por tanto, se seleccionan $L \ln L$

vectores aleatorios de K . Con esto, se puede calcular M y seleccionar los u vectores q_i que más discrepancia tengan con una distribución uniforme —i.e. lo que tengan un valor más alto—. Con esto se consigue reducir la complejidad del cálculo de la atención de $(\mathcal{O}(L^2))$ a $\mathcal{O}(L \ln L)$.

Cabe remarcar que el Informer mantiene la *Multi-head Attention* del Transformer original; lo que implica que el algoritmo de la Figura 6.35 se calcula varias veces por cada capa de atención, evitando de esta manera información redundante, y obteniendo distintos vectores $q \in \overline{Q}$, así como distintos vectores aleatorios $k \in \overline{K}$.

6.7.3 Comparativa Informer

Se muestra una tabla que compara el Informer con algunos algoritmos estado del arte hasta la fecha: LogTrans(i.e. LogSparse) (Li et al., 2019), Reformer (Kitaev et al., 2020), LSTM⁶¹ (Bahdanau et al., 2014), DeepAR (Salinas et al., 2017), ARIMA (Box et al., 2015), Prophet (Taylor & Letham, 2018).

Methods	Informer	Informer [†]	LogTrans	Reformer	LSTM ^a	DeepAR	ARIMA	Prophet	
Metric	MSE MAE	MSE MAE	MSE MAE	MSE MAE	MSE MAE	MSE MAE	MSE MAE	MSE MAE	
ETTh ₁	24	0.098 0.247	0.092 0.246	0.103 0.259	0.222 0.389	0.114 0.272	0.107 0.280	0.108 0.284	0.115 0.275
	48	0.158 0.319	0.161 0.322	0.167 0.328	0.284 0.445	0.193 0.358	0.162 0.327	0.175 0.424	0.168 0.330
	168	0.183 0.346	0.187 0.355	0.207 0.375	1.522 1.191	0.236 0.392	0.239 0.422	0.396 0.504	1.224 0.763
	336	0.222 0.387	0.215 0.369	0.230 0.398	1.860 1.124	0.590 0.698	0.445 0.552	0.468 0.593	1.549 1.820
	720	0.269 0.435	0.257 0.421	0.273 0.463	2.112 1.436	0.683 0.768	0.658 0.707	0.659 0.766	2.735 3.253
ETTh ₂	24	0.093 0.240	0.099 0.241	0.102 0.255	0.263 0.437	0.155 0.307	0.098 0.263	3.554 0.445	0.199 0.381
	48	0.155 0.314	0.159 0.317	0.169 0.348	0.458 0.545	0.190 0.348	0.163 0.341	3.190 0.474	0.304 0.462
	168	0.232 0.389	0.235 0.390	0.246 0.422	1.029 0.879	0.385 0.514	0.255 0.414	2.800 0.595	2.145 1.068
	336	0.263 0.417	0.258 0.423	0.267 0.437	1.668 1.228	0.558 0.606	0.604 0.607	2.753 0.738	2.096 2.543
	720	0.277 0.431	0.285 0.442	0.303 0.493	2.030 1.721	0.640 0.681	0.429 0.580	2.878 1.044	3.355 4.664
ETTm ₁	24	0.030 0.137	0.034 0.160	0.065 0.202	0.095 0.228	0.121 0.233	0.091 0.243	0.090 0.206	0.120 0.290
	48	0.069 0.203	0.066 0.194	0.078 0.220	0.249 0.390	0.305 0.411	0.219 0.362	0.179 0.306	0.133 0.305
	96	0.194 0.372	0.187 0.384	0.199 0.386	0.920 0.767	0.287 0.420	0.364 0.496	0.272 0.399	0.194 0.396
	288	0.401 0.554	0.409 0.548	0.411 0.572	1.108 1.245	0.524 0.584	0.948 0.795	0.462 0.558	0.452 0.574
	672	0.512 0.644	0.519 0.665	0.598 0.702	1.793 1.528	1.064 0.873	2.437 1.352	0.639 0.697	2.747 1.174
Weather	24	0.117 0.251	0.119 0.256	0.136 0.279	0.231 0.401	0.131 0.254	0.128 0.274	0.219 0.355	0.302 0.433
	48	0.178 0.318	0.185 0.316	0.206 0.356	0.328 0.423	0.190 0.334	0.203 0.353	0.273 0.409	0.445 0.536
	168	0.266 0.398	0.269 0.404	0.309 0.439	0.654 0.634	0.341 0.448	0.293 0.451	0.503 0.599	2.441 1.142
	336	0.297 0.416	0.310 0.422	0.359 0.484	1.792 1.093	0.456 0.554	0.585 0.644	0.728 0.730	1.987 2.468
	720	0.359 0.466	0.361 0.471	0.388 0.499	2.087 1.534	0.866 0.809	0.499 0.596	1.062 0.943	3.859 1.144
ECL	48	0.239 0.359	0.238 0.368	0.280 0.429	0.971 0.884	0.493 0.539	0.204 0.357	0.879 0.764	0.524 0.595
	168	0.447 0.503	0.442 0.514	0.454 0.529	1.671 1.587	0.723 0.655	0.315 0.436	1.032 0.833	2.725 1.273
	336	0.489 0.528	0.501 0.552	0.514 0.563	3.528 2.196	1.212 0.898	0.414 0.519	1.136 0.876	2.246 3.077
	720	0.540 0.571	0.543 0.578	0.558 0.609	4.891 4.047	1.511 0.966	0.563 0.595	1.251 0.933	4.243 1.415
	960	0.582 0.608	0.594 0.638	0.624 0.645	7.019 5.105	1.545 1.006	0.657 0.683	1.370 0.982	6.901 4.264
Count	32	12	0	0	0	6	0	0	

Figura 6.36: Tabla comparativa Informer. Fuente: (H. Zhou et al., 2021)

Nota

El Informer[†] de la segunda columna de la tabla 6.36 es la arquitectura del Informer sin haber modificado la atención, i.e. sin haber aplicado el algoritmo de *ProbSparse*.

⁶¹RNN + atención.

Referencias

- AWS. (2006). *Aws*. Retrieved June 1, 2024, from <https://aws.amazon.com/es/>
- Azure, M. (2009). *Microsoft azure*. Retrieved June 1, 2024, from <https://azure.microsoft.com/>
- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.
- Bennett, J., Lanning, S., et al. (2007). The netflix prize. *Proceedings of KDD cup and workshop, 2007*, 35.
- Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control*. John Wiley & Sons.
- Bramer, M. (2007). Avoiding overfitting of decision trees. *Principles of data mining*, 119–134.
- Breiman, L. (1997). *Arcing the edge* (tech. rep.). Citeseer.
- Brockwell, P. J., & Davis, R. A. (1991). *Time series: Theory and methods*. Springer science & business media.
- Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning*, 161–168.
- Chang, W., Liu, Y., Xiao, Y., Yuan, X., Xu, X., Zhang, S., & Zhou, S. (2019). A machine-learning-based prediction method for hypertension outcomes based on medical data. *Diagnostics*, 9(4), 178.
- Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 785–794.
- Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.
- Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

- Confederación Española de Asociaciones de Jóvenes Empresarios (CEAJE). (2024). *3 de cada 10 españoles quieren ser emprendedores*. <https://ceaje.es/3-de-cada-10-espanoles-quieren-ser-emprendedores/>
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.
- Duignan, B. (2024, January). *Ockham's razor*. Encyclopedia Britannica. <https://www.britannica.com/topic/Occams-razor>
- Esteve, E. H. (2002). La historia de la contabilidad. *Revista De Libros De La Fundación Caja Madrid*, (67/68), 25–28.
- Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. *icml*, 96, 148–156.
- Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119–139.
- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), 1189–1232. <https://doi.org/10.1214/aos/1013203451>
- GCP. (2009). *Gcp*. Retrieved June 1, 2024, from https://cloud.google.com/?hl=es_419
- Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *International conference on machine learning*, 1243–1252.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- Han, J. (n.d.). Kullback-leibler divergence.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.
- Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd). OTexts.
- Instituto de Contabilidad y Auditoría de Cuentas (ICAC). (2023). 50 aniversario del plan general de contabilidad: Reflexiones y desafíos. https://www.icac.gob.es/sites/default/files/2023-11/1.8.%20PGC-50-aniversario_Pich.pdf
- Instituto Nacional de Estadística (INE). (2023). Directorio central de empresas y establecimientos (dirce) 2023.
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

- Jordan, M. (1986). *Serial order: A parallel distributed processing approach. technical report, june 1985-march 1986* (tech. rep.). California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.
- Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the limits of language modeling.
- Kazemnejad, A. (2019). Transformer architecture: The positional encoding. *kazemnejad.com*. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.
- Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.
- Knowledge, C. (2024). Data type conversion in copy data activity — convert data types from json code — csv to azure sql db. https://www.youtube.com/watch?v=vB446EB_-aU
- Lässig, F. (2019). *Temporal convolutional networks and forecasting*. <https://unit8.com/resources/temporal-convolutional-networks-and-forecasting/>
- Lea, C., Vidal, R., Reiter, A., & Hager, G. D. (2016). Temporal convolutional networks: A unified approach to action segmentation. *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III* 14, 47–54.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32.
- Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.
- McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5, 115–133.
- Microsoft. (2024). Incremental refresh in power bi. <https://learn.microsoft.com/en-us/power-bi/connect-data/incremental-refresh-overview>
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.
- Mills, T. C. (2019). *Applied time series analysis: A practical guide to modeling and forecasting*. Academic press.
- Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.
- Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7, 21.
- Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1, 81–106.
- Rajadell Carreras, M., Trullàs, O., & Simó Guzmán, P. (2014). Contabilidad para todos: Introducción al registro contable.
- Rivera Resina, F. J., et al. (2018). Aplicación de business intelligence en una pequeña empresa mediante el uso de power bi.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533–536.
- Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29.
- Salinas, D., Flunkert, V., Gasthaus, J., & Deep, A. (2017). Probabilistic forecasting with autoregressive recurrent networks. 2017.
- Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5, 197–227.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.
- Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37–45.

- Tech, A. P. (2024a). Aget started with azure data factory in 2024. <https://www.youtube.com/watch?v=xBJbvTAi5lY>
- Tech, A. P. (2024b). Blinked service to azure sql db using managed identity. <https://www.youtube.com/watch?v=gc5mWkRPfWM>
- Tech, A. P. (2024c). Chow to copy a csv file from blob storage to azure sql db. <https://www.youtube.com/watch?v=07A3LPfiu18>
- Tech, A. P. (2024d). Dhow to use data factory triggers - easy to understand! <https://www.youtube.com/watch?v=p78BJ3A-EvQ&t=1313s&pp=ygUPYWxla3NpIHRyaWdnZXJz>
- TechBrothersIT. (2024). How to perform upsert insert/update with copy activity in azure data factory — adf tutorial 2022. <https://www.youtube.com/watch?v=fegEN1Z1viM>
- Todorovic, D. (2008). Gestalt principles. *Scholarpedia*, 3(12), 5345.
- Treisman, A. M., & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive psychology*, 12(1), 97–136.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2022). Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*.
- Werbos, P., & John, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences /.
- Wu, H., Xu, J., Wang, J., & Long, M. (2021). Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34, 22419–22430.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- Ysidora, P., Espinoza, A. I., & López, R. V. (2016). La importancia de la contabilidad en las empresas. *Contribuciones a la Economía*, 14(3), 7.
- Zebra BI. (2024). Zebra bi. <https://zebrabi.com/pbi-pro-trial/>
- Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., & Eickhoff, C. (2021). A transformer-based framework for multivariate time series representation learning. *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2114–2124.

- Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI conference on artificial intelligence*, 35(12), 11106–11115.
- Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., & Jin, R. (2022). Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. *International Conference on Machine Learning*, 27268–27286.