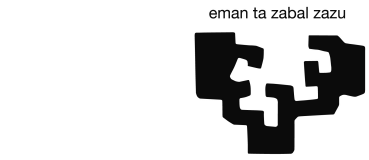

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

Proyecto de Fin de Máster

Desarrollo de soluciones automáticas para datos tabulares mediante Deep Learning

---

Autor

Iker Moya González

Director

Ibai Gurrutxaga Goikoetxea

2019

Agradecimientos

Primeramente quiero agradecer el que este proyecto haya llegado a buen término por toda la ayuda y el apoyo de mi director *Ibai Gurrutxaga*.

Agradezco a mi madre, *Maite*, por su paciencia, los consejos y el apoyo que he recibido de ella.

También, quiero dar las gracias a *Skootik* por haber confiado y dado la oportunidad de trabajar en este proyecto, y con especial mención, a *Borja Fernández* por las recomendaciones obtenidas.

Y en último lugar pero no menos importante, te agradezco a ti, por dedicarme parte de tu tiempo, por leer y consultar esta memoria.

Resumen

En este proyecto se ha realizado un módulo automático que procesa y adapta los datos de entrada, genera una red neuronal adecuada para predecir los resultados para un problema específico.

Con el objetivo de obtener los mejores resultados, se crea una red multicapa totalmente conectada dinámica en tamaño (nº de capas y neuronas) y características de la red (*learning rate*, *dropout* y *batch normalization*) en base a un histórico de datos.

La experimentación se ha realizado sobre una GPU para acelerar el proceso de entrenamiento.

Finalmente, se comparan los resultados obtenidos frente a algoritmos de *Machine learning tradicionales*. Generalmente la red neuronal se equipara con el mejor algoritmo tradicional para cada base de datos, y en algunos casos, se obtiene el mejor rendimiento.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

Este proyecto se centra en el campo del aprendizaje profundo (*deep learning*). Este campo engloba un conjunto de técnicas y algoritmos que son comúnmente utilizados para procesar grandes cantidades de datos.

La finalidad de este procesamiento es el desarrollo de sistemas que puedan ser utilizados para resolver problemas de ámbitos tan diversos como la biomedicina, la robótica y la visión por computador.

En multitud de ocasiones la naturaleza de los datos es muy diversa y a pesar de lo que se pueda pensar, en el mundo digital también se producen fallos.

Por ello, a veces los datos se encuentran en diferentes fuentes de información, contienen valores redundantes, valores faltantes, valores extremos, inconsistencias y otra serie de errores que generan lo que denominamos un dato "sucio".

La calidad de los datos sin duda es determinante a la hora de obtener unos resultados fiables y no disponer de unos datos fiables agregará una ardua tarea previa de preprocesamiento de los mismos.

En este proyecto se presenta un módulo automático que inicialmente carga los datos desde diferentes formatos (xml, csv, json, etc.), detecta los diferentes tipos de variables, los preprocesa con el fin de obtener unos datos limpios y correctamente estructurados y formateados, genera una red neuronal y la entrena con el objetivo de estimar futuras entradas.

A continuación, se presentan cuáles han sido los objetivos propuestos y la manera en la

que se ha planificado el proyecto para cumplirlos.

1.1. Objetivos del proyecto

Los principales objetivos de este proyecto son los siguientes:

- Realizar la carga de datos desde diferentes fuentes. La naturaleza de las fuentes puede ser tanto diferentes formatos de archivos como conectarse a bases de datos relacionales para obtener acceso a los datos.
- Ser capaz de implementar un preprocesamiento automatizado de datasets tabulares.
- Desarrollar un kernel que sea capaz de construir redes neuronales dinámicas automáticas que se adapten al problema presentado.
- Realizar un análisis de los resultados obtenidos con la red neuronal construída por el módulo automático frente a diversas técnicas de *Machine learning*.
- Finalmente, implementar un módulo de Deep Learning automatizado para AlchemyML que sea capaz de hacer frente a los datos de los clientes.

1.2. Motivación del proyecto

Alchemy Machine Learning, S.L. es una startup guipuzcoana fundada en 2018 e incubada por *Skootik Mobile Technologies, S.L.*, empresa de base tecnológica creada en 2018. Actualmente, *Alchemy Machine Learning, S.L.* (en adelante denominada AlchemyML) tiene su sede en el parque tecnológico de Miramón, Donostia - San Sebastián.

Fundada con el objetivo de dar solución a las dificultades tecnológicas a las que se enfrentan las empresas para seguir siendo competitivas en el mercado, el modelo de negocio de Skootik se centra en el tratamiento y gestión de grandes volúmenes de datos, así como en el desarrollo de soluciones avanzadas de análisis e inteligencia artificial.

La empresa ya cuenta con un módulo de *Machine Learning* para tratar y evaluar datasets tabulares. Por lo que, querían realizar un módulo parecido evaluando mediante *deep learning*. Finalmente, desean contrastar los resultados de los dos tipos de ejecuciones.

1.3. Fases del proyecto

El proyecto se ha planificado en tres fases: aprendizaje, implementación y experimentación.

La fase de aprendizaje ha servido para adquirir conocimiento el campo en el preproceso de datos y en la automatización de procesos.

Una vez adquirido este conocimiento se ha podido avanzar al resto de fases.

En los siguientes apartados se describe de manera más detallada cuáles han sido las tareas realizadas en cada fase.

1.3.1. Aprendizaje

Las tareas llevadas a cabo en esta fase han sido:

- Realizar un estudio de los diferentes formatos y características de los ficheros de datasets tabulares.
- Investigar diversas arquitecturas usadas para trabajar con estos datasets.
- Aprender los procedimientos más habituales utilizados en el preprocesamiento de datos y la automatización de estos.

1.3.2. Implementación

La fase de implementación se ha centrado en generar una serie de scripts para crear un programa que vuelque los datos en un dataframe, los preprocese y transforme para adaptarlos y lograr un dataset limpio y estructurado, cree una red neuronal totalmente conectada, la entrene y sea capaz de predecir futuras entradas.

Los detalles de esta etapa se presentan en el apartado 5.

1.3.3. Experimentación

La fase de experimentación ha consistido en la ejecución del módulo generado para las bases de datos seleccionados en la sección 4.2 y adquirir los resultados producidos por las diferentes combinaciones.

Una vez obtenidos, la siguiente tarea ha consistido en la creación de tablas que faciliten el análisis de estos datos. Las conclusiones obtenidas de este análisis se presentan en el capítulo 6 acompañadas de las correspondientes gráficas.

2. CAPÍTULO

Conceptos básicos sobre *Deep Learning*

Deep learning (aprendizaje profundo) es un conjunto de algoritmos de aprendizaje automático jerárquicos con unidades de procesamiento no lineal que permiten aprender múltiples niveles de representación de la entrada que se corresponden con diferentes niveles de abstracción.

Por su estructura en capas formadas por unidades de proceso relativamente simples, es un paradigma que se sitúa en el mundo de las redes neuronales artificiales.

Estas contienen un número elevado de capas (al menos dos transformaciones intermedias) y pueden ser utilizadas directamente sobre la información de entrada ya que son capaces de generar automáticamente las características sobre las que realizar la previsión a futuro de un evento.

Hoy en día, las redes neuronales y el aprendizaje profundo proporcionan las mejores soluciones a muchos problemas en reconocimiento de imágenes, reconocimiento de voz y procesamiento del lenguaje natural.

Pero, ¿qué se entiende por una red neuronal artificial?

"Una red neuronal artificial es un sistema de computación compuesto por un gran número de elementos de proceso simples, denominados **neuronas**, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas. Estos elementos siguen una organización jerárquica y están interconectados entre si" [Red, 2001].

2.1. Elementos básicos

2.1.1. Elementos básicos que componen una red neuronal

Para explicar los elementos básicos de una red neuronal en la figura 2.4 se presenta, a modo de ejemplo, la estructura una red neuronal.

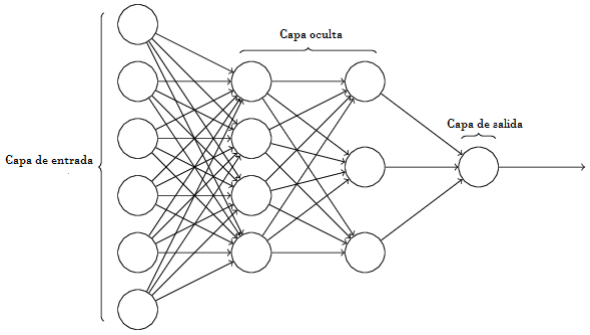

Figura 2.1: Estructura de una red neuronal [Nielsen, 2017]

En este ejemplo, la red está compuesta por neuronas interconectadas y organizadas en tres capas:

- La información proveniente de las fuentes externas es proporcionada a la red a través de la capa izquierda (denominada **capa de entrada**), las neuronas de esta capa se denominan neuronas de entrada.
- La **capa oculta**, es la parte central de la red neuronal y reciben datos de la capa de entrada, y proporcionan información procesada a la capa de salida. El número de niveles ocultos puede estar entre cero y un número elevado. Las neuronas de las capas ocultas pueden estar interconectadas de diferentes maneras, lo que determina las distintas topologías de redes neuronales.
- Finalmente, en la última capa se encuentra la **capa de salida**. Ésta es la responsable de dar los resultados finales de la red hacia el exterior.

Si analizamos la estructura de una neurona artificial (Figura 2.2), vemos que tiene asociadas unas entradas, unos pesos y unas funciones que utiliza para transformar los datos de entrada en los datos de salida (normalmente un único dato).

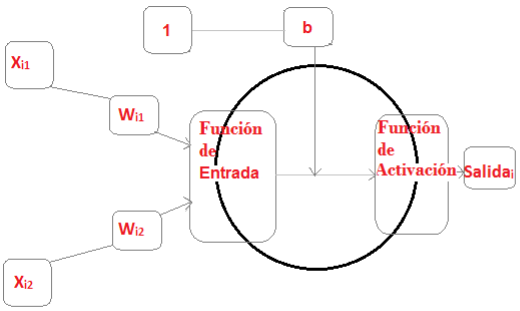

Figura 2.2: Ejemplo de una neurona con 2 entradas y 1 salida

2.1.2. Función de entrada

La neurona recibe diversos valores de entrada y los procesa mediante la expresión siguiente (producto interno entre las entradas y los pesos asociados a cada una de ellas):

$$input_i = (x_{i1} \cdot w_{i1}) + (x_{i2} \cdot w_{i2}) + ...(x_{in} \cdot w_{in})$$

Donde i es la neurona, x_i es el vector de entrada, w_i es el vector de pesos y n es el número de entradas a dicha neurona.

Los valores de entrada se multiplican por los pesos asociados a la neurona. De esta manera, los pesos permiten ajustar los valores de entrada en función de la relevancia que tenga cada característica para el problema que se desea resolver. Dicho de otro modo, los pesos es donde se almacena el 'conocimiento' de la red neuronal.

Normalmente, las neuronas tienen una entrada extra denominada *bias*. Este *bias* está formado por una entrada constante de magnitud 1 y un peso 'b'.

2.1.3. Función de Activación

La función activación determina el estado de actividad de una neurona; transforma la entrada global conocida como *net* en un valor (estado) de activación. El rango del valor puede estar acotado dependiendo de la función de activación empleada, por ejemplo en la sigmoidea [0,1], en la tangente hiperbólica [–1,1] o en la ReLU $[0,\infty)$.

Las funciones de activación más comunes son las siguientes, donde $z = net_i = input_i + b_i$:

1. Función Lineal:
$$f(z) = \begin{cases} -1 & z \leq \frac{-1}{a} \\ z * a & \frac{-1}{a} < z < \frac{1}{a} \\ 1 & z \geq \frac{1}{a} \end{cases}$$
2. Función Sigmoidea
$f(z) \equiv \frac{1}{1+e^{-z}}$
3. Función tangente hiperbólica
$f(z) \equiv \frac{e^{g*z} - e^{-g*z}}{e^{g*z} + e^{-g*z}}$
4. Función ReLU
$f(z) = max(0, z)$

2.2. Mecanismo de aprendizaje

Una red neuronal artificial tiene que aprender a calcular la salida correcta para cada entrada sobre un conjunto de datos de entrenamiento. A este proceso de aprendizaje se denomina: **proceso de entrenamiento**.

Se pueden identificar dos tipos de procesos de aprendizaje: supervisado y no supervisado.

Este proyecto se centra en el contexto del aprendizaje supervisado. En este tipo de aprendizaje se dispone de un conjunto de datos, de los cuales para unas características de entrada se conoce la salida, que permite entrenar la red de forma controlada. Esto posibilita que se pueda evaluar el rendimiento de la red comparando los resultados obtenidos de la red con los resultados ya clasificados.

Aunque no han sido objeto de estudio de este proyecto, existen redes neuronales artificiales que se aplican en el contexto de aprendizaje no supervisado (por ejemplo, clustering). En este tipo de aprendizaje no se dispone de una variable clase (etiqueta, salida deseada) asociada a cada patrón de entrenamiento.

2.2.1. Aprendizaje supervisado

El aprendizaje supervisado consiste en ajustar los pesos de las conexiones y *bias* de las neuronas de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red. Es decir, en función del error cometido en la salida.

En el proceso de aprendizaje se distinguen dos fases. La **fase de entrenamiento** y la **fase de testeo**, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de test utilizados en la correspondiente fase.

Según la metodología empleada los datos de entrenamiento se pueden dividir en dos, reservando un subconjunto de ellos como datos de validación que se utilizan para determinar los parámetros óptimos (o pseudo-óptimos) de la red neuronal.

Durante la fase de entrenamiento de una red neuronal artificial los pesos de las conexiones y los *bias* de las neuronas de la red sufren modificaciones. La expresión que describe la actualización de los pesos suele ser similar a la siguiente:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

o

Peso Nuevo = Peso Viejo + Cambio de Peso

Este proceso se repite de forma iterativa hasta que se cumpla alguno de los criterios de finalización establecidos.

Los criterios más comunes son los siguientes;

- Cuando los valores de los pesos y los *bias* se mantengan estables.
- Cuando se cumplen las iteraciones máximas de entrenamiento definidas anteriormente.
- Cuando la red sea capaz de clasificar correctamente más de un número determinado de casos de entrenamiento.

2.3. Testeo de la Red Neuronal

Después del proceso de entrenamiento los pesos de las conexiones de la red y los *bias* de las neuronas quedan fijos. El siguiente paso consiste en verificar si red neuronal es capaz de resolver de manera adecuada los problemas para los que ha sido entrenada.

Por lo tanto, se requiere de otro conjunto de datos diferente al utilizado en la fase de entrenamiento con el propósito de testear la red neuronal, denominado **conjunto de test**.

Cada ejemplo del conjunto de test contiene los valores de las variables de entrada y su correspondiente salida deseada. Luego se compara la solución calculada para cada ejemplo de test con la solución conocida.

2.4. Elección del conjunto inicial de pesos y *bias*

Al inicio del proceso de entrenamiento se debe determinar un estado inicial, esto es, seleccionar un conjunto inicial de pesos para las conexiones entre las neuronas y los *bias* de las neuronas de la red neuronal.

Esto puede realizarse por varios criterios:

- **Valor constante**

  Se inicializa los pesos y/o los *bias* de la red neuronal con un valor fijo.

- **Valor aleatorio**

  Este criterio se basa en asignar un peso a cada conexión de manera aleatoria.

  Existen dos casos típicos en la asignación de los pesos de la red:

  1. Distribución uniforme entre $[-n, n]$.
  2. Distribución Gaussiana con media 0.

Por lo general la inicialización de los *bias* no tiene tanta relevancia como la de los pesos. Por ello, es común inicializarlos a cero o con valores cercanos a cero.

Cabe mencionar que durante el transcurso del entrenamiento los pesos y los *bias* no se encuentran restringidos por la condición inicial.

2.5. Topologías Principales

2.5.1. Topología de las redes neuronales

La arquitectura de una red neuronal consiste en la organización y la distribución de las neuronas de la misma, formando diferentes tipos de capas o niveles.

En este sentido, los parámetros fundamentales de la topología de la red son: el número de capas, el número de neuronas por capa y el patrón de conectividad.

2.5.2. Redes Monocapa

Las redes monocapa, como expresa su nombre, son redes neuronales compuestas por una única capa. Este tipo de redes se utilizan para regenerar información de entrada que se presenta a la red de forma incompleta o distorsionada [Red, 2001].

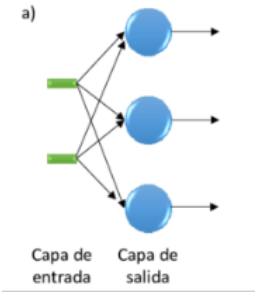

Figura 2.3: Red monocapa

2.5.3. Redes Multicapa

En las redes multicapas las neuronas están agrupadas en varios (2, 3, etc.) niveles o capas. Para distinguir la capa a la que pertenece una neurona hay que fijarse en el origen de las señales que reciben a la entrada y el destino de la señal de la salida.

Normalmente, todas la neuronas de una capa reciben señales de entrada desde otra capa anterior y envían señales de salida a una capa posterior.

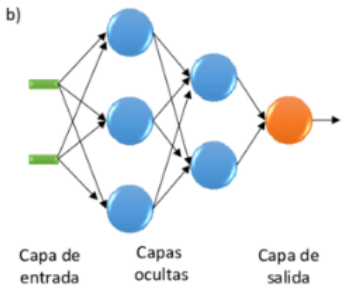

Figura 2.4: Red multicapa

2.5.4. Redes Convolucionales

Estas redes utilizan una arquitectura especial que está particularmente bien adaptada para clasificar imágenes. Son rápidas de entrenar y permiten generar redes profundas con muchas capas.

Las redes neuronales convolucionales se caracterizan por tres ideas básicas: los campos receptivos locales, los pesos compartidos y el agrupamiento de las capas.

- **Campos receptivos locales**

  En las redes que procesan imágenes es habitual que la entrada se represente como una matriz de dos dimensiones $n \times n$.

Figura 2.5: Capa de entrada en las redes convolucionales [Nielsen, 2017]

  Cada una de las neuronas de la capa oculta se conectan a una pequeña región de las neuronas de entrada denominada **el campo receptivo local de la neurona oculta**. Este campo es una pequeña ventana sobre los píxeles de entrada que es utilizada para aprender los pesos y *bias* asociados a dicha neurona. Se podría ver como una neurona oculta particular que aprende a analizar su campo receptivo (Figura 2.6).

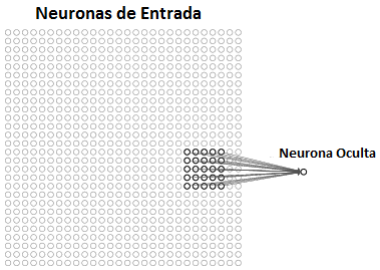

Figura 2.6: Conexión de la entrada con la neurona de la capa oculta en las redes convolucionales [Nielsen, 2017]

- **Pesos compartidos**

  Las conexiones de cada una de las neuronas ocultas a la capa de entrada comparten el mismo peso y *bias*. Al conjunto de neuronas ocultas se le denomina **mapa de características**, ya que permiten detectar características concretas de los datos de entrada.

  Para el reconocimiento de imágenes se necesita más de un mapa de características. De este modo, una capa de convolución completa se compone de varios mapas de características diferentes:

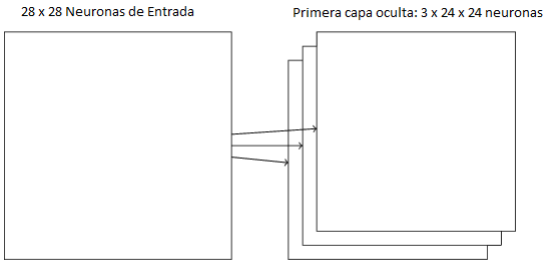

Figura 2.7: Estructura de la capa convolucional [Nielsen, 2017]

Una gran ventaja de compartir pesos y sesgos es que reduce en gran medida el número de parámetros involucrados en una red convolucional.

Como ejemplo, la imagen de entrada será una matriz 28×28 y el campo receptivo local de las neuronas ocultas de la primera capa de 5×5.

Para cada mapa de características se necesitarán $25 = 5 \times 5$ pesos compartidos, más un solo *bias* compartido. Así que cada mapa de características se requiere 26 parámetros. Si hay 20 mapas de características se tienen un total de $20 \times 26 = 520$ parámetros que definen la capa convolucional.

En comparación, si se define una capa completamente conectada de 30 neuronas ocultas, el número de parámetros asciende radicalmente. Si se tiene $784 = 28 \times 28$ datos de entrada, supone un total de 784×30 pesos más 30 *bias*, esto es, un total de 23.550 parámetros.

En otra palabras, la capa completamente conectada tendría aproximadamente 40 veces más parámetros que la capa convolucional.

Por esta misma razón, reduciendo el número de pesos a almacenar y calcular, se

reduce de manera significativa el tiempo de ejecución y permite generar redes de mayores dimensiones.

- **Capas de agrupamiento**

  Las capas de agrupamiento ("*pooling*") se usan generalmente inmediatamente después de capas convolucionales. Lo que hacen las capas de agrupación es simplificar la información en la salida de la capa convolucional, elevando el nivel de abstracción de la información aprendida.

  En detalle, una capa de *pooling* toma cada mapa de características de salida de la capa convolucional y prepara un mapa de características condensado.

  Por ejemplo, cada unidad en la capa de agrupación puede resumir la región de 2×2 neuronas en la capa anterior. La operación que más se utiliza para realizar la agrupación es la del valor máximo de la región y se conoce como *max-pooling*. La figura 2.8 ilustra la manera en la que se realiza el *max-pooling*:

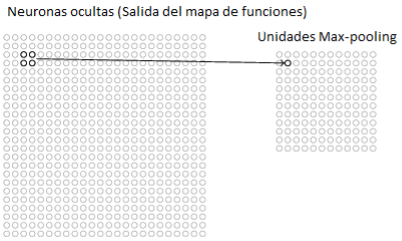

Figura 2.8: Max-pooling [Nielsen, 2017]

Una capa convolucional normalmente implica más de un solo mapa de características. Por lo que se aplica *max-pooling* a cada mapa de características por separado.

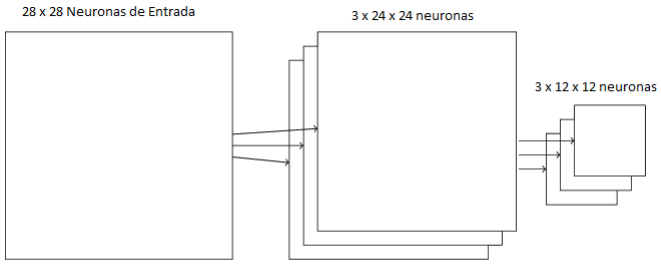

Figura 2.9: Ejemplo de una capa convolucional de una red CNN [Nielsen, 2017]

El *max-pooling* se puede describir como un método para que la red pregunte si la característica dada se encuentra en cualquier parte de una región de la imagen.

2.5.5. Capa *Softmax*

La idea del *softmax* es definir un nuevo tipo de capa de salida para las redes neuronales que se basa en la misma idea de una capa de neuronas totalmente conectadas. Sin embargo, se aplica la denominada función *softmax* para obtener las salidas, que se pueden interpretar como probabilidades.

Esto es muy útil en la clasificación, ya que da una medida de la probabilidad en las clasificaciones.

La función de activación softmax es:

$$a_j = \frac{e^{z}j}{\sum_k e^{z}k}$$

Donde, a_j es la activación de la neurona j, z_j es el valor calculado en la neurona j y el denominador es la suma de todos los valores calculados de las neuronas de salida.

2.6. Retropropagación

En 1986, *David Rumelhart*, *Geoffrey Hinton* y *Ronald Williams*, formalizaron un método ("*Backpropagation*") para que una red neuronal multicapa aprendiera la asociación que existe entre los patrones de entrada y las clases correspondientes [Reyes et al., 2016].

Este algoritmo consiste en un aprendizaje de un conjunto de pares de entradas-salida dados como ejemplo, empleando un ciclo de propagación-adaptación de dos fases:

1. Primera fase

   Se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas la capas superiores hasta generar una salida.

   Después, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida.

2. Segunda fase

   Estos errores se transmiten hacia atrás, partiendo desde la capa de salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total.

   Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada.

La importancia del algoritmo de retropropagación consiste en su capacidad de auto adaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe ente un conjunto de patrones de entrada y sus salidas correspondientes.

3. CAPÍTULO

Estado del arte

En este apartado se exponen diversos tipos de experimentos realizados mediante deep learning en diferentes ámbitos, los últimos avances en las redes neuronales y herramientas existentes para la automatización para machine learning y deep learning.

3.1. Tipos de Experimentos

3.1.1. Reconocimiento de objetos

La detección de objetos se refiere a la capacidad de los sistemas informáticos y de software para ubicar objetos en una imagen / escena e identificar cada objeto. La detección de objetos se ha utilizado ampliamente para la detección de rostros, detección de vehículos, conteo de peatones, imágenes web, sistemas de seguridad y automóviles sin conductor/a.

El método de red convolucional basada en la región (RCNN [Girshick et al., 2014]) logra una excelente precisión de detección de objetos al utilizando una ConvNet profunda para clasificar las propuestas de objetos. Se realiza una búsqueda selectiva [Uijlings et al., 2013] para averiguar las regiones de interés, cubriendo las áreas que podrían ser un objeto combinando píxeles y texturas similares en varios cuadros rectangulares.

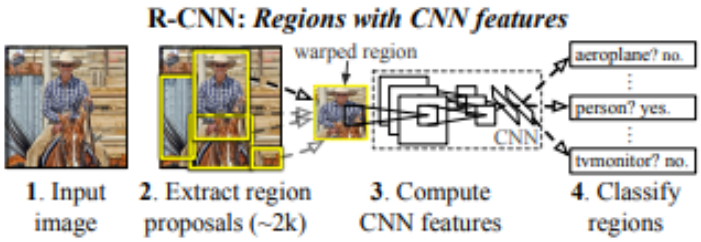

Figura 3.1: Visión general del sistema de detección de objetos

En el artículo [Girshick et al., 2014] se utilizan 2,000 áreas propuestas (cajas rectangulares) de búsqueda selectiva. Luego, estas 2,000 áreas se pasan a un modelo CNN pre-entrenado. Finalmente, las salidas (mapas de características) se pasan a una SVM para su clasificación. Se calcula la regresión entre los cuadros delimitadores predichos (bboxes) y los "ground-truth".

Fast R-CNN [Girshick, 2015] avanza un paso. En lugar de aplicar 2.000 veces la CNN a las áreas propuestas, solo pasa la imagen original a un modelo de CNN previamente entrenado una vez. El algoritmo de búsqueda selectiva se basa en el mapa de características de salida del paso anterior. Luego, la capa de agrupación de ROI se utiliza para garantizar el tamaño de salida estándar y predefinida. Estas salidas válidas se pasan a una capa totalmente conectada como entradas. Finalmente, se usan dos vectores de salida para predecir el objeto observado con un clasificador de softmax y adaptar las localizaciones del cuadro delimitador con un "linear regressor".

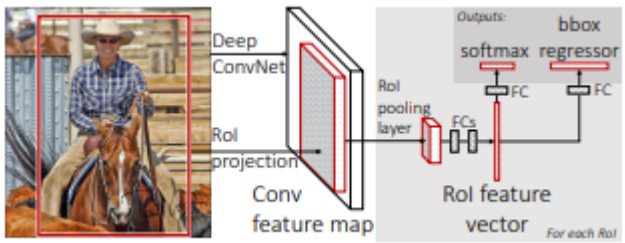

Figura 3.2: Arquitectura Fast R-CNN

Un Faster R-CNN [Ren et al., 2015] avanza más que el R-CNN rápido. El proceso de búsqueda selectiva es reemplazado por una *Region Proposal Network* (RPN), siendo este una red para proponer regiones. Después, se conecta a una capa convolucional con filtros 3x3, un relleno y 512 canales de salida. La salida está conectada a dos capas convolucionales 1x1 para clasificación y "box-regresión".

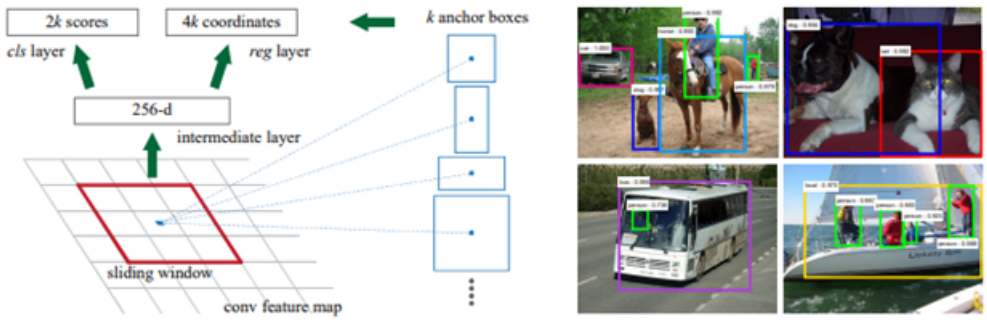

Figura 3.3: Izquierda: Region Proposal Network (RPN), Derecha: Ejemplo de detecciones utilizando propuestas RPN

En la segunda etapa del Faster R-CNN, similar al Fast R-CNN, se utiliza la agrupación ROI (región de interés) para las regiones propuestas. Seguidamente, se "aplana" mediante capas totalmente conectadas y finalmente se utiliza una función softmax para la clasificación y la regresión lineal para corregir la ubicación de los boxes.

3.1.2. NLP

El procesamiento del lenguaje natural (NLP) es una de las tecnologías más importantes de la era de la información. Comprender expresiones lingüísticas complejas es también una parte crucial de la inteligencia artificial. Las aplicaciones del NLP están en todas partes porque las personas comunican casi todo en lenguaje: búsqueda web, publicidad, correos electrónicos, servicio al/a la cliente, traducción de idiomas, informes de radiología, etc. Hay una gran variedad de tareas subyacentes y modelos de aprendizaje automático que funcionan con aplicaciones de NLP.

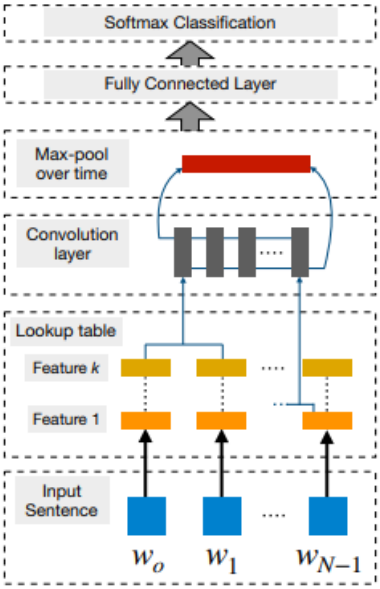

Figura 3.4: Una arquitectura cnn utilizada para nlp

El uso de CNN para el modelado de oraciones se remonta a Collobert y Weston [Collobert and Weston, 2008]. En este trabajo se utilizó el aprendizaje multitarea para generar predicciones múltiples para tareas de NLP, como etiquetas POS [1], trozos, etiquetas de entidad nombrada, roles semánticos, semánticamente palabras similares y modelo lingüístico. Se utilizó una tabla de consulta para transformar cada palabra en un vector de dimensiones definidas por el/la usuari@.

Esto se puede considerar como un método primitivo de inserción de palabras cuyos pesos se aprendieron en el entrenamiento de la red. Este trabajo desencadenó una enorme popularización de las CNN entre los investigadores de la NLP. Dado que las CNN ya habían demostrado su valía para las tareas de visión artificial, era más fácil para las personas creer en su desempeño.

Las CNN [Young et al., 2018] tienen la capacidad de extraer características sobresalientes de n-gramas de la oración de entrada para crear una semántica latente informativa representación de la oración para tareas posteriores.

Los RNN [Elman, 1990] utilizan la idea de procesar información secuencial. El término recurrente se aplica cuando realizan la misma tarea sobre cada instancia de la secuencia, de manera que la salida depende de los cálculos y resultados anteriores. En general, el vector de tamaño fijo se produce para representar una secuencia alimentando tokens uno por uno

[1] Es el proceso de asignar (o etiquetar) a cada una de las palabras de un texto su categoría gramatical.

a una unidad recurrente. En cierto modo, los RNNs tienen “Memoria” sobre los cálculos anteriores y utiliza esta información en el procesamiento actual.

Esta plantilla es naturalmente adecuada para muchas tareas de NLP como el modelado de lenguaje, la traducción automática o el reconocimiento de voz.

Dado que una RNN realiza un procesamiento secuencial mediante el modelado de unidades en secuencia, tiene la capacidad de capturar la inherente naturaleza secuencial presente en el lenguaje, donde las unidades son caracteres, palabras o incluso oraciones. Las palabras en un idioma desarrollan su significado semántico basado en las palabras anteriores de la oración, un ejemplo simple sería la diferenciar entre perro y perrito caliente. Las RNN están hechas a medida para modelar tales dependencias de contexto en el lenguaje y similares tareas de modelado de secuencias, lo que resultó ser una fuerte motivación para que los investigadores usen RNN en lugar de CNN en estas áreas.

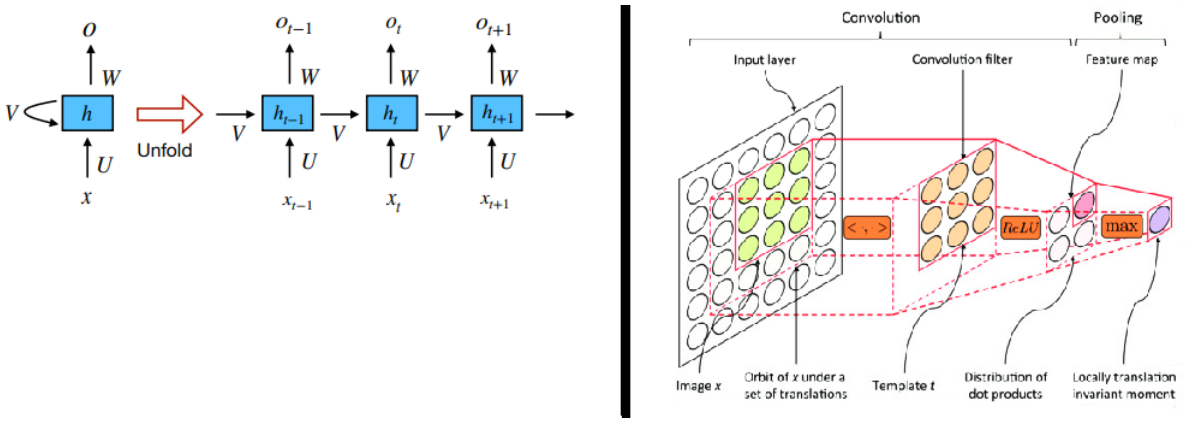

Figura 3.5: Comparativa de rnn vs cnn

3.1.3. Audio

Al igual que el dominio visual, se ha fomentado el progreso en el procesamiento de audio mediante redes neuronales profundas [[Lee et al., 2009], [Hinton et al., 2012a], [Deng et al., 2013], [Dai et al., 2017]], particularmente en el reconocimiento automático de voz (ASR) [[Rabiner et al., 1993], [Reddy, 1976]].

En el artículo *Interpreting and explaining deep neural networks for classification of audio signals* [Becker et al., 2018] se entrenan dos arquitecturas de redes neuronales las tareas de clasificación de audio, uno directamente en las formas de onda de audio sin procesar y la otra en espectrogramas de frecuencia de tiempo. Se usa la propagación de relevancia por

capas (LRP) [Bach et al., 2015] para investigar la relación entre los datos de entrada y la salida de la red y así, demostrar que la clasificación de género basada en espectrograma se basa principalmente en diferencias en rangos de frecuencia más bajos. Además, los modelos entrenados en forma de onda sin procesar se enfocan en una pequeña fracción de los datos de entrada.

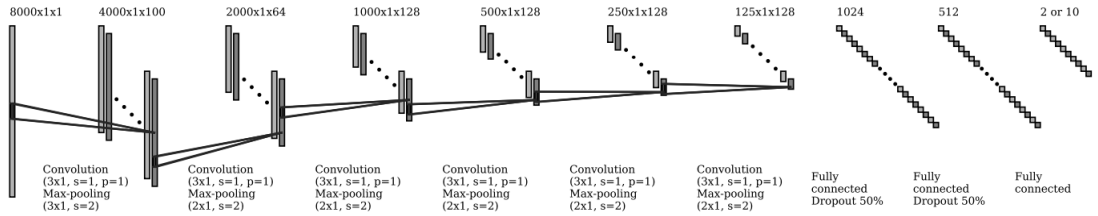

Figura 3.6: Arquitectura del modelo de AudioNet

En *Masked Conditional Neural Networks for Audio Classification* [Medhat et al., 2017] presenta la red neural condicional (CLNN) y su extensión la red neural condicional enmascarada (MCLNN). El CLNN está diseñado para explotar las propiedades de las señales temporales multidimensionales considerando la relación secuencial a través de los marcos temporales. La máscara en el MCLNN impone una sistemática en la dispersión que sigue un patrón en forma de banda de frecuencia.

Además, desempeña el papel de automatización en la exploración de una gama de combinaciones de características simultáneamente análoga a la búsqueda manual exhaustiva de las combinaciones de características hechas a mano.

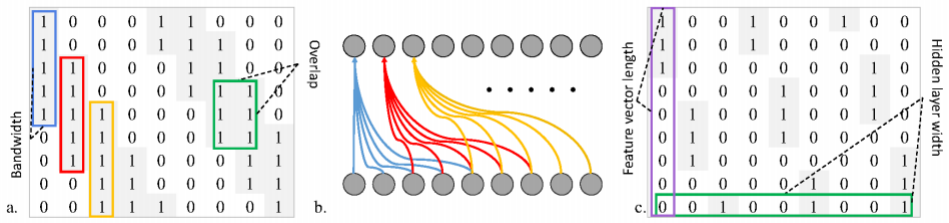

Figura 3.7: Patrones de enmascaramiento. a) Ancho de banda = 5 y Solapamiento = 3, b) los enlaces activos que siguen a los enlaces de a. c) Ancho de banda = 3 y Solapamiento = -1

La RLCN adapta a partir de la RBM Condicional [Taylor et al., 2007] los vínculos del nodo visible anterior y los nodos ocultos y se extiende a futuras tramas como en el ICRBM [Phone recognition using restricted boltzmann machines]. Adicionalmente, la RLCN adapta una operación de agrupamiento global [Lin et al., 2013], mejorando la precisión

de la clasificación [Bergstra et al., 2006]. La CLNNN permite que la relación secuencial a través de los marcos temporales de una señal multidimensional pueda ser considerada colectivamente al procesar una ventana de marcos.

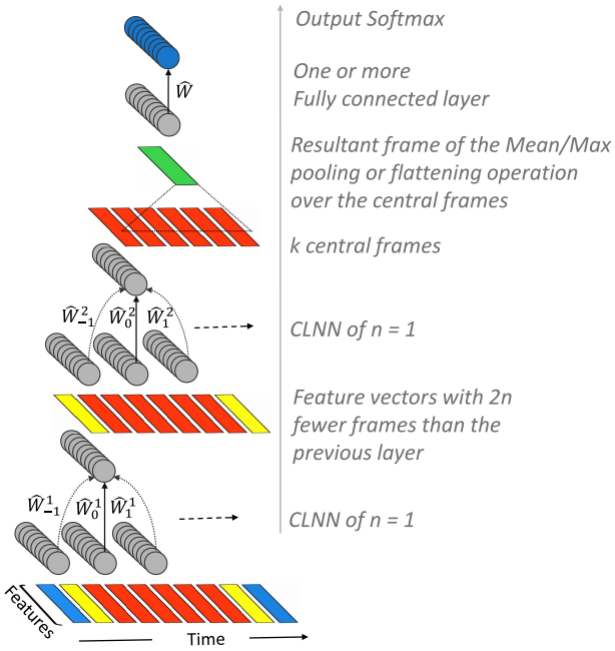

Figura 3.8: Dos capas de RLCNN con n = 1

3.1.4. Datasets tabulares

La tabla de datos es posiblemente la estructura de datos más antigua, es una forma de organizar los datos para su procesamiento por máquinas y a la vez de presentarlos visualmente para el consumo humano.

Para trabajar con este tipo de dato mediante redes neuronales ha sido muy común utilizar redes totalmente conectadas. Esto es, en una red de este tipo cada neurona de una capa está conectada a cada neurona de la capa anterior, y cada conexión tiene su propio peso. Este es un patrón de conexión de propósito totalmente general y no hace suposiciones sobre las características de los datos. También es muy costoso en términos de memoria (pesos) y cómputo (conexiones).

En el artículo *Using deep neural network with small dataset to predict material defects* [Feng et al., 2019] hacen uso de una red totalmente conectada pre-entrenada para

predecir defectos de solidificación mediante la regresión. Como resultado de la investigación resulta que el modelo de red neuronal profunda mostraba un mejor rendimiento de generalización que una red neuronal superficial y que un *support vector machine* [Suykens and Vandewalle, 1999].

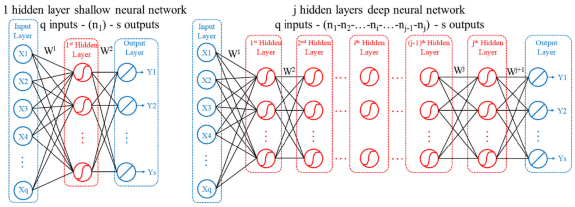

Figura 3.9: Diagramas esquemáticos de una red neuronal totalmente conectada poco profunda de 1 capa oculta y de j capas ocultas

Otro enfoque muy popular para trabajar con los dataset tabulares son los modelos basados en árboles. En *Deep Neural Decision Trees* [Yang et al., 2018] presentan modelos de árboles realizados por redes neuronales, *Deep Neural Decision Trees (DNDT)*. Una DNDT es intrínsicamente interpretable como un árbol, pero como también es una red neuronal (NN), se puede implementar fácilmente en kits de herramientas y entrenarse con el optimizador deseado.

Llegan a la conclusión de que las redes neuronales DNDT tienen mejor rendimiento que las NN para ciertas conjuntos de datos tabulares, al tiempo que proporcionan un árbol de decisión interpretable.

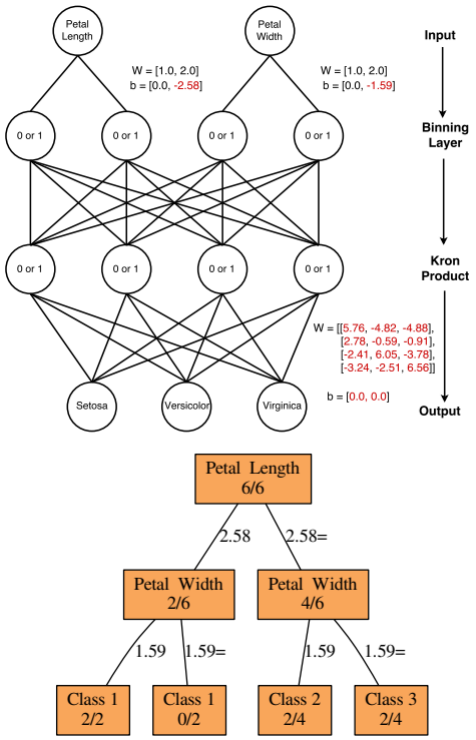

Figura 3.10: Una DNDT para el conjunto de datos de Iris. Arriba: vista DNDT donde las fuentes rojas indican variables entrenables y el negro indica constantes. Abajo: vista de árbol de decisión

Finalmente, en *Regularization Learning Networks: Deep Learning for Tabular Datasets* [Shavitt and Segal, 2018] presentan el *Counterfactual Loss* (LCF) y redes de aprendizaje de regularización (RLN) que utilizan la pérdida contrafactual para ajustar sus hiperparámetros de regularización efectivamente durante el aprendizaje junto con el aprendizaje de los pesos de la red.

Prueban el método en la tarea de predecir rasgos humanos a partir de covariables y datos de microbioma y muestran que los RLN mejoran significativamente y sustancialmente el rendimiento sobre los DNN clásicos.

3.1.5. Clasificación de imágenes

La clasificación de imágenes, que se puede definir como la tarea de categorizar imágenes de una de varias clases predefinidas, es un problema fundamental en visión por computador. Esta tarea constituye la base para otras tareas de visión artificial como localización, detección y segmentación [Karpathy et al., 2016].

Esta tarea es difícil para los sistemas automatizados dado que existen varias complicaciones como son la variabilidad del objeto dependiente del punto de vista o la variabilidad en clase al tener muchos tipos de objetos [Ciresan et al., 2011].

Tradicionamelmente, se utilizaba un enfoque de doble etapa para resolver el problema de clasificación, primero extrayendo descriptores de características de las imágenes y después, insertarlas como entrada para el clasificador. El mayor problema que tenía este enfoque fue que la precisión de la tarea de clasificación dependía del diseño de la etapa de la extracción de características, resultando ser una tarea formidable [LeCun et al., 1998].

En los últimos años, se ha demostrado que los modelos de aprendizaje profundo de múltiples capas de procesamiento de información no lineal son capaces de superar retos como la extracción y tansformación de características, así como el análisis y clasificación de patrones.

Entre ellos, CNNs ([LeCun et al., 1990]; [LeCun et al., 1989]) se han convertido en la arquitectura líder para la mayoría de las tareas de reconocimiento, clasificación y detección de imágenes [LeCun et al., 2015].

Aunque, inicialmente tuvieron éxito, las CNN profundas (DCNN) se llevaron la atención pública debido a que fueron alimentados por la GPU, conjuntos de datos más grandes y mejores algoritmos ([Krizhevsky et al., 2012]; [Deng et al., 2014]; [Simonyan and Zisserman, 2014]; [Zeiler and Fergus, 2014]).

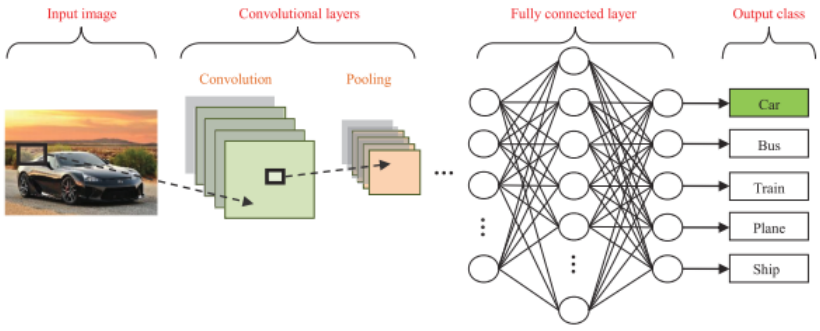

Figura 3.11: Ejemplo de una red convolucional

Varios avances, como la primera implementación de GPU [Chellapilla et al., 2006] y la primera aplicación de *max pooling* para DCNN [Huang et al., 2007] contribuyeron a su reciente popularidad. Pero, el avance más significativo que ha captado el interés en los DCNN para la clasificación de imágenes, fue en 2012 en el *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [Russakovsky et al., 2015]. Donde, Krizhevsky y los demás

utilizaron un DCNN para clasificar aproximadamente 1.2 millones de imágenes en 1000 clases, obteniendo resultados récord. Desde entonces, los DCNN se han denominado versiones posteriores del ILSVRC ([Simonyan and Zisserman, 2014]; [Zeiler and Fergus, 2014]; [Szegedy et al., 2015])

Además, en los últimos años se han implementado otros intentos de mejora relacionados con los diferentes aspectos de DCNN:

- Arquitectura de la red
  - *Network in network* [Lin et al., 2013]

    Se propone una nueva estructura que consiste en capas de *mlpconv* que utilizan perceptrones multicapa para convertir la entrada y una capa de agrupación promedio global como un reemplazo para las capas totalmente conectadas en las CNN convencionales.

    Las capas *mlpconv* modelan mejor los parches locales, y la agrupación promedio global actúa como un elemento estructural regularizador que evita el sobreajuste global.
  - *Multi-scale orderless pooling of deep convolutional activation features* [Gong et al., 2014]

    Para poder mejorar la invariancia de las activaciones de la CNN sin degradar su discriminación, se presenta un esquema denominado *multiscale orderless pooling* (MOP-CNN). Este esquema extrae las activaciones CNN para parches locales en múltiples niveles de escala, realizando VLAD (*Vectors of Locally Aggregated Descriptors* [Bergamo et al., 2013]) sin orden agrupando estas activaciones en cada nivel por separado, y concatenando el resultado.
- Funciones de activación no lineales
  - *Deep residual learning for image recognition* [He et al., 2016]

    Se presenta un marco de aprendizaje residual para facilitar el entrenamiento en las redes neuronales que son sustancialmente profundas. Se reformulan las capas como funciones residuales de aprendizaje con referencia a las entradas de capa, en lugar de aprender funciones sin referencia.
  - *Empirical evaluation of rectified activations in convolutional network* [Xu et al., 2015]

    Se investiga el rendimiento de diferentes tipos de funciones de activación rectificadas en redes neuronales convolucionales como son la unidad lineal

rectificada estándar (ReLU), unidad lineal rectificada con fugas (Leaky ReLU), paramétrica unidad lineal rectificada (PReLU) y nuevas unidades lineales rectificadas con fugas aleatorias (RReLU).

Los experimentos realizados sugieren que la incorporación de una pendiente distinta de cero para la parte negativa en unidades de activación rectificadas podría mejorar constantemente la resultados. Aunque, en conjuntos de datos de pequeña escala son propensos al sobreajuste.

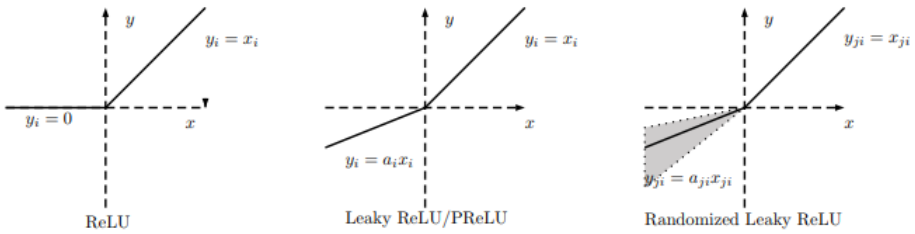

Figura 3.12: Variaciones de las activaciones ReLU

- Componentes de supervisión

  - *Deep learning using linear support vector machines* [Tang, 2013]

    En la mayoría de los modelos de aprendizaje profundo se emplea la función de activación softmax para predecir y minimizar la pérdida de entropía cruzada. En este trabajo, reemplazan la capa softmax con una máquina de vectores de soporte lineal. De esta manera el aprendizaje minimiza un margen basado en pérdida en lugar de la pérdida de entropía cruzada.

  - *Suppressing the unusual: towards robust cnns using symmetric activation functions* [Zhao and Griffin, 2016]

    Muchas redes neuronales convolucionales profundas realizan predicciones incorrectas sobre muestras adversas obtenidas por perturbaciones imperceptibles de muestras limpias. En este artículo se propone el uso de funciones de activación simétrica (SAF) en unidades transductoras de señales no lineales para convetir más robusta la red. Estas unidades suprimen señales de magnitud excepcional.

    Agregan a las CNN para mejorar su robustez y las entrenan fácilmente utilizando estrategias populares con la carga de entrenamiento moderada.

- Mecanismos de regularización

- *Improving neural networks by preventing co-adaptation of feature detectors* [Hinton et al., 2012b]

  Cuando una gran red neuronal de alimentación directa se entrena sobre un conjunto pequeño de entrenamiento, generalmente, se desempeña mal en los datos de prueba retenidos. Este "sobreajuste" es grandemente reducido al omitir al azar la mitad de los detectores de características en cada entrenamiento caso. También, evita complejas co-adaptaciones en las que un detector de características es solo útil en el contexto de varios otros detectores de características específicas.

  En su lugar, cada neurona aprende a detectar una característica que generalmente es útil para producir la respuesta correcta dada la gran variedad combinatoria de contextos internos en que debe operar. El *dropout* aleatorio da grandes mejoras en muchos tareas de evaluación y establece nuevos registros para el reconocimiento de voz y objetos.

- *Stochastic pooling for regularization of deep convolutional neural networks* [Zeiler and Fergus, 2013]

  Se utiliza método para regularizar redes reuronales convolucionales grandes, reemplazando las operaciones de agrupación deterministas convencionales con un procedimiento estocástico. Este selecciona aleatoriamente la activación dentro de cada región de agrupación mediante una distribución multinomial.

  El enfoque es libre de hiper-parámetros y puede ser combinado con otros enfoques de regularización, como la disminución de peso, el *dropout*, *data augmentation*, etc. para evitar el sobreajuste cuando se entrena en redes convolucionales profundas.

- Técnicas de optimización

  - *ImageNet Classification with Deep Convolutional Neural Networks* [Krizhevsky et al., 2012]

    Capacitan una red neuronal convolucional grande y profunda para clasificar los 1.2 millones imágenes de alta resolución en el concurso ImageNet LSVRC-2010 en 1000 clases diferentes.

    La red neuronal consiste de cinco capas convolucionales, de las cuales algunas son seguidas por capas de agrupación, tres capas totalmente conectadas y finalmente con un softmax. Para acelerar el entrenamiento, utilizan neuronas no saturantes y una implementación de GPU muy eficiente de la operación

de convolución. Para reducir el *overfitting* en las capas totalmente conectadas emplean el método de regularización muy efectivo denominado *dropout*.

Cuando se elimina una sola capa convolucional, el rendimiento de la red se degrada notoriamente, demostrando que la profundidad de la red es realmente importante para lograr buenos resultados.

3.2. Automatización del ciclo de vida del dato

Hoy en día en los foros educativos más populares de *Data Science* los términos de *Automated Machine Learning* y *Automated Deep Learning* están ocupando muchos titulares. Pero, la primera pregunta sería: "¿En realidad se puede automatizar todo el proceso de aprendizaje?"

Cuando un *Data Scientist* resuelve un problema, el flujo de trabajo estándar es el siguiente:

1. Recopilación de datos
2. Preprocesamiento de datos
3. Inicializar un modelo adecuado para el problema
4. Entrenar el modelo
5. Testear el modelo
6. Comunicar los resultados

El segundo paso, el preprocesamiento de datos es muy amplio, siendo una de las tareas más lentas (limpieza de datos, transformación de estos, etc.). Y las siguientes etapas se realizan para un único modelo, por lo que existe la posibilidad de que haya otro modelo mejor y más eficiente para el problema.

Como existe este problema, un requisito indispensable para crear un modelo automático, es que este debe de estar construido de numerosos modelos y compararlos en rendimiento como en precisión para el problema.

3.2.1. Herramientas para auto machine learning

Hay muchas bibliotecas disponibles para automatizar el aprendizaje automático, todas ellas intentan lograr más o menos el mismo objetivo, el de automatizar el proceso de aprendizaje automático. A continuación se presentan algunas de las bibliotecas de Python más utilizadas:

- Auto-Sklearn [Feurer et al., 2015]

  Es una herramienta de Python de código abierto que determina automáticamente las líneas de aprendizaje automático efectivas para los conjuntos de datos de clasificación y regresión. Está construido alrededor de la exitosa biblioteca scikit-learn.

- TPOT [Olson et al., 2016]

  Es una herramienta de automatización de *data science* de Python de código abierto, que funciona mediante la optimización de una serie de preprocesos de características y modelos, con el fin de maximizar la precisión de validación cruzada en conjuntos de datos.

- H2O [H2O.ai, 2016]

  H2O es una plataforma de código abierto, en memoria, distribuida, rápida y escalable de aprendizaje automático y analítica predictiva que le permite construir modelos de aprendizaje automático en big data y proporciona una fácil producción de dichos modelos en un entorno empresarial.

  El código central de H2O está escrito en Java y es capaz de adivinar el esquema del conjunto de datos entrantes y admite la ingesta de datos de múltiples fuentes en varios formatos.

  Dada la facilidad de uso y la implementación del modelo para los diversos algoritmos supervisados y no supervisados (Deep Learning, Tree Ensembles y GLRM) hacen del H2O una API muy buscada para la ciencia de big data data.

- Google's AutoML [Google, 2019]

  Contiene modelos exitosos de aprendizaje profundo para muchas aplicaciones, desde reconocimiento de imágenes hasta reconocimeinto de voz y taducción automática. El diseño manual de los modelos de aprendizaje automático es difícil debido a que el espacio de búsqueda de todos los modelos posibles puede ser combinadamente grande. Por ellos han explorando formas de automatizar el diseño de modelos

para el aprendizaje automático y entre muchos algoritomos han seleccionado los algoritomos evolutivos [Real et al., 2017] y los algoritmos de aprendizaje de refuerzo [Zoph and Le, 2016].

3.2.2. Herramientas para auto deep learning

Así como para el auto machine learning existen mucha variedad de software y librerías, para el auto deep learning no ocurre lo mismo, las herramientas para este entorno son escasas.

Por un lado, Auto-Keras [Jin et al., 2018] es una biblioteca de software de código abierto para el automatismo del aprendizaje automático (AutoML). Es desarrollado por DATA Lab en Texas AM University y colaboradores de la comunidad. El objetivo final de AutoML es proporcionar herramientas de aprendizaje profundo de fácil acceso para expert@s en dominios con conocimientos limitados de ciencia de datos o aprendizaje automático. Auto-Keras proporciona funciones para buscar automáticamente la arquitectura e hiperparámetros de los modelos de *deep learning* [Jin et al., 2018].

Por otro lado, la versión de Automatic Statistician [Statistician, 2019] es un sistema que explora un espacio abierto de posibles modelos estadísticos para descubrir una buena explicación de los datos, y luego produce un informe detallado con cifras y texto en lenguaje natural.

4. CAPÍTULO

Software y datasets

4.1. Keras

"Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano" [Wikipedia, 2019].

Está diseñada para posibilitar la experimentación en poco tiempo con redes de **Aprendizaje Profundo**. Sus caracteriza por su facilidad de uso y por ser modular y extensible.

Inicialmente fue desarrollada en el proyecto de investigación ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*) por **François Chollet**. Este ingeniero de Google fue su autor principal y ha sido el encargado de mantenerlo.

En 2017, el equipo de *TensorFlow* de Google decidió ofrecer soporte a Keras en la biblioteca de core de *TensorFlow*.

Microsoft añadió un backend en *CNTK* a Keras disponible desde la CNTK v2.0.

Keras ofrece un conjunto de abstracciones más intuitivas y de alto nivel haciendo más sencillo el desarrollo de modelos de aprendizaje profundo independientemente del *backend* computacional utilizado.

Contiene varias implementaciones de los bloques constructivos de las redes neuronales como por ejemplo los *layers*, funciones objetivo, funciones de activación, optimizadores matemáticos.

Además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales Convolucionales y para las Redes Neuronales Recurrentes.

También, es compatible con otras capas comunes de gran utilidad como el *dropout*, *batch normalizaction* and *pooling*.

Esta biblioteca permite generar modelos de *deep learning* en smartphones tanto sobre iOS como sobre Android, sobre una Java Virtual Machine o sobre web. También permite el uso de modelos de aprendizaje profundo en grupos de unidades de procesamiento de gráficos (GPU) y unidades de procesamiento de tensor (TPU).

En la Tabla 4.1 se muestran algunas peculiaridades de la biblioteca.

Software	**Keras**
Creador	François Chollet
Open Source	Sí
Plataforma	Linux, Mac OS X, Windows
Escrito en	Python
Interfaz	Python
Soporte de OpenMP	Sí
Soporte de Cuda	Sí
Redes Convolucionales	Sí

Tabla 4.1: Características de *Keras*

4.2. Base de datos

Para la realización del proyecto han sido necesarios varios datasets con diferentes características para poder validar el funcionamiento del módulo realizado. Dado que hoy en día las tareas de clasificación son más requeridas por los usuarios, se ha obtado por testear con 8 datasets de clasificación y 4 de regresión. Todas las bases de datos han sido obtendidas desde el repositorio web UCI [Dua and Graff, 2017].

4.2.1. Base de datos de clasificación

- Skin Segmentation Data Set

Este conjunto de datos está contruida de muestras recopiladas al azar que contiene valores RGB de las imágenes faciales de varios grupos de edad (jóvenes, adultos y tercera edad), grupos raciales (europeo, africano y asiático) y géneros obtenidos de la base de datos FERET [Phillips et al., 1998] y PAL [Minear and Park, 2004].

El tamaño total de la muestra es de 245057 entradas, de las cuales 50859 son de la clase "Skin" y 194198 son de la clase "No Skin".

Características	Univariado	**Nº instancias**	245057	**Area**	Computer
Tipo de variables	Real	**Nº variables**	4	**Fecha**	2012-07-17
Tipo de problema	Clasificación	**Missing Values?**	N/A	**Nº visitas web**	161652

Tabla 4.2: Resumen del dataset Skin Segmentation.

- Connect-4 Data Set

  Esta base de datos contiene todas las posiciones legales de 8 capas en el juego de connect-4 en el que ninguno de los jugadores ha ganado todavía, y en el que el siguiente movimiento no es forzado.

  Información del atributo: (x = jugador x ha tomado, o = jugador o ha tomado, b = en blanco)

  Los jugadores van tomando las diferentes posiciones del tablero, por ejemplo, 6a, 3d, etc. El tablero está estructurado de la siguiente manera:

	a	**b**	**c**	**d**	**e**	**f**	**g**
6	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.

Tabla 4.3: Tablero de Connect-4.

  La variable resultado es el valor teórico del juego para el primer jugador, donde puede tener los valores win,loss,draw.

Características	Multivariado	**Nº instancias**	67557	**Area**	Juegos
Tipo de variables	Categórico	**Nº variables**	42	**Fecha**	1995-02-04
Tipo de problema	Clasificación	**Missing Values?**	No	**Nº visitas web**	146967

Tabla 4.4: Resumen del dataset Connect-4.

- Adult Data Set

  La tarea de predicción es determinar si una persona gana más de 50 mil dólares al año. La extracción fue realizada por **Barry Becker** de la base de datos del Censo de 1994. Este conjunto de datos es una versión simplificada del dataset *"Census Income"* donde se tiene en cuenta el ingreso bruto ajustado.

Características	Multivariado	**Nº instancias**	48842	**Area**	Social
Tipo de variables	Categórico Entero	**Nº variables**	14	**Fecha**	1996-05-01
Tipo de problema	Clasificación	**Missing Values?**	Si	**Nº visitas web**	1549572

Tabla 4.5: Resumen del dataset Adult.

- Bank Marketing Data Set

  Los datos están relacionados con campañas de marketing de una institución bancaria portuguesa. Las campañas de marketing se basaron en llamadas telefónicas. El objetivo de clasificación es predecir si el cliente suscribirá (sí / no) un depósito a plazo (variable y).

Características	Multivariado	**Nº instancias**	45211	**Area**	Negocio
Tipo de variables	Real	**Nº variables**	17	**Fecha**	2012-02-14
Tipo de problema	Clasificación	**Missing Values?**	N/A	**Nº visitas web**	946370

Tabla 4.6: Resumen del dataset Bank Marketing.

- Default of credit card clients Data Set

  Esta investigación tuvo como objetivo el caso de los pagos predeterminados de los clientes en Taiwán y compara la precisión predictiva de la probabilidad de incumplimiento entre seis métodos de minería de datos.

  El resultado de la clasificación es binario: clientes creíbles o no creíbles, esto es, si el cliente realizó un pago predeterminado (Sí = 1, No = 0).

Características	Multivariado	**Nº instancias**	30000	**Area**	Negocio
Tipo de variables	Entero Real	**Nº variables**	24	**Fecha**	2016-01-26
Tipo de problema	Clasificación	**Missing Values?**	N/A	**Nº visitas web**	376361

Tabla 4.7: Resumen del dataset credit card clients.

- Diabetes 130-US hospitals for years 1999-2008 Data Set

  El conjunto de datos representa 10 años (1999-2008) de atención clínica en 130 hospitales de EE.UU. Incluye más de 50 características que representan los resultados del paciente y del hospital. Se extrajo información de la base de datos para encuentros que satisfacían los siguientes criterios.

  1. Es un encuentro hospitalario (un ingreso hospitalario).
  2. Es un encuentro diabético, es decir, uno durante el cual se ingresó al sistema cualquier tipo de diabetes como diagnóstico.
  3. La duración de la estadía fue de al menos 1 día y como máximo 14 días.
  4. Se realizaron pruebas de laboratorio durante el encuentro.
  5. Se administraron medicamentos durante el encuentro.

  Los datos contienen atributos tales como número de paciente, raza, sexo, edad, tipo de admisión, tiempo en el hospital, etc.

Características	Multivariado	**Nº instancias**	100000	**Area**	Vida
Tipo de variables	Entero	**Nº variables**	55	**Fecha**	2014-05-03
Tipo de problema	Clasificación Clustering	**Missing Values?**	Si	**Nº visitas web**	257277

Tabla 4.8: Resumen del dataset Diabetes.

- Census-Income (KDD) Data Set

  Este conjunto de datos contiene datos censales ponderados extraídos de las encuestas de población actual de 1994 y 1995 realizadas por la Oficina del Censo de los EE.UU. Los datos contienen 41 variables demográficas y relacionadas con el empleo.

  Una línea representa una instancia y los atributos están delimitador por comas. Hay 199523 instancias en el archivo de datos y 99762 en el archivo de test.

La tarea de predicción es determinar si una persona gana más de 50 mil dólares al año siendo el salario neto de la persona.

Características	Multivariado	**Nº instancias**	299285	**Area**	Social
Tipo de variables	Categórico Entero	**Nº variables**	40	**Fecha**	2000-03-07
Tipo de problema	Clasificación	**Missing Values?**	Si	**Nº visitas web**	152567

Tabla 4.9: Resumen del dataset Census-Income.

- Motion Capture Hand Postures Data Set

  Se usó un sistema de cámara de captura de movimiento Vicon para grabar a 12 usuarios que realizaban 5 posturas manuales con marcadores, unidos a un guante zurdo.

  Los datos presentados aquí ya están parcialmente preprocesados.

  Debido a la forma en que se capturaron los datos, es probable que para un registro y usuario determinados exista un registro casi duplicado que se origine en el mismo usuario.

  La variable respuesta varía de 1 a 5:

  1. Puño (con el pulgar hacia afuera)
  2. Stop (mano plana)
  3. Punto1 (apunte con el dedo índice)
  4. Punto2 (apunte con el dedo índice y medio)
  5. Agarre (dedos rizado como para agarrar)

Características	Multivariado	**Nº instancias**	78095	**Area**	Computer
Tipo de variables	Real	**Nº variables**	38	**Fecha**	2017-01-27
Tipo de problema	Clasificación Clustering	**Missing Values?**	Si	**Nº visitas web**	19759

Tabla 4.10: Resumen del dataset Hand Postures.

4.2.2. Base de datos de regresión

- Online News Popularity Data Set

  Este conjunto de datos resume un conjunto heterogéneo de características sobre artículos publicados por Mashable (www.mashable.com) en un período de dos años. El objetivo es predecir el número de acciones ("shares") en redes sociales (popularidad).

Características	Multivariado	**Nº instancias**	39797	**Area**	Negocio
Tipo de variables	Entero Real	**Nº variables**	61	**Fecha**	2015-05-31
Tipo de problema	Clasificación Regresión	**Missing Values?**	N/A	**Nº visitas web**	247368

Tabla 4.11: Resumen del dataset Online News.

- Online Video Characteristics and Transcoding Time Dataset Data Set

  El conjunto de datos contiene instancias de video muestreadas al azar que enumeran características fundamentales de video junto con la identificación de video de YouTube.

  El objetivo es predecir el tiempo de transcodificación del video.

Características	Multivariado	**Nº instancias**	168286	**Area**	Computer
Tipo de variables	Entero Real	**Nº variables**	11	**Fecha**	2015-05-19
Tipo de problema	Regresión	**Missing Values?**	N/A	**Nº visitas web**	36345

Tabla 4.12: Resumen del dataset *Online Video*.

- YearPredictionMSD Data Set

  El objetivo de este conjunto de datos es predecir el año de lanzamiento de una canción a partir de funciones de audio. Las canciones son en su mayoría canciones comerciales occidentales que van desde 1922 hasta 2011, con un pico en el año 2000.

  El dataset esta dividido en 463,715 entradas como train y 51,630 entradas como test. Contiene 90 atributos, siendo 12 datos de timbre promedio y 78 representando el timbre covarianza.

El primer valor es el año (objetivo), que va de 1922 a 2011.

Características	Multivariado	**Nº instancias**	515345	**Area**	N/A
Tipo de variables	Real	**Nº variables**	90	**Fecha**	2011-02-07
Tipo de problema	Regresión	**Missing Values?**	N/A	**Nº visitas web**	153139

Tabla 4.13: Resumen del dataset YearPredictionMSD .

- BlogFeedback Data Set

  Las instancias en este conjunto de datos contienen características extraídas de publicaciones de blog. La tarea asociada con los datos es predecir cuántos comentarios recibirá la publicación en las próximas 24 horas.

Características	Multivariado	**Nº instancias**	60021	**Area**	Social
Tipo de variables	Entero Real	**Nº variables**	281	**Fecha**	2014-05-29
Tipo de problema	Regresión	**Missing Values?**	N/A	**Nº visitas web**	82773

Tabla 4.14: Resumen del dataset BlogFeedback .

5. CAPÍTULO

Implementación

En este capítulo se describen las partes más relevantes de las implementaciones realizadas.

El código realizado se encuentra en un repositorio de *bitbucket*. Para mirar la implementación acceder a "*https://bit.ly/2lRBdCS*".

Para realizar las pruebas experimentales que se presentan en el capítulo 6 ha sido necesario realizar la implementación de cuatro módulos:

- Ingesta de los datos
- Transformación de las columnas al tipo de dato requerido
- Preprocesamiento de los datos
- Creación, entrenamiento y evaluación de la red neuronal

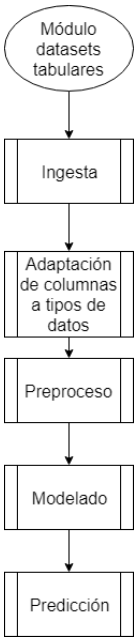

Figura 5.1: Diagrama de flujo del módulo de datasets tabulares

Estos módulos se han programado en *Python* ya que uno de los lenguajes de programación más usados para *Machine learning*. Se hace uso de librearías típicas como numpy, pandas, sklearn, imblearn o keras.

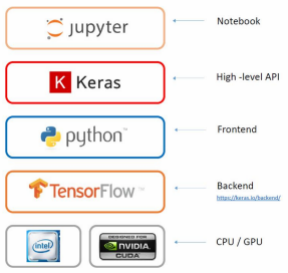

Figura 5.2: Esquema de trabajo

5.1. Ingesta de los datos

El funcionamiento de este módulo se muestra en el siguiente diagrama 5.3, donde primeramente, se detecta si el elemento recibido como parámetro corresponde a una conexión a base de datos o un fichero.

En el primer caso, se realiza la conexión y se cargan los datos desde la tabla deseada. En caso de la detección de un fichero, se observa el tipo mediante la extensión, si pertenece a las aceptadas por el programa (xlsx, sql, xml, json) se cargarán los datos en memoria.

Una vez cargados los datos se observa la cantidad de entradas del dataset, si este es mayor a un umbral seleccionado (4000 entradas) la ingesta terminará correctamente continuando con el siguiente proceso.

En el caso de que ocurra algún error en la carga, el fichero no corresponda a ninguna extensión permitida o no conste del mínimo de entradas, se producirá un error deteniendo la ejecución del programa y se mostrará un mensaje indicando el error.

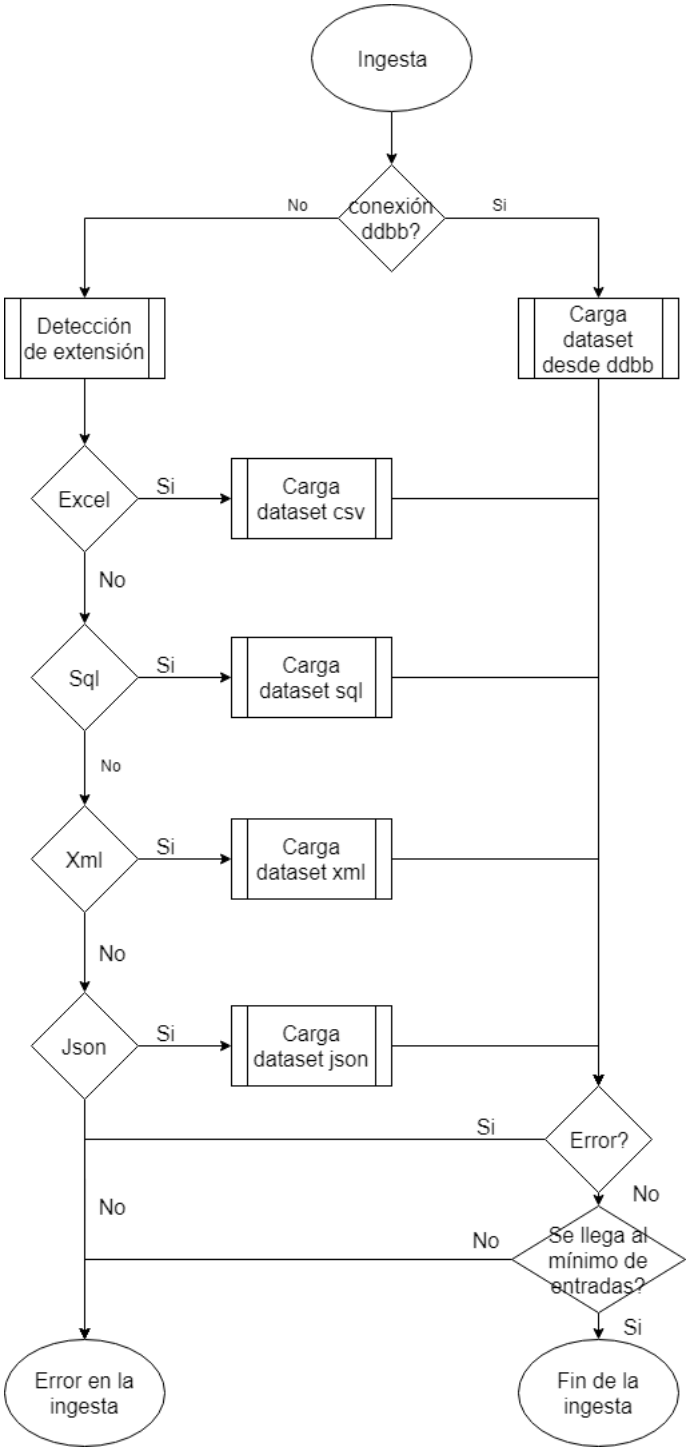

Figura 5.3: Diagrama de flujo de la ingesta de datasets tabulares

5.2. Adaptación de los tipo de datos

En este apartado se modifica el tipo de datos de cada columna del dataset, al finalizar este proceso las columnas será numéricas, categóricas o tipo objeto.

Este proceso tiene gran importancia dado que, una vez definido el tipo de la columna, se aplicarán diversos preprocesos dependiendo de la naturaleza de esta. Por esta razón, en base a la exploración de las características que describe la figura 5.4 se determinará el tipo de cada columna.

La secuencia de ejecución será la siguiente:

- Primeramente, si la columna contiene todos los valores únicos se determinará como tipo **objeto**.
- Si la columna inicialmente es detectada como numérica se examina la cantidad de únicos. Si este valor es inferior a 20 la columna se definirá como tipo **categórica**, en caso contrario como tipo **numérica**.
- Si inicialmente es detectado como objeto, se intenta realizar una conversión a numérico. Si la conversión es válida, como en el paso anterior, se observan la cantidad de únicos para definir si puede ser categórica o numérica.

  En cambio, si la conversión no es efectiva, se mira si la cantidad de únicos es menor a 150 convertiendo la columna a tipo **categórico**.

  Si no se cumple la condición, se intenta realizar una limpieza de valores “basura” y se intenta realizar la conversión a numérico.

  Finalmente, si no se ha detectado la naturaleza de la columna se comprueba si puede contener datos tipo fecha. En caso afirmativo la columna se convertirá a **numérica**, en caso contrario, será de tipo **objeto**.

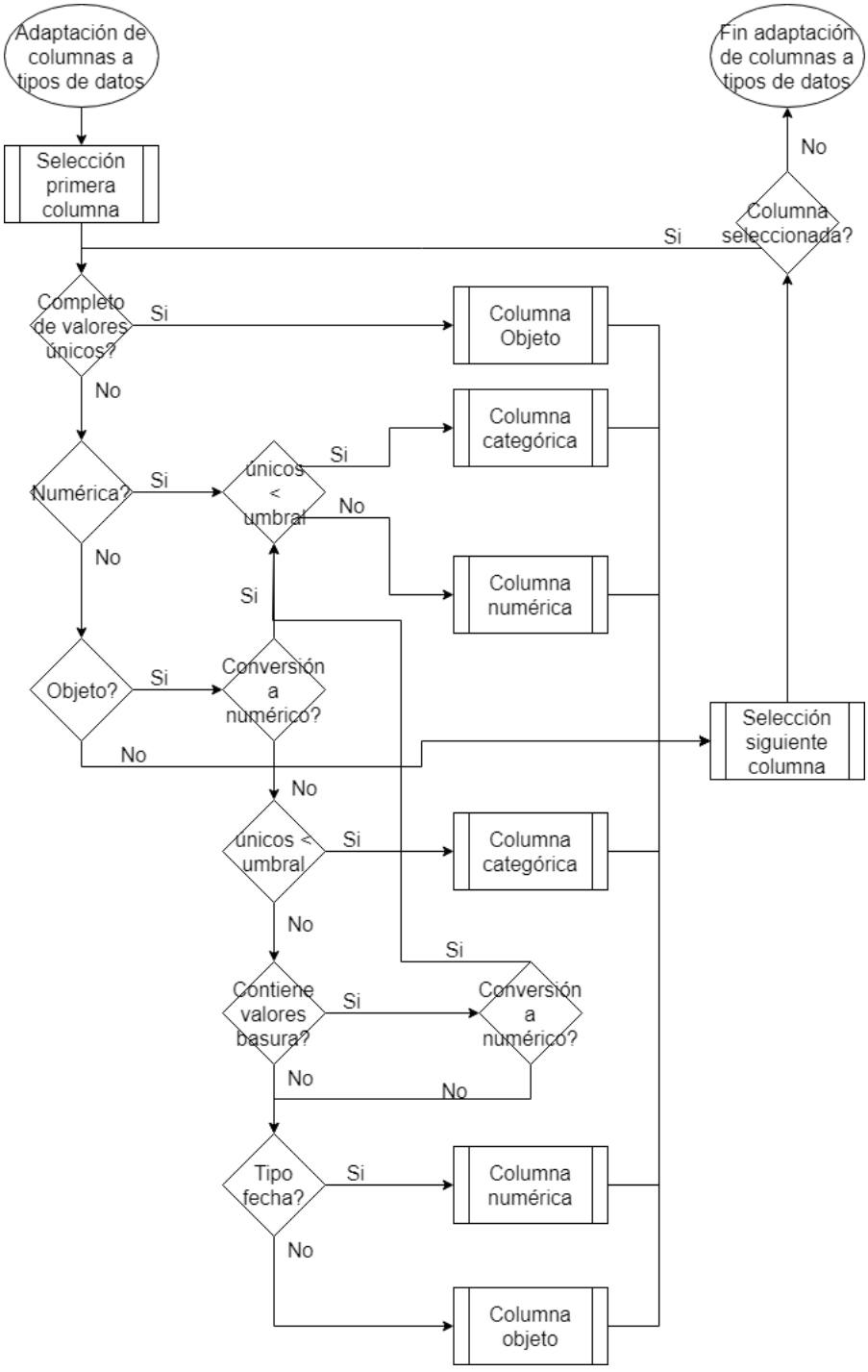

Figura 5.4: Diagrama de flujo de la adaptación de datos de datasets tabulares

5.3. Preprocesamiento de los datos

En esta fase se realiza un *data scrubbing* [1] y la tarea de transformaciones de cada columna logrando como resultado que todos los datos sean numéricos en el rango [0,1].

Inicialmente, se identifica la columna a predecir, si no se encuentra dicha columna o si es de tipo **objeto** se terminará la ejecución del programa y se mostrará un mensaje identificando el error.

Después, se eliminan las columnas que cumpla cualquiera de las siguientes condiciones:

- Columnas tipo objeto.
- Que superan el 30 % de NaNs.
- Columnas repetidas o altamente correlacionadas ($|cor| < 90\,\%$).
- Columnas irrelevantes, esto es, únicamente contienen un valor.

Si la columna a predecir no contiene datos, se terminará la ejecución del programa y se mostrará un mensaje identificando el error.

A continuación, se procede a eliminar las entradas duplicadas, que contengn NaN en la variable a predecir o que supere el 50 % de NaNs. Dado que se han eliminado en número de entradas, se revisa si se supera el umbral establecido (4000 entradas), deteniendo también, la ejecución del programa y mostrando el error en pantalla.

El siguiente proceso consiste en detectar los *outliers*, para ello se analiza la distribución de cada variable. Si la variable contiene una distribución normal, los *outliers* se eliminarán mediante el **método de desviación estándar**. Si la variable no sigue una distribución normal, los *outliers* se eliminarán mediante la técnica **IQR** o **rango intercuartílico**.

Siguiendo la misma táctica, se observan la cantidad de entrada revisando si se supera el umbral establecido (4000 entradas), deteniendo la ejecución del programa y mostrando el error en pantalla.

Una vez realizada la "limpieza" de los datos se procede a modificar y adaptar el dataset para que finalmente se transfieran a la red neuronal.

[1] También denominado *data cleansing*, es el proceso de corregir o eliminar datos en una base de datos que es incorrecta, incompleta, con formato incorrecto o duplicada.

Se comienza imputando todos los valores NaN del dataset columna por columna. Si esta es numérica se insertará la media y si es categórica se imputará la moda.

Posteriormente, se normalizan los datos numéricos acotando los valores entre 0 y 1. También, se adaptan los datos categóricos realizando un *encoding*. Si la columna contiene más de dos valores se realizará el *one hot encoding* y si contiene dos valores se efectuará el *ordinal encoding*.

Si no se ha recibido un dataset como test por parámetro se dividirá en train/test del único dataset recibido. La partición denominada como train constará del 70 % de los datos y el test con el 30 % restante seleccionados aleatoriamente.

En caso de que la problemática sea clasificar la partición tendrá unas tareas añadidas. Se eliminan las entradas de las clases que no consten de al menos 6 instancias, se dividen el 70 % de cada clase en el train y el resto como test. Una vez se conste de dos conjuntos de datos, se contempla si hay desbalanceo de datos en el train, en caso afirmativo, se balanceará mediante el método *SMOTE* y se finalizará esta etapa con éxito.

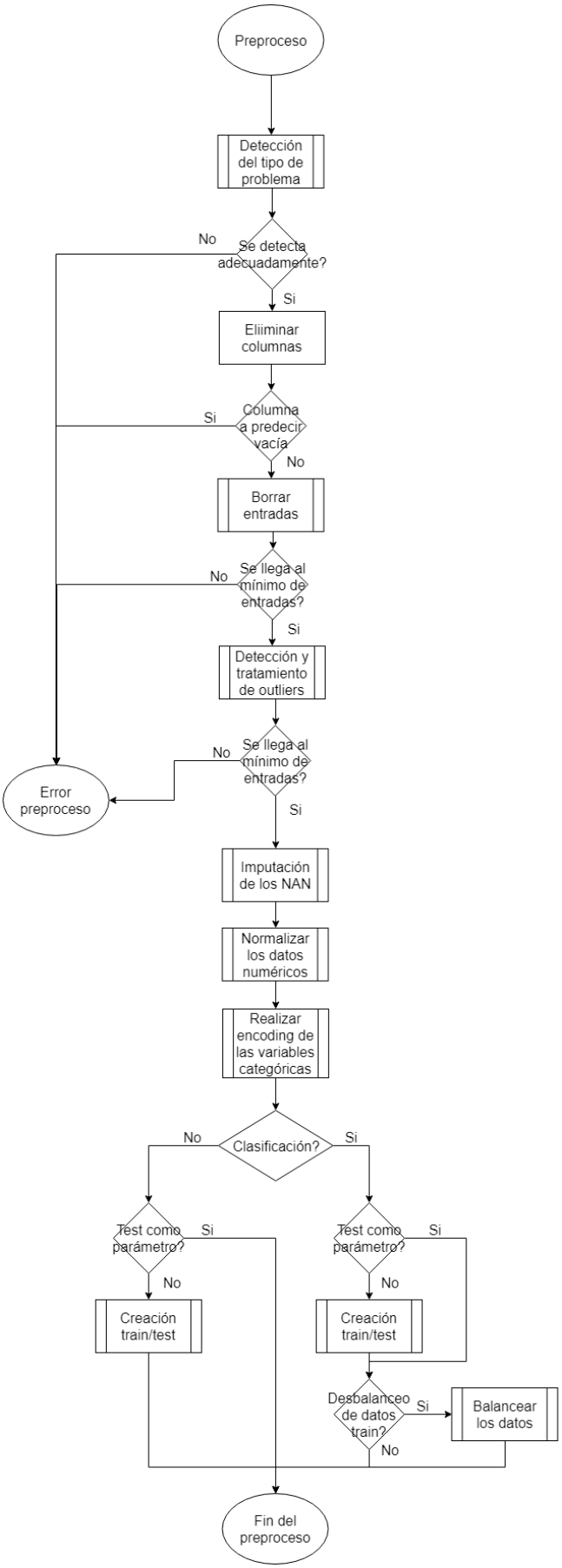

Figura 5.5: Diagrama de flujo del preproceso de datasets tabulares

5.4. Modelado y predicción

5.4.1. Primera fase

Debido a la vasta experiencia de Ibai en el ámbito del aprendizaje profundo y que yo previamente en mi trabajo de fin de grado elavoré redes neuronales para procesar datos, tabulares entre otros, basandose en reglas heurísticas se ha procedido ha seleccionar los hiperparámetros de la red.

En esta etapa inicial se crea una red neuronal totalmente conectada con unas dimensiones dinámicas basandose en el tamaño del dataset recibido. Los pesos de las conexiones de la red irán sujetas a una distribución normal con media 0.

Las características son las siguientes:

- El número de capas irá acotado a la siguiente fórmula: $round(log_{10}N - 1)$ donde N representa el número de entradas del dataset.
- El número de neuronas por capa irá sujeta a una función lineal entre la cantidad de neuronas de la primera de la red y el de la salida. La dimensión de la primera capa (Fl) se ciñe a la siguiente ecuación: $Fl = M * 2 + N * 0{,}02$ acotado entre $200 \leq Fl \leq 1000$, donde N representa el número de entradas y M el número de variables del dataset.
- La función de activación utilizada en la red es la función *ReLU*.
- Para los problemas de clasificación el algoritmo de aprendizaje será el *SGD* y para los problemas de regresión el optimizador *Adam*. En las dos casuisticas la tasa de aprendizaje será $lr = 10^{-3}$ y un *batch size* de 32.

  Este valor del *learning rate* has sido seleccionado debido a la siguiente proposición escrita en [Goodfellow et al., 2016] "Typically, a grid search involves picking values approximately on a logarithmic scale, e.g., a learning rate taken within the set {.1, .01, 10^{-3}, 10^{-4} , 10^{-5}}".
- Por un lado, el 30 % de los datos del conjunto train se utilizará como validación en la fase de entrenamiento. Por otro lado, se implementa un $early - stop$ de 10 épocas, de esta manera se reduce el tiempo de ejecución del programa evitando entrenar

una red neuronal que se encuentra en un estado de *overfitting* o que simplemente no mejora los resultados.

Este $early - stop$ actuará de la siguiente manera: El programa almacenará el mejor resultado de la validación época a época (el máximo *accuracy* o el mímimo *mean squared error*), si este valor no cambia en las siguientes 10 iteraciones se detiene el entrenamiento, devolviendo la red en el estado que se encontraba con el mejor resultado.

```
# Early stop para regresión
EarlyStopping(monitor='val_mean_squared_error', mode='min', verbose=0, patience=10)
# Early stop para clasificación
EarlyStopping(monitor='val_acc', mode='max', verbose=0, patience=10)
```

- La última capa de la red neuronal, la de salida, se construirá con diversas características en base al tipo de problema:
  - Para las tareas de regresión esta capa constará de una única neurona con la función de activación lineal.
  - Para las tareas de clasificación que se compone de más de dos clases, se utilizará una capa *softmax* con el número de neuronas equivalentes a las clases.
  - Para los problemas de clasificación binaria se creará una neurona con la función de activación sigmoidal.
- Del mismo modo que en la última capa, la función de coste varia en base al objetivo:
  - Para las tareas de regresión se empleará el *mean squared error*.
  - Para las tareas de clasificación que se compone de más de dos clases, se utilizará el *categorical cross-entropy*.
  - Para los problemas de clasificación binaria se basará en el *binary cross-entropy*.
- Para finalizar como regularizadores, se establecerá un *batch normalization* [Luo et al., 2018] y un *dropout* del 30 %.

Con el propósito de verificar de que el funcionamiento de la red neuronal es el óptimo se realizan unas pruebas preliminares con datasets como el conocido *Statlog (Shuttle)*.

Los resultados obtenidos para la red neuronal creada en esta primera fase se presentan en la sección 6.2.1.

5.4.2. Segunda fase

En este apartado, se realizan unos cambios de parámetros de la red neuronal definida en el apartado anterior con el proposito de mejorar los resultados obtenidos para cada base de datos en la parte 6.2.1.

Con esta finalidad en mente, la idea es aumentar el número de capas de la red, modificar la cantidad de neuronas por capa, variar la tasa de aprendizaje, excluir el *dropout* y desactivar el *batch normalization*.

De esta manera, se crean 107 combinaciones nuevas posibles con las siguientes propiedades:

- Ampliación del número de capas: $+0$, $+1$, $+2$.
- Alteración de la cantidad de neuronas por capa: $-20\,\%$, $+0\,\%$, $+20\,\%$.
- Variación de la tasa de aprendizaje: $*10$, $*1$, $*\frac{1}{10}$.
- *Dropout* $0\,\%$ o $30\,\%$.
- *Batch normalization* Si o no.

El número de combinaciones es elevado y la modificación de un hiperparámetro produce comportamiento diferente para distintos dataset. Por esta razón, se seleccionarán 34 combinaciones nuevas, realizando búsquedas aleatorias de los hiperparámetros de la red neuronal prensentadas, entrenado y evaluando cada una de ellas. El número de combinaciones es configurable para que el usuario pueda escoger en base al tiempo disponible como a los resultados deseados.

Finalmente, se seleccionará la red neuronal cuyo resultado sea el mejor, *accuracy* para la clasificación y *mean squared error* para la regresión.

Los resultados obtenidos de las combinaciones creadas en esta segunda fase se exponen en la sección 6.2.2.

5.5. Dificultades Surgidas

A lo largo de la implementación han ido apareciendo diversas complicaciones:

- La primera idea para detectar si la columna era de tipo fecha fue utilizar la librería *dateparser*. Aunque la identificación era muy precisa el uso de esta librería tenía un problema: Su tiempo de detección para cada entrada era muy elevado, haciéndola inviable para el programa. A cambio de la precisión, se ha decidido utilizar el módulo *datetime* de *python* agilizando el proceso de detección.

- Inicialmente, se quería utilizar el paquete *MICE* para realizar la imputación de los NaNs, pero por errores en la instalación del software dependiente se declinó este método.

  El siguiente paso fue utilizar un módulo experimental denominado *impute._iterative* de la librería *sklearn* disponible a partir de la versión 0.21 que realizaba la imputación multivariada de los NaN. Dado que no detectaba una vez instalado, con el fin de no perder más tiempo, se decidió utilizar el *SimpleImputer* de esta misma biblioteca.

6. CAPÍTULO

Experimentación

En este capítulo se presentan las pruebas realizadas y los resultados obtenidos en ellas.

La especificación de la máquina encargada de ejecutar los programas se describe en la sección 6.1.2 de este capítulo.

6.1. Descripción de las pruebas

El estudio se ha realizado sobre las base de datos descritas en el apartado 4.2, esto es, el software se evaluará con 8 datasets de clasificación y 4 de regresión.

6.1.1. Objetivo de las pruebas

La meta de la experimentación es observar el comportamiento del software desarrollado en su totalidad (preprocesamiento, modelado y predicción) frente a diversas bases de datos.

Para evaluar la calidad del software elaborado se ha optado por comparar con varias técnicas conocidas de *Machine learning* tradicional. Para los problemas de clasificación se usarán **Random Forest**, **KNN** y **Naive Bayes**; y para regresión **Random Forest**, **XGBoost** y **Linear regression**.

Todos los algoritmos utilizados son bien conocidos en el mundo del aprendizaje automático, salvo, tal vez, el **XGBoost**. Sin embargo, se incluye este algoritmo por los buenos resultados que está obteniendo en el ámbito de la regresión.

XGBoost [Chen and Guestrin, 2016] implementa algoritmos de aprendizaje automático en el marco *Gradient Boosting*. Esta librería proporciona un *tree boosting* paralelo (también conocido como GBDT, GBM) que resuelve muchos problemas de ciencia de datos de una manera rápida y precisa.

Cabe mencionar que existen otros métodos, como por ejemplo, el *Bayes error rate* o error óptimo siendo este el error de predicción más bajo posible que se puede lograr para los problemas de clasificación.

6.1.2. Especificación de la máquina

Esta computadora se ha encargado de realizar las ejecuciones sobre una GPU, concretamente, en una ***NVIDIA GeForce GTX TITAN Black***.

En las siguientes tablas se muestran las especificaciones de la GPU (Tabla 6.1), los detalles de su memoria (Tabla 6.2) y algunas características adicionales de la tarjeta gráfica (Tabla 6.3).

Núcleos CUDA	2880
Frecuencia de reloj normal	889 MHz
Frecuencia acelerada	980 MHz
Tasa de relleno de texturas	213 GigaTexels/s

Tabla 6.1: Especificaciones de la *NVIDIA GeForce GTX TITAN Black*

Frecuencia de la memoria (Gbps)	7,0
Cantidad de memoria	6144 MB
Interfaz de memoria	384-bit GDDR5
Ancho de banda máx.	336 GB/s

Tabla 6.2: Memoria de la *NVIDIA GeForce GTX TITAN Black*

Entorno de programación	*CUDA*
Temperatura máxima	95 ℃
Consumo	250 W
Requisitos mínimos de potencia	600 W

Tabla 6.3: Características adicionales de la *NVIDIA GeForce GTX TITAN Black*

Para más información sobre la GPU visitar la página oficial [NVIDIA, 2017].

6.2. Resultados obtenidos

6.2.1. Resultados de la primera fase

Las redes neuronales normalmente se ajustan a mano para cada problemática por un experto siendo esta una tarea incremental y muy costosa. Por lo que, el primer objetivo de esta primera es ver si la parametrización mediante reglas heurísticas definidas en 5.4.1 funciona de manera satisfactoria.

Para realizar la experimentación, se generan varias particiones independientes del dataset recibido como parámetro que cumple con la condición de $n^{\circ} entradas > 4000$.

Primeramente, se efectúa la división de 70 % de los datos como el conjunto *train* y el 30 % restante como *test*. A continuación, se ejecuta otra partición del subconjunto *train*, siendo el 80 % para realizar el entrenamiento y el 20 % para la fase de validación.

Por lo que, el 56 % del conjunto inicial servirá para el entrenamiento, el 14 % para validar en cada época y el 30 % para realizar el testeo final.

Cada experimento se ha repetido 3 veces adquiriendo en cada iteración los resultados finales y después, se ha obtenido la media para cada dataset.

El criterio de evaluación utilizado para las tareas de clasificación has sido la **tasa de acierto** y en los problemas de regresión el **error cuadrático medio**.

A continuación, se presentan gráficamente los resultados de la red neuronal definida inicialmente.

6.2.1.1. Clasificación

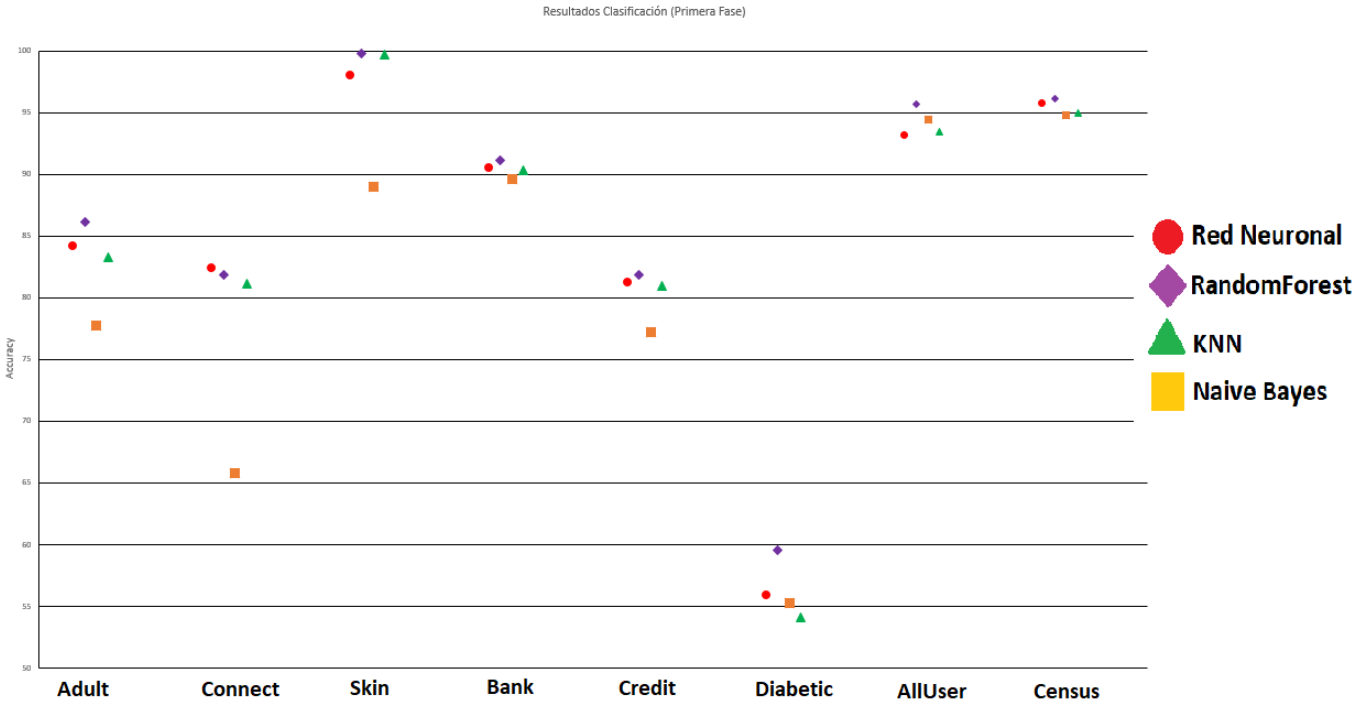

Figura 6.1: Gráfico de la experimentación en la primera fase (Clasificación)

Como se puede observar en la figura 6.1 la red neuronal diseñada en el apartado 5.4.1 alcanza diferentes niveles de rendimiento en las diversas bases de datos. Para las datasets ***Census***, ***Credit*** y ***Bank*** se consiguen muy buenos resultados, logrando el mejor rendimiento en ***Connect***.

En cambio, para los casos ***Adult***, ***Skin*** y ***Diabetic*** el *accuracy* alcanzado está algo alejado del algoritmo *RandomForest* (siendo el que logra los resultados *top*).

Para la base de datos ***AllUser*** aunque se haya obtenido una cifra elevada, comparando con los otros tres algoritmos, la red neuronal se posiciona en último lugar. Esto nos advierte de que ocurrirán casos en los que la red diseñada no sea eficiente y que se deben realizar cambios para conseguir mejores resultados.

Por último, hay que recordar que se ha generado una red genérica y no una exclusiva para el problema a resolver, por lo que, es interesante conseguir estos resultados con la primera versión.

6.2.1.2. Regresión

En los gráficos de los problemas de regresión el eje Y tiene un significado diferente a los de clasificación. Esto ocurre porque las mágnitudes del error cuadrático medio obtenidos en las pruebas son muy dispares entre las bases de datos. Por lo que, se ha optado por utilizar la diferencia relativa usando como referencia el resultado de la red neuronal.

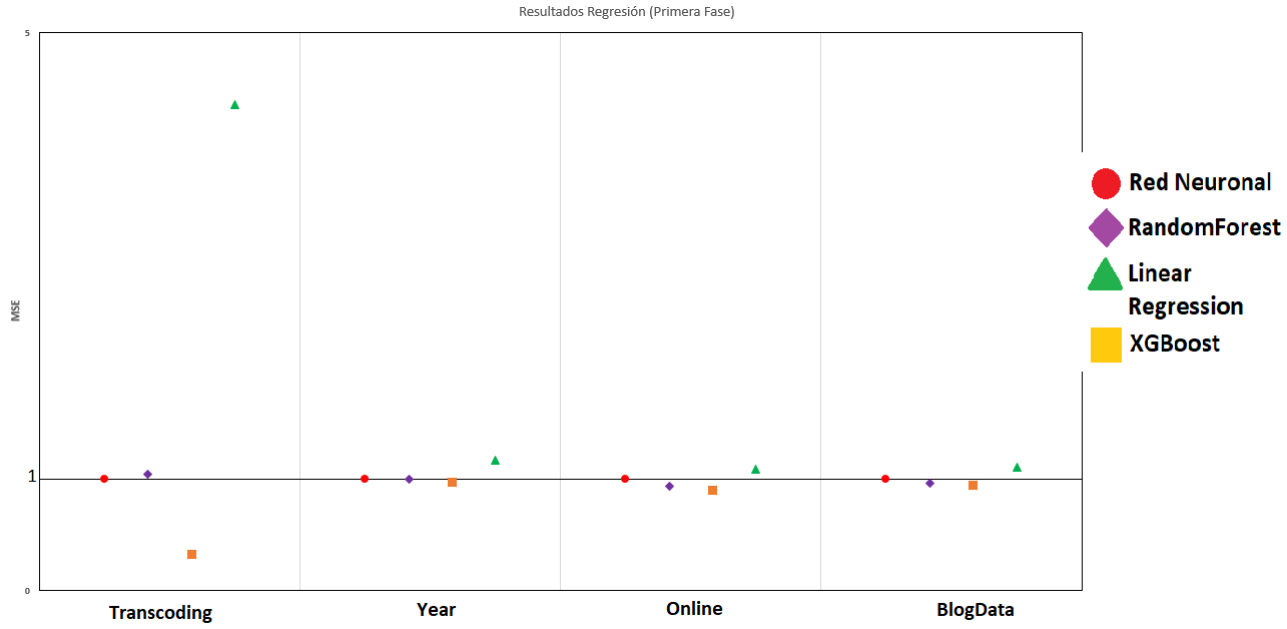

Figura 6.2: Gráfico de la experimentación en la primera fase (Regresión)

Como el resultado obtenido en la regresión lineal para el dataset ***Transcoding*** en la figura 6.2 está distante del valor unidad, no se aprecia adecuadamente los demás resultados. Por lo que, se generará un nuevo gráfico (6.3) acotando el eje vertical entre 0 y 2.

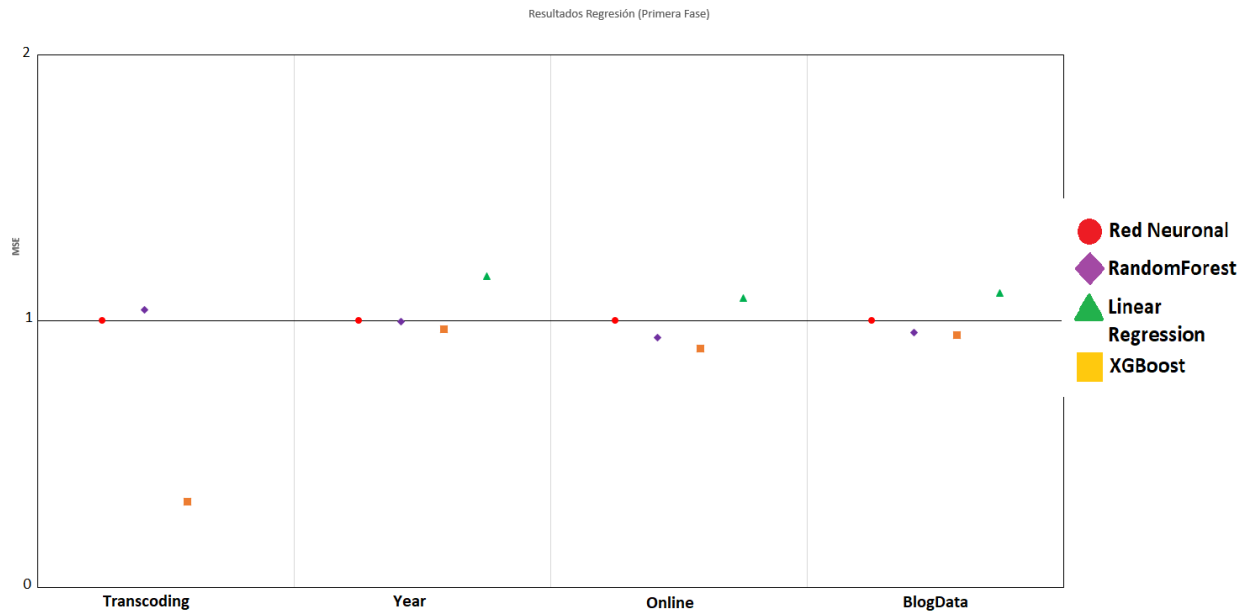

Figura 6.3: Gráfico de la experimentación en la primera fase en el rango $y = [0, 2]$ (Regresión)

Primeramente, hay que destacar el rendimiento del algoritmo **XGBoost** ya que es el mejor en todas la bases de datos, especialmente llama la atención el resultado extraordinario en ***Transcoding***. Esto nos demuestra lo eficiente que puede ser el algoritmo y justifica el uso en incremento de este método.

El caso opuesto se encuentra con la regresión lineal, en todos las pruebas realizadas es el que peor rendimiento obtiene. Esto viene sujeto a que es el método más simple entre los seleccionados.

En cuanto a los resultados que logra la red neuronal, se encuentra entre los algoritmos seleccionados, alcanzando unos rendimientos regulares para todos los casos. Esto indica que hay un margen de mejora e incita a realizar mejoras modificando los hiperparámetros de la red para alcanzar valores más eficientes.

6.2.2. Resultados de la segunda fase

En este apartado, del mismo modo que en la primera fase, se muestran los resultados de las diferentes composiciones formadas. Estas se han generado probando distintos parámetros para la red neuronal explicado en 5.4.2.

Se ha reproducido 3 veces cada experimentación adquiriendo el rendimiento y después, se

visualiza la media final para cada combinación.

6.2.2.1. Clasificación

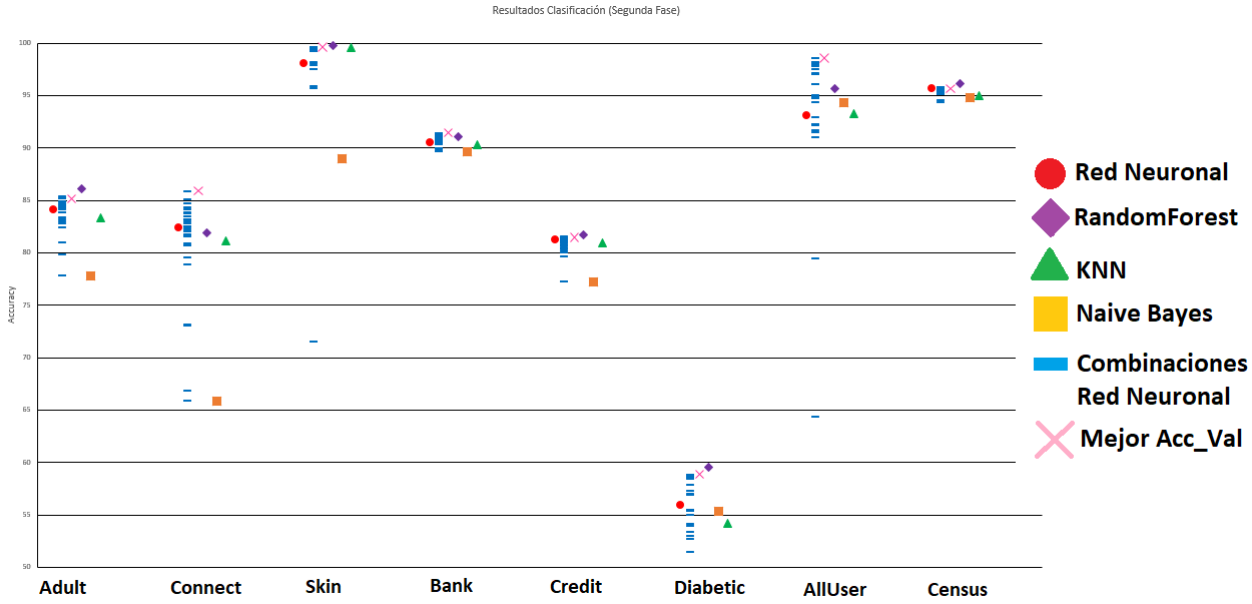

Figura 6.4: Gráfico de la experimentación en la segunda fase (Clasificación)

En la imagen 6.4 se han añadido los resultados de las nuevas 34 combinaciones aletorias defininas en 5.4.2 . Se puede percibir como la red neuronal del 5.4.1 en la mayoría de los datasets no es el más óptimo logrando establecerse en el centro de todos los resultados. Este suceso son buenas noticias, porque permiten mejorar y obtener un incremento en el rendimiento para cada base de datos.

Hay casos en los que la varianza es alta, por lo que esta segunda fase puede lograr mejoras importantes. Por ejemplo, en el dataset ***AllUser***, donde se conseguía el peor resultado, se logra alzanzar el *top* de rendimiento. También, para ***Connect***, ***Skin*** o ***Diabetic*** se mejora mucho llegando a valores comparables o mejores que *RandomForest*.

Dependiendo la base de datos con la que se ha trabaje la dispersión del *accuracy* de las combinaciones cambia. Para los datasets ***Credit*** y ***Census*** la varianza de los resultados es muy pequeña, pero en los demás no ocurre lo mismo. Este acontecimiento indica que existen situaciones donde el resultado decae generando *outliers* en el gráfico.

Si se analiza el la imagen 6.4 la combinación que ha logrado el mejor *accuracy* en la fase de validación, se sitúa en la parte superior de los resultados de *test* de cada bases de datos.

Ante esta situación, se puede deducir que se puede seleccionar una combinación adecuada a cada problema que vaya a obtener buenos resultados sin realizar la fase de testeo.

6.2.2.2. Regresión

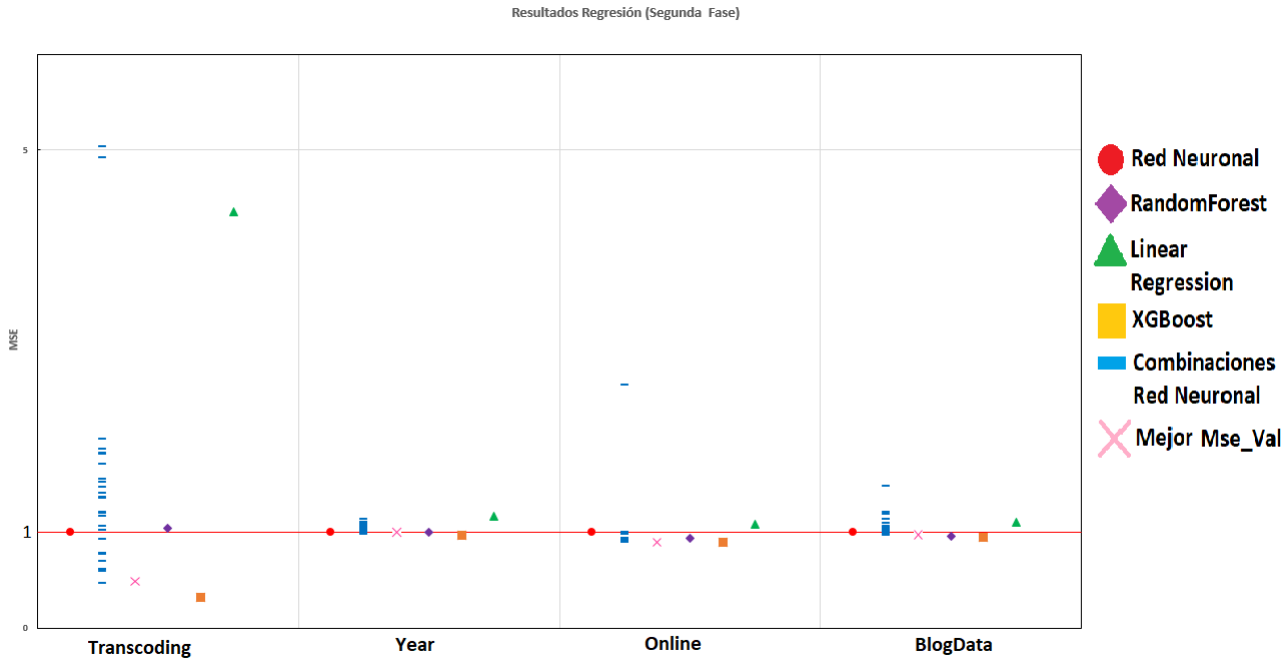

Figura 6.5: Gráfico de la experimentación en la segunda fase (Regresión)

Para poder visualizar de mejor las diferencias de resultados, se delimitará el rango del eje *y* como en el apartado anterior.

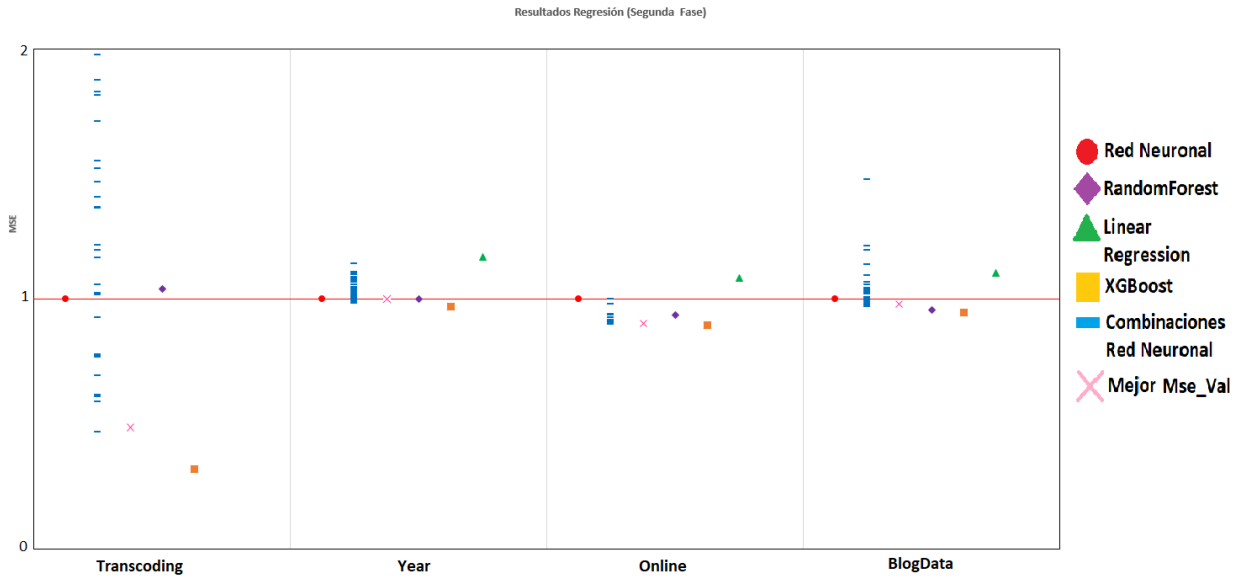

Figura 6.6: Gráfico de la experimentación en la segunda fase en el rango $y = [0,2]$ (Regresión)

Como en la clasificación se integran los resultados de las 34 combinaciones junto a los expuestos en 6.2. Al igual que en la fase anterior, el algoritmo **XGBoost** es el que obtiene mejor rendimiento.

Para el dataset ***Transcoding*** y ***Online*** se consiguen resultados muy variados, pero con los parámetros adecuados se logran valores casi tan buenos como los de **XGBoost**. También, en el caso de ***Transcoding***, esta variabilidad genera muchos *outliers* mostrando casos en los que los resultados han empeorado considerablemente.

Para las bases de datos restantes, los resultados de las diferentes combinaciones no consiguen mejorar mucho, aunque inicialmente no eran malos para la red neuronal.

Del mismo modo que en la clasificación, la combinación que consigue el menor *mse* de validación, se situa como mejor resultado en el testeo. Por lo que, el valor obtenido en la validación es un buen estimador para seleccionar una combinación final con un buen rendimiento.

7. CAPÍTULO

Conclusiones

Este capítulo presenta, a modo de cierre, un resumen del trabajo realizado junto a varias propuestas de mejora para futuros proyectos.

7.1. Valoración de los resultados

Se ha creado un módulo automático que consta de cuatro procesos. Inicialmente, se realiza la ingesta de los datos desde diferentes fuentes, tanto desde ficheros (csv, xml, json o sql) como desde una base de datos.

Una vez la carga se haya finalizado satisfactoriamente, se transforman las columnas al tipo de dato requerido. Después, se realiza el preprocesamiento de los datos, esto es, se eliminan los datos redundantes, se detectan y corrigen los valores faltantes y se transforma el formato de la información haciéndolo legible para la red neuronal.

Finalmente, se generarán y entrenarán una serie de redes neuronales. Primero, se construirá una red que constará con las características definidas en 5.4.1. Posteriormente, se realiza una búsqueda aleatoria de parámetros presentado en la sección 5.4.2 y se modifica la red inicial.

Cabe mencionar que la parametrización de una red neuronal es una tarea muy compleja que requiere mucha experiencia y conocimientos. Además, el problema se dificulta cuando se agrega el proceso de automatización de los hiperparámetros.

Aunque el problema a llevar a cabo es complejo, dado que se desea obtener buenos resultados para cada dataset a partir de una red neuronal automática, se obtienen unos resultados dignos para cada problemática. Por lo que, el módulo automatizado es una herramienta muy útil y competente.

7.2. Posibles mejoras

El software realizado en este proyecto ha satisfecho los objetivos propuestos al comienzo del mismo. Sin embargo, está abierto a otros trabajos que permitan complementar y justificar de manera más detallada los resultados obtenidos. Concretamente, las tareas que se podrían desarrollar son las siguientes:

- Mejorar las técnicas aplicadas en los obstáculos mencionados en la sección anterior 5.5, por ejemplo, se podría imputar los NaN en las columnas categóricas insertando los valores en base a sus apariciones en esta.
- Detectar diferentes niveles de desbalanceo de datos y aplicar en base a los niveles *RUS*, *SMOTE* o *RUS+SMOTE*.
- Implementar diferentes arquitecturas o parámetros internos de la red neuronal.

Bibliografía

[Red, 2001] (2001). Redes neuronales: Conceptos básicos y aplicaciones. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf. Accessed: 2017-07-14.

[Bach et al., 2015] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

[Becker et al., 2018] Becker, S., Ackermann, M., Lapuschkin, S., Müller, K.-R., and Samek, W. (2018). Interpreting and explaining deep neural networks for classification of audio signals. *arXiv preprint arXiv:1807.03418*.

[Bergamo et al., 2013] Bergamo, A., Sinha, S. N., and Torresani, L. (2013). Leveraging structure from motion to learn discriminative codebooks for scalable landmark classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 763–770.

[Bergstra et al., 2006] Bergstra, J., Casagrande, N., Erhan, D., Eck, D., and Kégl, B. (2006). Aggregate features and a da b oost for music classification. *Machine learning*, 65(2-3):473–484.

[Chellapilla et al., 2006] Chellapilla, K., Puri, S., and Simard, P. (2006). High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

[Ciresan et al., 2011] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[Dai et al., 2017] Dai, W., Dai, C., Qu, S., Li, J., and Das, S. (2017). Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE.

[Deng et al., 2013] Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE.

[Deng et al., 2014] Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[Feng et al., 2019] Feng, S., Zhou, H., and Dong, H. (2019). Using deep neural network with small dataset to predict material defects. *Materials & Design*, 162:300–310.

[Feurer et al., 2015] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Pro-*

ceedings of the IEEE conference on computer vision and pattern recognition, pages 580–587.

[Gong et al., 2014] Gong, Y., Wang, L., Guo, R., and Lazebnik, S. (2014). Multi-scale orderless pooling of deep convolutional activation features. In *European conference on computer vision*, pages 392–407. Springer.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Google, 2019] Google (2019). Google cloud automl. [Web; accessed 2019-05-01].

[H2O.ai, 2016] H2O.ai (2016). H2o. [Web; accessed 2019-05-01].

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton et al., 2012a] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29.

[Hinton et al., 2012b] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[Huang et al., 2007] Huang, F. J., Boureau, Y.-L., LeCun, Y., et al. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE.

[Jin et al., 2018] Jin, H., Song, Q., and Hu, X. (2018). Auto-keras: Efficient neural architecture search with network morphism.

[Karpathy et al., 2016] Karpathy, A. et al. (2016). Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2009] Lee, H., Pham, P., Largman, Y., and Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104.

[Lin et al., 2013] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

[Luo et al., 2018] Luo, P., Wang, X., Shao, W., and Peng, Z. (2018). Towards understanding regularization in batch normalization. *CoRR*, abs/1809.00846.

[Medhat et al., 2017] Medhat, F., Chesmore, D., and Robinson, J. (2017). Masked conditional neural networks for audio classification. In *International Conference on Artificial Neural Networks*, pages 349–358. Springer.

[Minear and Park, 2004] Minear, M. and Park, D. C. (2004). A lifespan database of adult facial stimuli. *Behavior Research Methods, Instruments, & Computers*, 36(4):630–633.

[Nielsen, 2017] Nielsen, M. (2017). Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/. (Acceso: 2017-07-04).

[NVIDIA, 2017] NVIDIA (2017). Tarjeta gráfica GeForce GTX TITAN Black. http://www.nvidia.es/object/geforce-gtx-titan-black-es.html. (Acceso: 2017-07-04).

[Olson et al., 2016] Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., and Moore, J. H. (2016). *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing.

[Phillips et al., 1998] Phillips, P. J., Wechsler, H., Huang, J., and Rauss, P. J. (1998). The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, 16(5):295–306.

[Rabiner et al., 1993] Rabiner, L. R., Juang, B.-H., and Rutledge, J. C. (1993). *Fundamentals of speech recognition*, volume 14. PTR Prentice Hall Englewood Cliffs.

[Real et al., 2017] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org.

[Reddy, 1976] Reddy, D. R. (1976). Speech recognition by machine: A review. *Proceedings of the IEEE*, 64(4):501–531.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

[Reyes et al., 2016] Reyes, M. A. V., Márquez, C. Y., and Fernández, L. P. S. (2016). Algoritmo backpropagation para redes neuronales: conceptos y aplicaciones.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

[Shavitt and Segal, 2018] Shavitt, I. and Segal, E. (2018). Regularization learning networks: deep learning for tabular datasets. In *Advances in Neural Information Processing Systems*, pages 1379–1389.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Statistician, 2019] Statistician, T. A. (2019). The automatic statistician. [Web; accessed 2019-05-01].

[Suykens and Vandewalle, 1999] Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Tang, 2013] Tang, Y. (2013). Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*.

[Taylor et al., 2007] Taylor, G. W., Hinton, G. E., and Roweis, S. T. (2007). Modeling human motion using binary latent variables. In *Advances in neural information processing systems*, pages 1345–1352.

[Uijlings et al., 2013] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2):154–171.

[Wikipedia, 2019] Wikipedia (2019). Keras — wikipedia, la enciclopedia libre. [Internet; descargado 23-agosto-2019].

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

[Yang et al., 2018] Yang, Y., Morillo, I. G., and Hospedales, T. M. (2018). Deep neural decision trees. *arXiv preprint arXiv:1806.06988*.

[Young et al., 2018] Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75.

[Zeiler and Fergus, 2013] Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[Zhao and Griffin, 2016] Zhao, Q. and Griffin, L. D. (2016). Suppressing the unusual: towards robust cnns using symmetric activation functions. *arXiv preprint arXiv:1603.05145*.

[Zoph and Le, 2016] Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.