UNIVERSIDAD NACIONAL DEL SUR

TESIS DE DOCTORA EN CIENCIAS DE LA COMPUTACIÓN

Redes neuronales profundas para la resolución de problemas de regresión con bases de datos tabulares pequeñas

Lucila Lourdes Chiarvetto Peralta

BAHÍA BLANCA ARGENTINA

2023

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado académico de Doctora en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Departamento de Ciencias e Ingeniería de la Computación durante el período comprendido entre el 1/03/2019 y el 30/04/2023, bajo la dirección de la Dra. Nélida Beatriz Brignole.

Lucila Lourdes Chiarvetto Peralta

UNIVERSIDAD NACIONAL DEL SUR

Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el 05 /12 /2023 mereciendo la calificación de .... (....8.(OCHO).....) DISTINGUIDO

Resumen

Las redes neuronales artificiales (ANN: Artificial Neural Networks) recientemente han tenido un gran impacto en la resolución de algunos problemas debido a que ha sido posible la implementación de algoritmos que requieren una gran capacidad de cómputo. Simultáneamente, enormes volúmenes de datos han sido dispuestos para el entrenamiento de estos modelos. Aunque la capacidad de cómputo ya no supone una restricción, la disponibilidad de volúmenes de datos puede ser un problema en algunos contextos en los cuales no siempre es factible obtener nuevos.

En esta tesis se analizan modelos de ANN profundas con bases de datos tabulares pequeñas circunscritos a problemas de regresión. Una tarea de regresión se define como el mapeo de un valor numérico en referencia a una entrada también numérica. Se espera que cumpla con las siguientes características: carencia de sesgo, consistencia y eficiencia. Además, se ha propuesto una definición de base de datos pequeña para ANN a fines prácticos: la cantidad de parámetros de una ANN es una medida de su complejidad. El balance entre los parámetros y la cardinalidad del conjunto de observaciones indica que nos encontramos ante un problema de base de datos pequeña. Asimismo, se ha considerado que esta es una definición necesaria aunque perfectible.

El estudio del la división del conjunto de datos en la evaluación del modelo se ha enfocado en las situaciones en las que se emplea un modelo de regresión (MR) construido con una ANN. Como contribución, se ha estudiado si el control del la división del conjunto de datos produce una estimación más precisa del error de generalización teniendo en consideración el mismo punto inicial. Con este propósito, se presenta un algoritmo evolutivo original que pueda controlar la división de los datos en función de una distancia. Otro de los aportes originales es la introducción de un algoritmo híbrido de optimización de hiperparámetros basado en recocido simulado. Con el algoritmo propuesto, se estudian algunos de los mecanismos de compensación entre los hiperparámetros que gobiernan la construcción del modelo, teniendo en consideración que esta dinámica en muchos casos es desconocida.

Puede decirse finalmente que el control de la división del conjunto de datos a menudo reporta beneficios al reproducir fielmente la distribución estadística de datos. Asimismo, se observa la

importancia de contar con algoritmos más refinados para la optimización de los hiperparámetros.

Abstract

Artificial neural networks (ANN) have recently had a great impact on the resolution of some problems because it has been possible to implement algorithms that require a large computational capacity. Moreover, huge volumes of data have simultaneously been made available to train these models. Although computational capacity is no longer a constraint, the availability of data volumes may be a problem in some contexts where it is sometimes unfeasible to obtain new data.

In this thesis, deep neural network models with small databases circumscribed to regression problems are analyzed. A regression task is defined as the mapping of a numerical value with reference to a numerical input. The following characteristics are expected to be fulfilled: unbias, consistency and efficiency. In addition, a definition of a small database for ANNs has been proposed for practical purposes: the number of parameters of an ANN is a measure of its complexity. The trade-off between the parameters and the cardinality of the set of observations indicates a small database problem. It has also been considered that this is a necessary definition, but always definition.

The study of the dataset division on model evaluation is focused on situations where a regressor built with an ANN is employed. As a contribution, it has been studied whether variance control produces a more accurate estimation of the generalization error when the same starting point is considered. Consequently, an original evolutionary algorithm that makes it possible to control the division of the data according to a distance is presented. Another original contribution is the introduction of a hybrid hyperparameter optimization algorithm based on simulated annealing. The trade-off mechanisms between the hyperparameters governing the model construction are studied with the proposed algorithm, taking into consideration that these dynamics are in often cases unknown.

In summary, it is benefitial to control data-set splitting in the order to reproduce faithfully the statistic data distribution. The importance of more refined algorithms for hyperparameter optimization is also noticeable.

Agradecimientos

Tal vez la gratitud no sea la virtud más importante,

pero sí es la madre de todas las demás.

– Marco Tulio Cicerón

Agradezco a mi familia y amigos por su paciencia a pesar de aquellas ocasiones en las que no puede estar a causa de esta tesis. Quiero agradecer a mis seres queridos por su comprensión y apoyo incondicional. En particular quiero expresar mi gratitud a Daniel Arrieta, Maria Baccaro, Rosana Bertone, Rodrigo Cañas, Mónica Diaz, Mauro Fernandez, Carlos Gigola, Martin de Meio, Marcelo Zabaloy y el Comité Técnico Ejecutivo.

Agradezco especialmente, a mi Directora, Dra. Beatriz Nélida Brignole por enseñarme el valor de la duda. Mi eterna gratitud por su enseñanza y dedicación.

Índice general

Índice de figuras

Índice de cuadros

Acrónimos y abreviaturas

ADALINE	Elemento lineal adaptativo
AE	Autocodificador - *AutoEncoders*
AED	Autocodificador disperso
ANN	Redes neuronales artificiales - *Artificial Neural Networks*
AI	Inteligencia artificial - *Artificial Intelligence*
BP	Propagación hacia atrás - *Back Propagation*
DGP	Proceso Generador de los Datos - *Data Generating Process*
DL	Aprendizaje profundo - *Deep Learning*
DNN	Redes neuronales profundas - *Deep Neural Networks*
DNNM	Redes neuronales profundas multicapa
DS	Ciencias de datos - *Data Science*
EEE	Error medio esperado
F1	Friedman 1
F2	Friedman 2
F3	Friedman 3
FP	Propagación hacia adelante - *Forward Propagation*
fig.	figura
FP	Propagación hacia adelante - *Forward Propagation*
GA	Algoritmo genético - *Genetic Algorithm*
GAN	Modelos generativos adversos - *Generative Adversial Models*
GP	Programación genética
GRU	Unidad de compuerta recurrente - *Gated Recurrent Unit*
HPO	Optimización de hiperparámetros - *Hyperparameter Optimization*
IQR	Rango intercuartílico
LR	Tasa de aprendizaje - *Learning Rate*

LT	Ticket de lotería - *Lottery ticket*
MAE	Error medio absoluto - *Mean Absolute Error*
MAPE	Error porcentual medio absoluto - *Mean Absolute Percentage Error*
ML	Aprendizaje automático - *Machine Learning*
MLP	Perceptrón multicapa - *Multilayer Perceptron*
MR	Modelo de regresión
NN	Redes neuronales - *Neural Networks*
NNP	Red neuronal profunda
PM	Material particulado
ReLU	Función de activación lineal rectificada - *Rectified Linear Unit*
RL	Aprendizaje por refuerzo - *Reinforcement learning*
RS	Búsqueda aleatoria - *Random Search*
SGD	Descenso estocástico del gradiente - *Stochastic Gradient Descent*
TL	Transferencia de aprendizaje - *Transfer Learning*
VAE	Autocodificador variacional - *Variational AutoEncoder*

1. Introducción

Contenidos

En este capítulo se introduce la motivación y la relevancia del tema de investigación de esta tesis. A continuación, se presenta el objetivo general y los objetivos específicos y el método utilizado para alcanzar los mismos, también se dimensiona el aporte teórico realizado.

Por último, se dan a conocer las aplicaciones específicas para lograr una percepción entre la aplicabilidad y el aporte teórico realizado.

1.1. Motivación

We are drowning in information and starving for knowledge.[1]

–Rutherford D. Roger

En el infierno de la mitología griega, algunos sirvientes de Hades eran seres fabricados con cierta autonomía. Si bien la palabra robot, que proviene del idioma checo, no adquiere su sentido actual hasta la década del 1920, podemos afirmar con cierta libertad que los sirvientes del infierno eran robots. Esto sugiere que el concepto de inteligencia artificial (AI: Artificial Intelligence) ha estado presente durante milenios.

Actualmente, la AI no es una quimera sino un área cimentada de las Ciencias de la Computación. Como en el caso de los sirvientes de Hades, cuyo desempeño podía ser tanto corporal como intelectual, algunos consideran que la inteligencia es la emulación de una característica introspectiva de los procesos de pensamiento-razonamiento, mientras que otros se enfocan en el comportamiento inteligente como una característica externa observable.

El aprendizaje automático (ML: Machine Learning) es un subcampo de la IA que estudia la capacidad de mejorar el rendimiento en función de la experiencia. Algunos sistemas de IA utilizan métodos de ML para lograr aptitud. Y otros, por ejemplo, recurren a heurísticas.

El aprendizaje profundo (DL: Deep Learning) tiene su origen en los primeros trabajos que intentaron emular algunos aspectos de la biología cerebral. Por esta razón, las redes entrenadas por métodos de DL a menudo se denominan redes neuronales (NN: Neural Networks), aunque el parecido con las células y estructuras biológicas sea superficial.

El DL es una amplia familia de técnicas para el ML que en la computación toma la forma de circuitos algebraicos complejos con puntos fuertes de conexión condicional.

El DL es hoy el enfoque más utilizado para aplicaciones como el reconocimiento de objetos, la traducción automática, el reconocimiento de voz, la síntesis de voz y la síntesis de imágenes; también juega un papel importante en las aplicaciones de aprendizaje por refuerzo (RL: Reinforcement Learning).

La Ciencia de datos (DS: Data Science) es un nuevo actor que se ha establecido como un campo científico, y un paradigma que impulsa la evolución de la investigación en disciplinas

[1] Colin, 1985. “Nos estamos ahogando en información y estamos hambrientos de conocimiento”. [Col85]

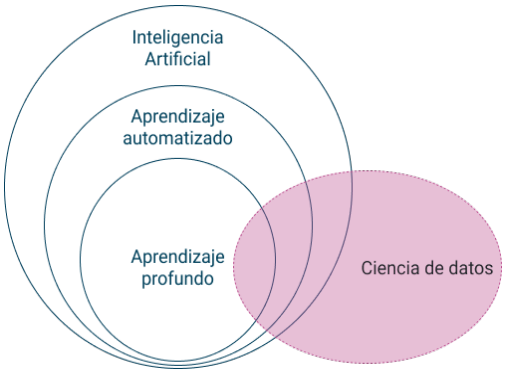


Figura 1.1: Diagrama de Venn

como la estadística, las ciencias de la computación y la IA.

Entonces, ¿cómo encaja la DS en este panorama? Esta disciplina parece recientemente creada pero, en realidad, el concepto fue encontrado en material bibliográfico del ganador del Premio Turing Peter Naur, como una alternativa a la ciencia de la computación en la década de 1960 [Cao17]. Esto ha planteado cierta confusión sobre los alcances de la DS.

La DS abarca amplias áreas de estadística, AI, análisis de datos, ML, reconocimiento de patrones, comprensión del lenguaje natural, manipulación de macrodatos y procesamiento de grandes volúmenes de datos. La figura (fig.) 1.1 es un diagrama de Venn que muestra cómo se conjugan estas áreas.

Las NN han tenido en los últimos años un notable impacto en la resolución de algunos problemas. Se han visto avances en las arquitecturas de las NNs diseñadas a medida de áreas de aplicación específicas. Las innovaciones provienen de modificaciones algorítmicas que han conducido a ganancias significativas en el desempeño para varios campos. Algunas de estas innovaciones incluyen pre-entrenamiento, la deserción, aumento de datos mediante ejemplos virtuales, normalizaciones batch y aprendizaje residual. En [GBC16] se elaboro una exposición detallada. Esta lista parcial de innovaciones algorítmicas potenciales resalta la rapidez y continuidad del progreso en este campo.

Estos avances se han llevado a cabo porque ha sido posible la implementación de algoritmos que requieren una gran capacidad de cómputo. Paralelamente, enormes volúmenes de datos se han puesto a disposición para el entrenamiento de estos modelos. A diferencia de lo que ocurre con la capacidad de cómputo que ya no supone una restricción, la disponibilidad de volúmenes de datos puede ser un problema en algunos contextos. Esta limitación se debe al costo

de adquirir nueva información. Sin embargo, en otras situaciones no es factible obtener nuevos datos dado que los disponibles son todos los que pudieran utilizarse referidos a cierto problema en particular. Las NNs tienen la propiedad de ser aproximadores universales [HSW89], por este motivo siempre serán un posible modelo candidato en un problema. Sin embargo, su desempeño está fuertemente condicionado por los datos disponibles.

En esta tesis se estudia el caso de la construcción de modelos de NN profundas (DNN: Deep Neural Networks) con bases de datos pequeñas tabulares[2]. Las prácticas actuales para el desarrollo de una NN dependen fuertemente del supuesto de la disponibilidad de grandes volúmenes de datos. Cuando este supuesto no se verifica, las prácticas para el desarrollo de estos modelos pueden tener efectos contraproducentes.

La comunidad científica ha expresado especial interés en esta cuestión y, por ende, es un tema abierto de investigación [GS11, GBC16, SK17]. La investigación sobre nuevas metodologías para aprendizaje de DNN a partir de datos pequeños sin información externa ha sido muy limitada [BD20]. De hecho, en el Encuentro Latinoamericano de Inteligencia Artificial "Khipu" que tuvo lugar en noviembre de 2019 en la ciudad de Montevideo (Uruguay) fue presentado como uno de los desafíos actuales del DL.[3]

1.2. Objetivos

En un problema de regresión se pretende que el modelo realice una predicción numérica con base en una entrada dada. Se espera de un método de regresión que cumpla con las siguientes propiedades[4]: carencia de sesgo, consistencia y eficiencia.

Cuando se observa carencia de sesgo, el método no introduce un error sistemático en las estimaciones al tratar de simplificar un problema complejo mediante un modelo más simple. En otras palabras, busca representar la relación real entre las variables de manera precisa y sin distorsiones.

Un estimador es consistente si, a medida que aumenta el tamaño de la muestra o la cantidad

[2]La expresión "base de datos pequeña" refiere a conjuntos de datos de pocas observaciones. No se trata de la tecnología de motores de bases de datos.

[3]Presentado en la ceremonia de apertura, puede consultarse en https://khipu.ai/home-2020/

[4]Las definiciones formales de estas propiedades serán presentadas posteriormente en la sección 2.2.2.

de experiencia, tiende a converger hacia el valor verdadero de la población. Esto significa que a medida que se dispone de más datos, las estimaciones del método se vuelven cada vez más precisas y se acercan al valor real que se busca estimar.

La eficiencia se relaciona con la precisión de las estimaciones generadas por el modelo de regresión. Un método eficiente produce estimaciones que son cercanas al valor real, lo que significa que es capaz de capturar de manera precisa la variabilidad en los datos y proporcionar resultados confiables.

El objetivo general de esta tesis es desarrollar una metodología para DNN en contextos de bases de datos pequeñas para problemas de regresión. En consecuencia, se buscará alcanzar un mejor desempeño de los modelos en términos de las propiedades de los mismos.

Los objetivos específicos son:

- Se explora la validez del método que se aplica comúnmente a grandes volúmenes respecto a cómo se realiza la división del conjunto de datos en el caso de bases de datos tabulares pequeñas para problemas de regresión.

- Identificar las propiedades estadísticas de los conjuntos de datos que permitan prever un mejor desempeño.

- Proponer un algoritmo de optimización de hiperparámetros (HPO: Hyperparameter Optimization) basado en el recocido simulado.

- Estudiar -con el HPO propuesto- alguno de los mecanismos de compensación entre los hiperparámetros que gobiernan la construcción de los modelos de DNN mediante un algoritmo propio de DL.

1.3. Organización de la tesis

En el capítulo 2 se realizará una introducción al ML y se presentará en profundidad el modelo de NN y DNN. Además, se expondrá el estado del arte actual. Se definirá cuándo decimos que estamos ante un contexto de base de datos pequeña.

En el capítulo 3 se analizarán distintas facetas de la división del conjunto de base de datos tabulares pequeño y la relación con las propiedades de un modelo de regresión (MR). En cuanto a la influencia de la varianza en la construcción de MR, se presentará un algoritmo que permita controlar sus efectos.

En el capítulo 4 se estudiarán los efectos de control de la división del conjunto de datos mediante experimentos que determinen cómo las prácticas usuales en el desarrollo de modelos de NN -generalmente, sin inconvenientes en cuanto a la disponibilidad de información- impactan en los contextos de bases de datos pequeñas para problemas de regresión.

En el capítulo 5 se propondrá un algoritmo de HPO basado en el recocido simulado libre de hiperparámetros propios. Asimismo, mediante el uso de este algoritmo, se estudiarán algunas dinámicas de hiperparámetros.

Finalmente, en el capítulo 6 se presentarán las conclusiones y una revisión sobre las contribuciones realizadas por esta tesis y se indicarán los lineamientos de los trabajos futuros con base en lo expuesto.

2. Sobre Redes Neuronales

Contenidos de este capítulo

La presente tesis se circunscribe al estudio de problemas de regresión mediante el uso de DNN. Una tarea de regresión se define como el mapeo de un valor numérico en referencia a una entrada también numérica.

Se presenta una descripción detallada de este modelo matemático que en sus orígenes pretendía ser un mero modelo de la célula biológica y una definición precisa de los mecanismos de aprendizaje utilizados en estos modelos.

Se propone una definición de base de datos pequeña para NN. Asimismo, se observa que es una definición necesaria pero perfectible y se expone una descripción del estado del arte concerniente al problema a tratar.

2.1. Introducción al aprendizaje automático

"Farmers combine seeds with nutrients to grow crops.

Learners combine knowledge with data to grow programs."[1]

– Pedro Domingos

Un algoritmo de ML es un algoritmo que puede aprender relaciones a partir de datos; sin embargo, esta aproximación es ambigua. Según [Mit97], una definición más precisa sería: "Un programa de computadora $\mathcal{P}$ con respecto a una clase de tarea $\mathcal{T}$ y una medida de desempeño $\mathcal{D}$, **si el desempeño en tareas $\mathcal{T}$, medido por $\mathcal{D}$, mejora con la experiencia $\mathcal{E}$**". Si bien no hace referencia explícita a la experiencia $\mathcal{E}$, lo anterior podría reescribirse como: un programa de computadora $\mathcal{P}$, se dice que aprende de una experiencia $\mathcal{E}$, con respecto a una clase de tarea $\mathcal{T}$ y una medida de desempeño $\mathcal{D}$, **si el desempeño en tareas $\mathcal{T}$, medido por $\mathcal{D}$, mejora con la experiencia $\mathcal{E}$**.

La experiencia $\mathcal{E}$ se define como una colección de observaciones en la que cada una está compuesta por un conjunto de atributos. Cada uno de estos atributos puede corresponder a diversos tipos de datos. En caso de que todos estos atributos fueran números, una observación puede ser considerada como un punto $x \in R^n$. Dependiendo de si x_i con $1 \leq \mathrm{i} \leq n$ toma valores discretos, estaremos ante atributos categóricos, en caso contrario será un atributo continuo.

El aprendizaje es el método de procesamiento de las observaciones a través del cual se adquiere la habilidad de realizar la tarea $\mathcal{T}$. De esta forma, la distinción entre el aprendizaje y la tarea queda explicitada. Una medida de desempeño $\mathcal{D}$ definirá el método de evaluación con el que se realiza la tarea $\mathcal{T}$.

Siguiendo el criterio planteado previamente, en la sección 2.2 se definirá el **programa $\mathcal{P}$,** objeto de estudio de esta tesis: las ANN.

A continuación, en la sección 2.2 se detallan:las ANN, las **tareas $\mathcal{T}$** en las que actualmente se aplican las misma, se hará especial énfasis en las tareas de regresión y en las propiedades deseables de un MR.En la sección 2.3 se enumerarán algunas de las arquitecturas de NN con más relevancia para esta tesis (distintos tipos de programas $\mathcal{P}$) junto con una breve descripción de

[1] "Los granjeros combinan semillas y nutrientes para el cultivo.
Los estudiosos combinan el conocimiento con los datos para cultivar los programas". [Dom12]

cada una de ellas. Luego, en la sección 2.4 se presentará una breve taxonomía de los mecanismos de aprendizaje acordes al estado del arte. En la sección 2.5 se resumirá la metodología para la construcción de una ANN. En la sección 2.6, se mostrarán aspectos específicos de lo que ocurre cuando la cardinalidad de la **experiencia** $\mathcal{E}$ es pequeña. Mientras que en la sección 2.7 se describirán aspectos específicos de los conjuntos de datos tabulares en problemas de regresión. Por último, se exponen las conclusiones del capítulo.

2.2. El programa P: redes neuronales artificiales

Las NN son modelos matemáticos que en sus orígenes pretendían ser un modelo de la célula biológica, la unidad funcional y estructural del sistema nervioso humano. Para la neurobiología, estas redes están compuestas por un grupo de células conectadas física y/ o químicamente. Cuando la conexión es química, la comunicación se realiza mediante un mecanismo de envío y recepción de distintos compuestos químicos llamados neurotransmisores; mientras que la conexión física se da a través de numerosas prolongaciones ramificadas desde el cuerpo celular -llamadas dendritas- que reciben una señal eléctrica enviada por otra célula [TA12].

El primer modelo computacional de una neurona fue propuesto por [MP43]. Con él se implementaron funciones lógicas básicas (por ejemplo: AND, OR, NOR). En 1958, [Ros58] propuso las llamadas perceptrón multicapa (en inglés: *Feedforward Neural Networks*) que en la actualidad son un bloque fundamental del DL[2] [GBC16]. La propuesta de [Ros58] fue motivada por interrogantes tales como:

1. *"¿Cómo es la información del mundo físico sentida o detectada por los sistemas biológicos?";*

2. *"¿Cuál es la forma en la que dicha información es almacenada o recordada?";*

3. Por último,*"¿cómo esta información almacenada tiene influencia en el reconocimiento de patrones o el comportamiento?"*.

[2]DL refiere al nombre con el que en la actualidad es conocida el área de estudio de las NN artificiales [GBC16].

Estas preguntas aún no han sido totalmente contestadas y siguen motivando la investigación en el área. Por ejemplo, en el caso del RL, una de las ideas que vinculan la neurociencia con la IA, se han publicado avances inspirados en recientes descubrimientos con respecto al mecanismo del neurotransmisor dopamina en el cerebro [DKNU$^+$20].

El DL ha adoptado diferentes nombres y recientemente es conocido como DL. En términos generales, ha habido tres instancias de desarrollo: la primera conocida como cibernética (1940 a 1960), la segunda como conexionismo (1980 a 1990) y, actualmente, con el nombre de DL, a partir de 2006.

El modelo de McCulloch-Pitts [MP43] era un modelo lineal que podía reconocer dos categorías diferentes de entradas probando si $f(x, w)$ es positiva o negativa. Sin embargo, para que el modelo correspondiera a la definición deseada de las categorías, los pesos debían establecerse correctamente por el operador humano. En la década de 1950, el perceptrón se convirtió en el primer modelo que podía aprender los pesos que definían las categorías dados ejemplos de entradas de cada categoría [Ros58].

El elemento lineal adaptativo (ADALINE), que data aproximadamente de la misma época, devolvió el valor de $off(x)$ para predecir un número real [WH60] y pudo aprender a predecir estos números a partir de los datos. El algoritmo de entrenamiento utilizado para adaptar los pesos de ADALINE fue un caso especial llamado descenso de gradiente estocástico. Versiones ligeramente modificadas de este algoritmo se utilizan de forma dominante para modelos actuales.

Se produjo el fin de la primera etapa cuando en [MP69] se demostró que el perceptrón no era capaz de resolver problemas no lineales. Esto puso en evidencia las limitaciones del perceptrón, dado que estas funciones son extensamente empleadas en los problemas del mundo real.

En la década de 1980, la segunda época de investigación surgió en el contexto de la ciencia cognitiva, a través de un movimiento llamado conexionismo o procesamiento distribuido paralelo [RHW86, McC95]. La ciencia cognitiva es un enfoque interdisciplinario para comprender la mente que combina múltiples niveles de análisis. La idea central del conexionismo es que una gran cantidad de unidades computacionales simples pueden lograr un comportamiento inteligente cuando se conectan en red. Esta idea se aplica tanto a las neuronas en los sistemas nerviosos biológicos como a las unidades ocultas en los modelos computacionales.

Varios conceptos clave surgieron en el movimiento conexionista de la década de 1980 que son fundamentales para el DL actual. Uno de estos conceptos es el de representación distribuida [Hin86]. De acuerdo a esta concepción, cada entrada a un sistema debe estar representada por muchas características y cada característica debe estar involucrada en la representación de varias entradas posibles.

Otro logro importante del movimiento conexionista fue el uso de la retropropagación para entrenar DNN con representaciones internas y la difusión del algoritmo de retropropagación [RHW86]. La frecuencia en el uso de este algoritmo ha tenido altibajos, sin embargo, es el enfoque dominante para entrenar modelos profundos. De esta forma se logra entrenar múltiples capas con funciones de activación no lineales. En [HSW89] demuestran que el perceptrón multicapa es un aproximador universal sobreponiéndose a la limitación demostrada por [MS69]. Un perceptrón multicapa puede aproximar relaciones no lineales entre los datos de entrada y salida.

La tercera etapa en la investigación de NN comenzó con un avance relevante en 2006. Hinton demostró que un tipo de red neuronal llamada red de creencias profundas podía entrenarse de manera eficiente usando una estrategia llamada pre entrenamiento por capas "greedy" [HOT06]. Otras investigaciones demostraron que la misma estrategia podía usarse para entrenar otros tipos de redes profundas [BLPL06, RPCC06] y ayudaron sistemáticamente a mejorar la generalización en los ejemplos de prueba. En esta época se extendió el uso del término DL para enfatizar que los investigadores podían entrenar NN más profundas de lo que había sido posible, y centrar la atención en la importancia teórica de la profundidad del modelo [BL07, DB11, PMB13, MPCB14] . Entre 2009 y 2012, las NN comenzaron a ganar notoriedad en concursos de reconocimiento de imágenes, de patrones y de escritura a mano.

2.2.1. Las tareas $\mathcal{T}$ resolubles mediantes redes neuronales

El aprendizaje es el método de procesamiento por el cual se adquiere la destreza de realizar la tarea $\mathcal{T}$. Estas tareas pueden responder a distintos objetivos. A continuación se describirán algunas a modo de ejemplo. En una tarea de **clasificación**, se pide asignar a una entrada x a

qué categoría c pertenece de una cantidad $k \in \mathbb{N}$ de categorías. Aprender[3] este tipo de tareas se traduce en determinar una función $f : R^n \rightarrow \{1, ..., k\}$. Llamamos clasificador al programa que la resuelve.

El **clasificador probabilístico,** en lugar de asignar una categoría a una entrada, determina la probabilidad de que dicha entrada pertenezca a una categoría k[4]. La clasificación se vuelve más desafiante cuando no todas las dimensiones del vector de entrada perteneciente a $x \in R^n$ se encuentran definidas. Este caso se relaciona con otro tipo de problema conocido como **imputación de valores faltantes**, en el que se trata de predecir el valor del x_i con $1 \leq i \leq n$.

Las tareas de **transcripción**, **traducción mecanizada** y **salida estructurada** se caracterizan por recibir como entrada una secuencia. En el caso de una transcripción se trata de una entrada desestructurada y la tarea consiste en estructurarla de alguna forma. En el caso de la traducción mecanizada, la entrada debe ser traducida de un lenguaje a otro, cuya aplicación más notoria es en el área de lenguaje natural, por ejemplo traducir un texto de inglés al francés o al alemán [VSP+17].

Otras tareas a mencionar son la **detección de anomalías**, **muestreo y resumen**, **eliminación de ruido** [GBC16].

En el problema de **estimación de una función de densidad** se pide que se aprenda la función $p_{modelo} : \mathbb{R}^n \rightarrow \mathbb{R}$, donde p_{modelo} puede ser interpretada como una función de densidad de probabilidades, dependiendo de que sea una distribución continua o discreta. Se espera que el algoritmo sepa cómo se agrupan las observaciones y qué agrupamientos son menos probables. Si apelamos a una aplicación de ejemplo, podría usarse la estimación de densidad de la distribución aprendida para resolver una imputación de un valor faltante.

2.2.2. Tareas de regresión

Se ha demostrado que las NN tienen la propiedad de ser aproximadores universales [HSW89]. Por este motivo serán un posible modelo candidato en un problema de regresión. La presente

[3]Téngase en consideración que la concepción de aprendizaje en este campo se limita a la forma en que se resuelve la tarea. No se desconoce la discusión en torno a lo que significa filosóficamente este término.

[4]Es usual no solo determinar la clase, sino también obtener una probabilidad de la etiqueta respectiva. Esto supone el desafío de que las probabilidades se encuentren bien calibradas. La calibración permite que el resultado se interprete con un nivel de confianza estadístico.

tesis se circunscribe a estos problemas mediante el uso de DNN, por esto presentaremos este tipo de tarea con mayor detalle.

Una tarea de **regresión** se define como el mapeo de un valor numérico en referencia a una entrada también numérica, es decir una función:

$$f : D^k \subseteq R^k \rightarrow I^m \subseteq R^m \tag{2.1}$$

Tal que

$$y = f(x) \tag{2.2}$$

dónde:

- $k \in \mathbb{N}$ y $m \in \mathbb{N}$
- $y \in I^m$ es la variable objetivo o valor observado
- $X \in T \subseteq D^k$ se define como los atributos de entrada
- $\widehat{f}(X)$ o $\widehat{y}$ es un modelo de regresor o estimador entrenado sobre un conjunto T. tal que $n = |T|$ es la cardinalidad del conjunto de entrenamiento. Alternativamente, $\widehat{f}(X)$ o $\widehat{y}$ representará el valor de las predicciones por el modelo de estimador.

Si $m = 1$, entonces decimos que se trata de un problema de regresión simple. Caso contrario, se dirá que es un problema de regresión múltiple.

En la estadística clásica se requiere que un conjunto de suposiciones se cumpla para aplicar los métodos propuestos por esta disciplina con confiabilidad. Sin embargo, no siempre este conjunto de supuestos es verificado [OW02, WGGK13]. En el caso de las técnicas de ML en general y en las NN en particular, no se hacen supuestos y, por lo tanto, tampoco es necesario realizar estas verificaciones. Por lo que resulta una técnica alternativa cuando no pueden verificarse tales presunciones [OW02].

Se espera que un método de regresión cumpla con las siguientes características:

- Que sea carente de sesgo.

- Consistencia: un estimador es consistente si, a medida que crece el tamaño de la muestra o experiencia, converge al valor verdadero de la población. Esto es: a medida que aumenta la cardinalidad de la experiencia tiende a mejorar su precisión.
- Eficiencia: refiere a la precisión de las estimaciones producidas por el MR.

El sesgo de un estimador es definido como:

$$Sesgo(\widehat{\Theta}_m) \ = E(\widehat{\Theta}) - \ \Theta \tag{2.3}$$

dónde la esperanza es calculada sobre los datos que son considerados como una muestra de una variable aleatoria y Θ es la verdadera media de la distribución o proceso generador de los datos (DGP: Data Generating Process).

Se dice que un estimador es insesgado si el valor medio esperado es igual al valor medio de la población, es decir que $\widehat{\Theta}_m$ se dice insesgado si $sesgo(\widehat{\Theta}_m) = 0$. Por otro lado, un estimador $\widehat{\Theta}$ se dice asintóticamente insesgado si $\lim_{m\to\infty} E(\widehat{\Theta}) = \Theta$

Existen múltiples medidas que permiten evaluar la calidad de las estimaciones en términos de las propiedades esperables de un MR. Estas evaluaciones suelen realizarse en virtud de las diferencias entre y e $\widehat{y}$ y son estimación de los valores de estas propiedades.

El caso del error de sesgo medio (MBE: *Mean Bias Error*) para un conjunto T, se define como:

$$MBE_T = \frac{\sum_{x\in T}(\widehat{f}(x) - y)}{n} \tag{2.4}$$

Otra medida de desempeño de los estimadores se conoce como error medio absoluto (MAE: *Mean Absolute Error*) para un conjunto T. Se define como:

$$MAE_T = \frac{\sum_{x\in T}\left|\widehat{f}(x) - y\right|}{n} \tag{2.5}$$

Por definición, $MAE_T \geq 0$ para todo T, es claro que $MAE_T \geq MBE_T$. Así mismo, también se verifica que $MAE_T \geq MBE_T \geq -MAE_T$, por lo que el valor de MAE_T acota tanto superior como inferiormente al valor de MBE_T para todo T [WM05].

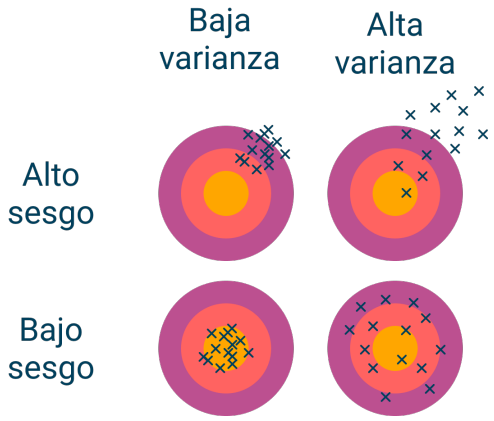


Figura 2.1: Sesgo y varianza en los datos

Probablemente, la medida más frecuente de evaluación de los modelos sea el error medio cuadrado (del inglés: *mean squared error*, MSE) para un conjunto T, definida como:

$$MSE_T = \frac{\sum_{x \in T} (\widehat{f}(x) \ - \ y)^2}{n} \tag{2.6}$$

Cuando estas medidas son calculadas sobre un conjunto $T' \neq T$, dónde T' es una muestra independiente tomada de forma aleatoria de la distribución conjunta de la cual fue extraída el conjunto de entrenamiento T' [GBC16, HTFF17], a este conjunto T' se lo conoce como conjunto de testeo y a sus medidas de error como error de generalización.

El error de generalización puede ser descompuesto en sesgo y varianza [Dom12, HTFF17]. Esta descomposición plantea un dilema entre ellos: "*the bias-variance dilemma*", un fenómeno bien conocido en el ML [HTFF17] (Ver Cap. 3.2). El origen de este dilema surge al tratar de minimizar simultáneamente estas dos fuentes diferentes de error. En la fig. 2.1 se muestra cómo varía el comportamiento de un estimador en los distintos escenarios que puede presentar el dilema. El estado ideal sería lograr los valores más bajos de sesgo y varianza.

La capacidad de un modelo es la habilidad de aproximar a una amplia variedad de funciones. De manera informal, puede decirse que los problemas de generalización del modelo se manifiestan en dos formas: subajuste o sobreajuste (del inglés *overfitting*). El subajuste se produce cuando el modelo no puede lograr un error aceptable en el conjunto utilizado en su construcción (error de entrenamiento). El sobreajuste ocurre cuando la brecha entre el error de entrenamiento y el de testeo es amplia. Los modelos con poca capacidad tienen tendencia a subajustar, mientras

que los modelos con alta capacidad pueden sobreajustarse. Para los mismos puntos, un modelo puede subajustar, ajustar adecuadamente, o sobreajustar.

Decimos que se produce un sobreajuste cuando el modelo se desempeña apropiadamente en el conjunto de entrenamiento, mientras que presenta un amplio desajuste en el conjunto de prueba. Por otro lado, los modelos sobreajustados tienden a memorizar todos los datos, incluido el ruido inevitable en el conjunto de entrenamiento, en lugar de generalizar las relaciones ocultas de los datos.

2.3. Tipos de redes neuronales

Gran variedad de arquitecturas de redes neuronales han sido aplicadas en las más diversas áreas del conocimiento. Algunos problemas provienen de la agricultura, ciencias médicas, educación, finanzas, seguridad, ingeniería, transporte, energía, etc. [AJO+18]. Se presentará una breve recopilación de las arquitecturas más relevantes para el estado del arte, con especial énfasis en la que es objeto de estudio de esta tesis. En el Anexo A se hará referencia a los modelos actuales de mayor relevancia.

2.3.1. Redes profundas multicapa

Como se menciono anteriormente, las NN multicapa tienen la propiedad de ser aproximadores universales cuando cuentan con la suficiente cantidad de capas y unidades [HSW89].

Las DNN multicapa (DNNM), o perceptrones multicapa (MLP, *del inglés Multilayer Perceptron*) son los modelos de DL por excelencia. Esta arquitectura está compuesta por múltiples capas de neuronas artificiales o unidades. Las capas están compuestas por un conjunto de unidades interconectadas.

Conceptualmente, las capas son conjuntos de unidades en paralelo, donde cada una de ellas representa la función de un vector a un escalar ($\mathbb{R}^n \rightarrow \mathbb{R}$). En la figura 2.2 puede observarse cómo se realiza la computación dentro de los mismos. Cada una de las entradas de la unidad está ponderada por un valor real. Para producir la salida, la suma de las entradas ponderadas es transformada por la función de activación f. Cada valor numérico recibido por la unidad es

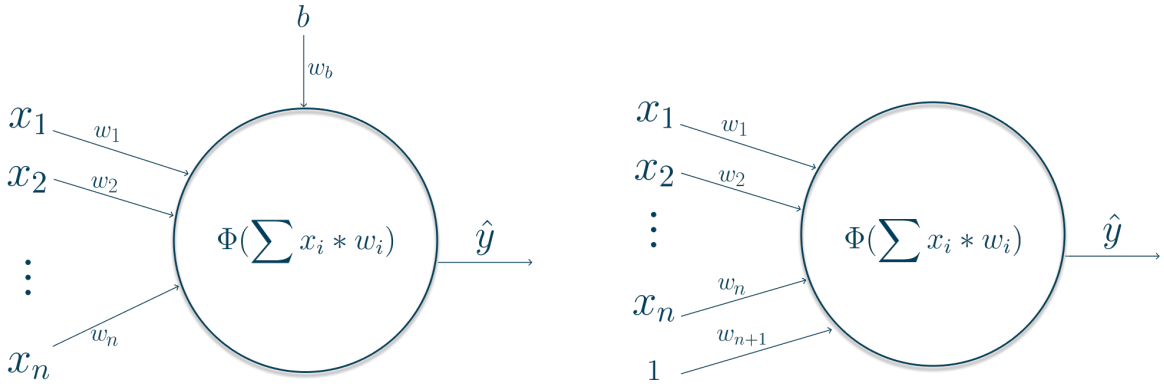


Figura 2.2: Unidad de una MLP

ponderado por un peso específico (similar a la capacidad estimuladora de cada dendrita) y sobre esta suma ponderada se aplica la función de activación que da como resultado la salida de esa unidad hacia otras unidades. De esta forma, el flujo de información es análogo al impulso del sistema nervioso en los animales.

El objetivo de una DNNM es aproximar alguna función f^* . Una red define un mapeo $y = f(x, \bar{\Theta})$, donde $\bar{\Theta}$ corresponde al valor de los pesos o parámetros. Idealmente, se desea encontrar aquellos $\bar{\Theta}$ que resulten en la mejor aproximación de la función f^*. El cómputo de estos parámetros puede realizarse de varias maneras. Una de estas es analíticamente. Sin embargo, el DL está impulsado por un algoritmo principal: el descenso estocástico del gradiente (SGD: *Stochastic Gradient Descent*) [GBC16].

Supongamos que tenemos una función $y = f(x)$ con $x, y \in \mathbb{R}$. La derivada $f'(x)$ indica la pendiente de $f(x)$ en el punto x. En otras palabras, especifica cómo escalar un pequeño cambio en la entrada para obtener el cambio correspondiente en la salida: $f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$. Por lo tanto, la derivada es de utilidad para minimizar una función, dado que indica cómo un pequeño cambio en x permite hacer una pequeña mejora en y. Por ejemplo, sabemos que $f(x) - \varepsilon\ signo(f'(x)) \to 0$ para un ε suficientemente pequeño. Por tanto, podemos reducir $f(x)$ alterando x en pequeños pasos con el signo opuesto de la derivada. Esta técnica se denomina descenso de gradiente [Cau47].

En la figura 2.3 se muestran algunas de las funciones de activación utilizadas. En el caso de la función de activación lineal rectificada (ReLU: *Rectified Linear Unit*), se considera la recomendación estándar para la mayoría de las implementaciones [GBC16]. Esta función se

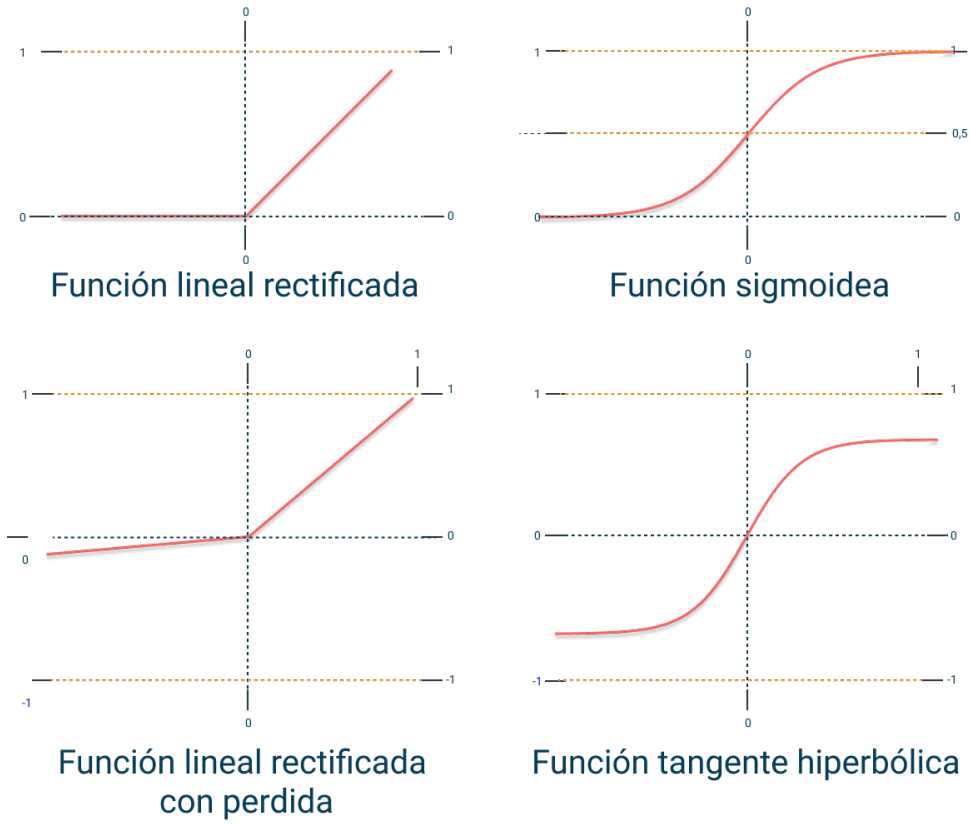


Figura 2.3: Funciones de activación

define como $g(z) = max\{0, z\}$ [JKRL09, NH09, GBB11].

Aunque con esta función se resuelven algunos problemas planteados por otras funciones de activación[5], se producen nuevos[6]. Es el caso de la unidad ReLU moribunda (del inglés, *Dying ReLU*), una propuesta superadora es la Relu con filtraciones (del inglés, *Leaky ReLu*). Leaky ReLU redefine lo que ocurre cuando la entrada es negativa y lo hace agregando una ligera pendiente en este rango. Esto modifica la función para generar pequeñas salidas negativas cuando la entrada es menor que 0. En la figura 2.3 se muestra una composición de diferentes funciones de activación, en particular se puede observar las funciones mencionadas. ReLU se mantiene

[5]Las funciones tangente hiperbólica y sigmoidea son propensas al problema de la desaparición del gradiente, donde los gradientes se reducen drásticamente en la etapa de retropropagación, de modo que la red ya no puede aprender. ReLU evita esto preservando el gradiente, ya que en el rango de entrada positivo permite que los gradientes fluyan bien en las rutas activas de las unidades y permanezcan proporcionales a las activaciones de los nodos y es una función sin valor máximo.

[6]El problema de la ReLU moribunda se refiere al escenario en el que gran cantidad de unidades ReLU solo emiten cero. Si bien esta característica es lo que le da a ReLU sus fortalezas, se convierte en un problema cuando la mayoría de las entradas están en el rango negativo. Cuando toda la red muere, se convierte en una función constante. Debido a que la pendiente de ReLU en el rango de entrada negativo también es cero. Una vez que se muere (es decir, se atasca en el rango negativo y da cero), es probable que permanezca irrecuperable. Cuando la mayoría devuelve cero, los gradientes no fluyen durante la propagación hacia atrás y los pesos no se actualizan. Al final, una gran parte de la red se vuelve inactiva y no aprende más.

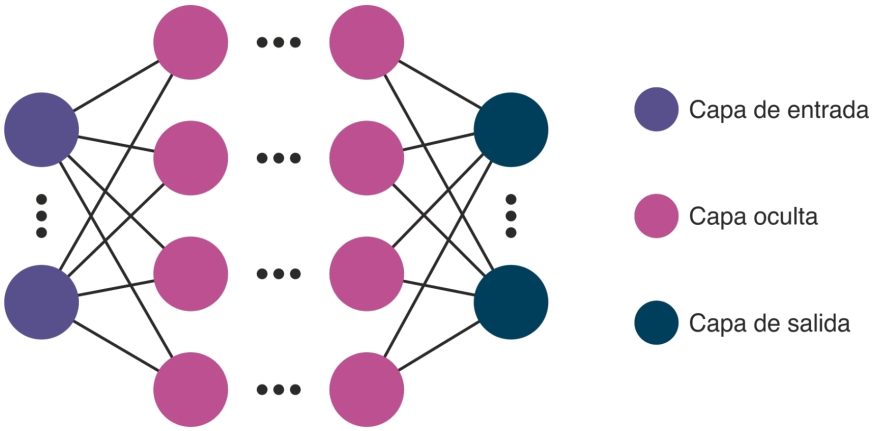


Figura 2.4: Red multicapa profunda

constante en cero cuando la entrada es negativa. Contrariamente, Leaky ReLU, en lugar de mantenerse constante, varía linealmente con una pequeña pendiente alpha. De esta manera el problema de la Relu moribunda no puede manifestarse.

En las NN alimentadas hacia adelante (del inglés: *feedforward*), la información fluye en un solo sentido: desde la capa de entrada hacia la capa de salida. Una capa puede ser clasificada como de entrada, oculta o de salida. En la figura 2.4 se puede observar un diseño de esta red. La capa de entrada recibe la información del exterior de la red y es transmitida hacia el interior: las capas ocultas. Las capas ocultas poseen conexiones con las capas adyacentes. Por último, la capa de salida comunicará al exterior de la red el resultado de la computación.

2.4. Tipos de entrenamiento

La forma en la que los algoritmos hacen uso de la experiencia durante el aprendizaje permite elaborar una taxonomía de los mismos. Tradicionalmente, los algoritmos se categorizan en supervisado, semi-supervisado y no supervisado. El término “supervisar” se origina en que el objetivo a realizar es provisto por un maestro o instructor que le indica al sistema qué es lo que se pretende hacer. En el primer caso, el aprendizaje se realiza a través de una explotación explícita de la asociación de cada observación con su etiqueta o un valor.

En el caso no supervisado, el aprendizaje se realiza a través de la búsqueda de estructuras en el conjunto de datos. En el aprendizaje no supervisado (en el contexto del DL) se identifican características útiles a partir de la estructura de la experiencia. Se pretende que se aproxime

la distribución de probabilidades que genera el conjunto de datos, ya sea de forma explícita o implícita [GBC16].

Por otro lado, en semi-supervisado, el aprendizaje se combina la información proporcionada tanto por ejemplos no etiquetados como por ejemplos etiquetados para llevar a cabo la tarea, creando así una categoría intermedia. En el contexto del aprendizaje profundo, el aprendizaje semi-supervisado generalmente se refiere a aprender una representación de manera que los ejemplos de la misma clase tengan representaciones similares [GBC16].

El RL podría considerarse una subcategoría del aprendizaje no supervisado que presenta particularidades frente a los algoritmos. Esta técnica aprovecha la continua interacción con el entorno para aprender de su experiencia. En particular, la combinación de DNN y el RL han mejorado significativamente el desempeño de la robótica [FTD$^+$16].

En las próximas subsecciones se expondrán los detalles de cómo se realiza el aprendizaje supervisado en la red perceptrón multicapa. Por último, se hará referencia a otros tipos de aprendizaje para las NN utilizados por otras arquitecturas.

2.4.1. Retropropagación

El término backpropagation o retropropagación (BP: *Backpropagation*) se refiere únicamente al método de cómputo del gradiente, mientras que otro algoritmo es utilizado para realizar la actualización de los pesos considerando el gradiente previamente calculado.

Cuando se utiliza una red multicapa para procesar una entrada x y producir una salida $\widehat{y}$, la información fluye hacia adelante (i.e. en dirección hacia la capa de salida). La entrada x provee la información inicial que luego es propagada hacia las capas ocultas para producir $\widehat{y}$. Esto es llamado propagación hacia adelante (FP: *Forward Propagation*).

Durante la etapa de entrenamiento, el FP continua hasta calcular un escalar $J(\theta)$. El algoritmo de BP permite que la información de costo fluya en el sentido de la capa de salida hacia la de entrada para computar el gradiente.

2.4.2. Optimización

ML tiene sus fundamentos en la optimización. En particular, las NN optimizan sobre una función compuesta de circuitos algebraicos complejos. Además, el uso de estrategias diseñadas explícitamente para reducir el error de testeo (conocidas como técnicas de regularización[7]) son cruciales para prevenir el sobreajuste. Esta estructura general de optimización es central del DL. La mayoría de los algoritmos de aprendizaje involucran alguna forma de optimización: nos referimos a la tarea de maximizar o minimizar una función $f(x)$ mediante una alteración de x. En este contexto usualmente se quiere minimizar la función de pérdida o de error (también llamada función de costo).

Supongamos que tenemos una función $y = f(x)$, donde $x \in R$ e $y \in R$. Recordemos que la derivada $f'(x)$ resulta en la pendiente de $f(x)$ en el punto x. De esta forma, se especifica cómo escalar un pequeño cambio en la entrada para obtener el cambio correspondiente en la salida: $f(x + \varepsilon) \approx f(x) + \varepsilon * f'(x)$.

En consecuencia, la derivada es útil para minimizar una función, ya que indica cómo un cambio en x puede implicar una mejora en y. Es sabido que reduciendo iterativamente $f(x)$ con el signo opuesto de la derivada podemos hallar el mínimo de esta función [Cau47]. Los puntos donde $f'(x) = 0$ se conocen como puntos críticos o puntos estacionarios: estos no proporcionan información que indique la dirección a tomar. En la Fig. 2.5 se muestran diferentes tipos de puntos críticos: mínimo, máximo y punto silla[8]. El mínimo local es un punto donde $f(x)$ es más bajo que en todos los puntos vecinos, por lo que ya no es posible disminuir $f(x)$ haciendo pasos infinitesimales. El máximo local es un punto donde $f(x)$ es más alto que en todos los puntos vecinos. Esta estrategia es la base de los algoritmos de optimización usados en la actualidad.

En el caso del DL, es posible que las funciones a optimizar puedan tener muchos mínimos locales que no son óptimos y muchos puntos de silla rodeados por regiones muy planas. Estas situaciones suponen una dificultad adicional para la optimización, en particular cuando la entrada x es multidimensional. Así pues, se suele tolerar un valor muy bajo pero no necesariamente mínimo en ningún sentido formal. En la figura 2.6 se muestran diferentes situaciones que podrían

[7] En la sección 4.4 se presenta la deserción ó dropout, ampliamente difundida en los usos del DL actual.

[8] En este caso se trata de un punto en el cual cambia la concavidad de la función. En términos de su derivada, refiere a un punto en el cual se produce un cambio de signo en su entorno reducido.

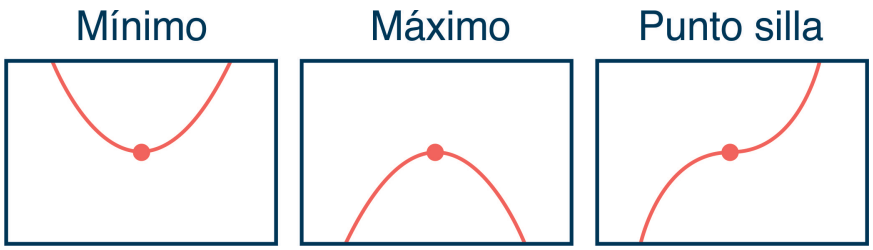


Figura 2.5: Puntos de interés

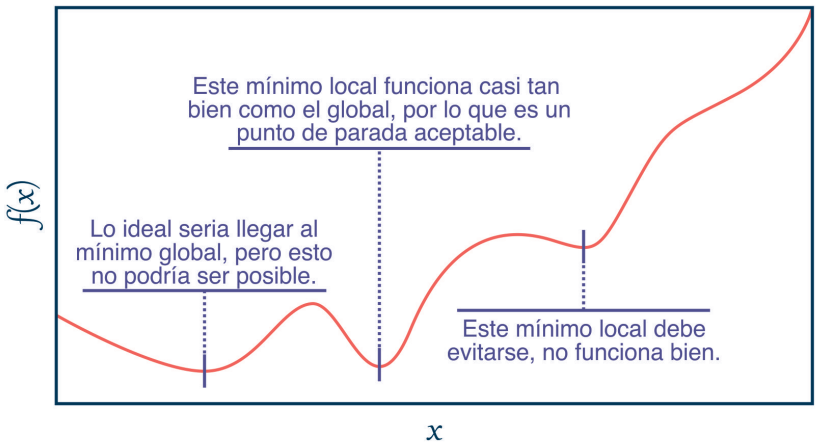


Figura 2.6: Situaciones que se pueden presentar durante el entrenamiento

presentarse. Los algoritmos de optimización podrían fallar al encontrar un mínimo global cuando hay múltiples mínimos o mesetas locales. En el DL, generalmente son toleradas tales soluciones aunque no sean realmente mínimas, siempre y cuando correspondan a valores significativamente bajos de la función de costo [GBC16].

El SGD y su familia de variantes son probablemente los algoritmos de optimización más utilizados para el entrenamiento de las DNN.

Un parámetro crucial de este algoritmo es la tasa de aprendizaje (LR: *Learning Rate*). Si bien este parámetro puede tomar un valor fijo durante, en la práctica resulta necesario que este valor sea paulatinamente disminuido siguiendo la planificación (del inglés: *rate schedule*) [MHS+13].

Aunque este algoritmo ha demostrado ser central en el éxito de muchas aplicaciones, en ciertas ocasiones, en las que la función objetivo tiene mucho ruido [HSK+12], se requerirían otras técnicas. Para esta situación, [KB15] presenta un algoritmo llamada Adam (cuyo nombre proviene del inglés *"adaptive moment estimation"*) para la optimización basada en gradientes de primer orden de funciones objetivas estocásticas, basado en estimaciones adaptativas de

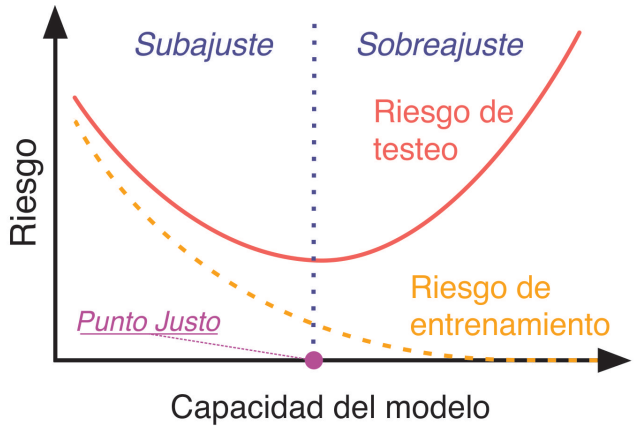


Figura 2.7: Esquema tradicional

momentos de orden inferior. El método es sencillo de implementar, es computacionalmente eficiente, tiene pocos requisitos de memoria, es invariante al cambio de escala diagonal de los gradientes y es adecuado para problemas que son grandes en términos de datos y/ o parámetros.

Dado que las NN tienen capacidad para sobreajustar los datos, se ha considerado detener la optimización con base en el desempeño obtenido en un conjunto independiente llamado de validación[9] [BK19]. Sin embargo, esta práctica, a priori aceptada, ha sido discutida: se ha encontrado un régimen en el que entrenar más tiempo invierte el sobreajuste (doble descenso del gradiente) [BHMM19]. En la fig. 2.7 se muestra el esquema clásico, mientras que en la fig. 2.8 el doble descenso del gradiente. En el esquema clásico, la optimización o entrenamiento se detenía en un punto en el que el error de validación comenzaba a aumentar. En oposición, en los modelos sobredimensionados se observa el fenómeno del doble descenso, donde el entrenamiento, una vez pasado el punto en cual el esquema clásico se detenía, el error de validación comienza a crecer para luego descender (umbral de interpolación). Es decir, se observa que un mayor entrenamiento revierte el sobreajuste. Actualmente, es el esquema del doble descenso el deseado.

La comprensión completa de los mecanismos detrás del doble descenso en las DNN es aún una pregunta abierta importante [Nak19].

En el Anexo A se presentan otros tipos de entrenamiento no mencionados en este capítulo que son de interés del estado del arte actual.

[9]En la sección 3.2.1 se retoma la definición y función de estos conjuntos.

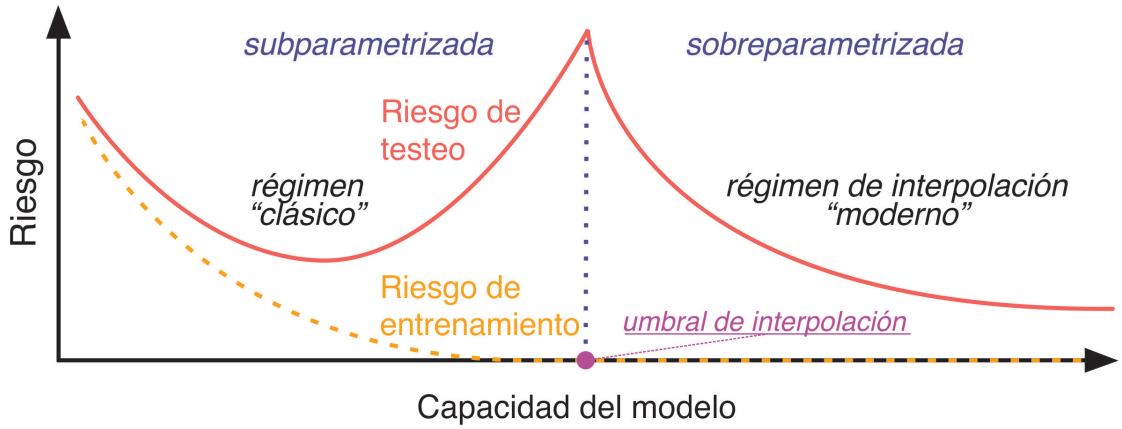


Figura 2.8: Esquema del doble descenso

2.5. Metodología para la construcción de una red neuronal feedforward

La metodología actual para la construcción de una red neuronal feedforward sigue una secuencia de acciones. La primera de ellas, consiste en la definición del problema, este paso repercute en cómo deben ser realizadas las consecuentes.

Consecuentemente, debe iniciarse el preprocesamiento de los datos. En este paso, se preparan los datos que se utilizarán para entrenar la red neuronal. Generalmente, esto incluye tareas como la normalización de los datos, la eliminación de valores atípicos, y la selección de las características relevantes.

Luego, se debe decidir que arquitectura de la red debe utilizarse, esta decisiones suelen estar relacionadas con la definición del problema. Tradicionalmente, esto implica la elección del número de capas y el número de unidades en cada capa, así como la selección de una función de activación adecuada para cada capa.

Previo al entrenamiento, es usual realizar una serie de ajustes adicionales a la arquitectura de la red o a los hiperparámetros para mejorar su desempeño. Este procedimiento implica -idealmente- la búsqueda y configuración óptima de factores críticos como la tasa de aprendizaje, el número de capas y unidades, y otros parámetros relevantes para maximizar el rendimiento del modelo. Esta tarea puede llevarse a cabo de forma manual, donde un experto es quien ajusta

estos parámetros basándose en su conocimiento y experiencia, o de manera automática con técnicas que exploran de forma sistemática el espacio de hiperparámetros en busca de una buena combinación.

Realizadas las etapas anteriores, podemos dar comienzo al entrenamiento de la red. Esto es llevado a cabo por un algoritmo de optimización.

Por último debe realizarse una evaluación del desempeño de la red: Una vez que la red ha sido entrenada, se evalúa su desempeño en términos de la precisión o el error. Si el desempeño de la red neuronal no es satisfactorio, hay varias acciones que se pueden tomar para intentar mejorarlo. Realizar un ajuste de los hiperparametros pueden afectar significativamente el desempeño de la red.

Si el problema persiste después de ajustar los hiperparámetros, se puede considerar cambiar la arquitectura de la red, por ejemplo, aumentando el número de capas o cambiando la función de activación.

En algunos casos, una red neuronal puede tener un desempeño deficiente debido a la falta de datos suficientes para entrenar la red. Por tanto, adquirir más datos y utilizarlos para entrenar la red puede mejorar significativamente su desempeño, aunque esto no siempre es factible.

De esta forma, se puede describir el procedimiento de construcción de un modelo de predicción como no secuencial, dado que usualmente puede volver a etapas anteriores si el desempeño no fue acorde a lo esperado. El proceso de mejorar el desempeño de una red neuronal puede ser un proceso iterativo y requiere una combinación de diferentes técnicas para obtener los mejores resultados.

2.6. Redes neuronales profundas y base de datos pequeñas: estado del arte

La dependencia de grandes volúmenes de datos es uno de los problemas más relevantes del DL para el reconocimiento de las relaciones latentes. Un conjunto de entrenamiento insuficiente es un problema inevitable en algunas áreas de la ciencia, debido a una recopilación de datos compleja o costosa. Por lo tanto, cuando el tamaño de la base de datos se reduce, es conveniente

contar con herramientas y conocimientos específicos sobre el modelado de ANN. A continuación se hará referencia al estado del arte correspondiente.

En [BD20] para el caso de problemas de clasificación de imágenes, mostraron que el uso de una función de pérdida diferente a la usual para ese tipo de problemas (coseno en lugar de entropía cruzada) produjo mejores resultados con amplios márgenes. Los autores atribuyen este beneficio principalmente a un efecto de regularización presente en la norma L2 involucrado en la función de pérdida de mencionada. Las estrategias de regularización están diseñadas explícitamente para reducir el error de prueba, tal vez a expensas de un mayor error de entrenamiento. La regularización en las NN toma muchas formas. Una de ellas es añadir una penalización a la función de pérdida, la norma L2 es uno de estos modos de penalización.

Una alternativa a la escasez de datos es, naturalmente, pensar en ampliar el conjunto. A continuación reportamos diferentes métodos para aumentar el tamaño del conjunto con medios artificiales, sin la recolección de nuevas muestras.

Una de estas opciones es el uso de autocodificadores variacionales para ampliar el conjunto de datos (ver sección A.1). Otra alternativa para la generación de datos es el uso de modelos generativos adversos (GAN: *Generative Adversial Models*) propuesto originalmente por [GPAM$^+$14]. En esta arquitectura, se entrenan simultáneamente dos modelos: un modelo generativo G y un modelo discriminativo D. Luego del entrenamiento, el modelo discriminativo es descartado y el modelo generativo es consultado con entradas aleatorias que se utilizan para generar nuevas muestras de datos. Estos modelos fueron utilizados en el ámbito de reconocimiento de imágenes. En la fig. 2.9 elaborada según se presento en [RMC16] se puede observar un ejemplo de la capacidad de generación de estos modelos que utilizan aritmética vectorial para conceptos visuales. Esta propiedad de los modelos generativos fue demostrada por primera vez en este trabajo. Un ejemplo: el vector(“hombre con anteojos”) - vector(“hombre sin anteojos”) + vector(“mujer sin anteojos”) produce vector(“mujer con anteojos”).

Aunque estos modelos permiten la generación de nuevos datos, se plantea una paradoja: el uso de ambos modelos generativos requiere una gran cantidad de datos para ser generados fielmente [CA21]. En [CA21] utilizan una técnica para la construcción de estos modelos para problemas de clasificación. En la técnica utilizada se propone una forma de saltar la limitación

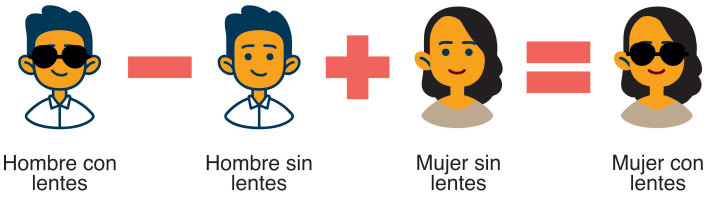


Figura 2.9: Ejemplo de aritmética vectorial presentada

de la cantidad de datos mostrando sus resultados sobre los conocidos conjuntos de caracteres MNIST y EMNIST.

Alternativamente a la generación artificial, existe la posibilidad de utilizar datos o modelos diseñados para resolver otras tareas que mejoren el desempeño del problema original. En la transferencia de aprendizaje (TL: Transfer learning) un modelo desarrollado para una tarea se utiliza como punto de partida para otro modelo destinado a otro problema [GBC16]. Esta técnica recurre a la experiencia pasada de la tarea de origen para mejorar el desempeño en una tarea de destino, ya sea para acelerar el aprendizaje como para mitigar la falta de datos [HAMS21]. Otro incentivo para su uso, en ciertos problemas de aplicación como visión por computadoras o procesamiento del lenguaje natural, es la reducción del uso de recursos necesarios para desarrollar nuevos modelos que son costosos en términos de tiempo de procesamiento y capacidad de cómputo.

Otra posibilidad es contar con algoritmos menos sensibles a la cantidad de observaciones. El campo del meta aprendizaje, “aprender a aprender”, ha experimentado un aumento del interés en los últimos años. Aparece por primera vez en la literatura en trabajos simultáneos, separados e independientes, de [Sch87] y [HP96]. El meta aprendizaje tiene como objetivo mejorar el algoritmo de aprendizaje en sí. Este paradigma permite abordar muchos de los desafíos convencionales del DL, incluido la escasez de datos [HAMS21].

Una de las aplicaciones más importantes del meta aprendizaje es el aprendizaje de pocas oportunidades (FSL: *Few-Shot Learning*) [Pen20]. El FSL se torna particularmente complejo cuando el entrenamiento de modelos grandes con conjuntos de datos pequeños conduce a un sobreajuste, incluso a la no convergencia. Estos métodos entrenan algoritmos que permiten que redes profundas aprendan con éxito en pequeños conjuntos de datos [HAMS21]. Varios de los

aportes que se presentan en esta tesis pueden ser considerados meta aprendizaje.

El tamaño de los conjuntos de datos ha ido en aumento con el tiempo. Esta tendencia se debe al impulso de la sociedad a su digitalización [GBC16]. A su vez, la capacidad de la infraestructura, tanto de hardware como de software específico para el desarrollo de NN profundas, ha mejorado con el paso del tiempo [GBC16].

Las NN son conocidas por su necesidad de grandes volúmenes de datos [BMR+20]. Como consecuencia, el DL ha cobrado mayor importancia en la medida en que se dispone de conjuntos de datos de mayor tamaño. Otras áreas del conocimiento[10] han tenido que lidiar con la situación en la que el tamaño de la experiencia es pequeña.

La maldición de la dimensionalidad (del inglés, *curse of dimensionality*), introducida por primera vez por [Bel61], si bien no constituye en sí una definición de base de datos pequeña, da precisiones matemáticas de la naturaleza del problema. Se establece que el número de muestras necesarias para estimar una función arbitraria con un nivel dado de precisión crece exponencialmente con respecto al número de variables de entrada (es decir, dimensionalidad de la función). En un caso particular presentado por [HTFF17], con 10 dimensiones se necesitan el 80 % de los rangos de cada coordenada para capturar el 10 % de los datos. Asimismo, se presentan las ecuaciones correspondientes que permiten hacer estas estimaciones para otros escenarios.

En otros tipos de problemas -distintos de los de regresión- algunos autores han dado definiciones alternativas de un conjunto de datos pequeños. Por ejemplo, es el caso de [BD20] para un problema de clasificación de imágenes dónde se indica que se trata de un conjunto de datos pequeño cuando se tiene menos de cien imágenes por clase.

En el caso de las arquitecturas DNNM, la definición se relaciona con distintos aspectos de la naturaleza del problema. La relación entre la cantidad de observaciones (m) en el conjunto de datos y la cantidad de parámetros del modelo fue establecida a través de la experiencia como 100 a 1 observaciones por parámetro [Whi89].

[10]Por ejemplo, el término micronumerosidad, definido por el económetra Goldberger [Gol91], refiere a un problema específico que ocurre cuando tenemos una cantidad de observaciones limitada y simultáneamente se dan ciertas condiciones matemáticas. El problema se presenta cuando existe una correlación lineal entre alguna o todas las variables de entrada del modelo. Para este problema, la econometría suele barajar distintas opciones: incorporar nuevas observaciones, usar otros tipos de modelos e incluso la eliminación del modelo de las variables correlacionadas [SGBIGG18].

A los efectos de esta tesis se propone una definición de una base de datos pequeña. En líneas generales, se presume un conjunto de datos pequeño cuando la cantidad de observaciones disponibles es menor a la cantidad de parámetros que requiere el mínimo modelo posible de cierta arquitectura. Formalmente, decimos que estamos ante una base de datos pequeña dado un número m de observaciones pertenecientes a $\mathbb{R}$ y sea p_{min} la cantidad de parámetros del modelo mínimo para dicho conjunto de datos. Si se cumple que:

$$p_{min} > \frac{m}{c} \tag{2.7}$$

para una constante c dependiente de la arquitectura, $c > 0$ ($c \in \mathbb{N}$). Tomando $c = 100$ en función de lo propuesto por [Whi89], la ec. 2.7 queda establecida para este caso como la inecuación 2.8:

$$p_{min} > \frac{m}{100} \tag{2.8}$$

Siendo este el indicador de que nos encontramos ante un problema de base de datos pequeña: la relación entre el número de parámetros del modelo mínimo y la cardinalidad del conjunto de observaciones.

La definición propuesta deberá ser revisada minuciosamente en trabajos futuros, esta supone una relación implícita entre la cantidad de parámetros y su desempeño. Se entiende que la definición propuesta es necesaria aunque perfectible. Un sólido contra argumento está dado por el hecho de que una NN sobredimensionada generaliza en lugar de sobre ajustar. En [FC18] mostraron que una red de gran tamaño que luego fue reducida mediante la eliminación de muchos de sus pesos pudo alcanzar un mejor desempeño que la misma red inicializada aleatoriamente y luego entrenada, mostrando la relevancia de los valores iniciales de los pesos. Esto sugiere que la cantidad de parámetros no está relacionada directamente con su desempeño, siendo condición necesaria pero no suficiente.

Otro argumento relacionado con el sobredimensionamiento de las NN, es que el fenómeno del doble descenso del gradiente se observa en modelos sobre parametrizados [Nak19]. Incluso se encontró que mayor cantidad de observaciones puede conducir a un menor desempeño.

El sobredimensionamiento lleva un aumento significativo del tiempo de cómputo y los requerimientos de memoria. Podemos pensar que estamos aquí en presencia de un compromiso en la toma de decisiones. Sin embargo, es una falacia suponer que con mayor tiempo de entrenamiento se puede compensar un menor tamaño del modelo. Esto recuerda a la argumentación "*ad antiquitatem*" en la que el transcurso del tiempo es un criterio de verdad.

Dado que la definición propuesta no considera el punto inicial de cuál parte el entrenamiento de las NN y que recientes trabajos [FC18] han mostrado la relevancia del punto inicial, la definición propuesta en base a la cantidad de parámetros no da relevancia al valor primigenio de estos. Por tanto, es de esperar que los modelos sean sobredimensionados con el objetivo de maximizar la probabilidad de contener un punto inicial favorable, llamado por los autores ticket de lotería (LT: "Lottery Ticket").

2.7. Conjuntos de datos tabulares pequeños en problemas de regresión

El desafío de construir modelos para conjuntos de datos pequeños es mayor cuando estos conjuntos tienen una representación tabular, ya que los problemas que requieren de datos tabulares son actualmente una frontera para la investigación en DL [GRB22]. Si bien el DL ha permitido un progreso en los conjuntos de datos de texto e imágenes, su superioridad en los datos tabulares no está clara. En conjuntos de datos homogéneos, las DNN han mostrado repetidamente un excelente rendimiento y, por tanto, han sido ampliamente adoptadas. Los datos tabulares heterogéneos son la forma más utilizada, esencial para numerosas aplicaciones críticas y de gran exigencia computacional; no obstante, la adaptación a datos tabulares para tareas de inferencia o generación de datos sigue siendo un reto [BLS+22].

En [GOV22] concluyen que los modelos basados en árboles (bosques aleatorios[11] y máquinas de potenciación del gradiente[12]) superan a los modelos MLP para datos tabulares en conjuntos

[11]El modelo de bosque aleatorio es una forma de ensamble de árboles de decisión en el que se toman medidas para asegurar la diversidad del conjunto de los mismos reduciendo así la varianza [RN10]. Estos modelos pueden ser utilizados para problemas de clasificación y regresión.

[12]Las máquinas de potenciación del gradiente (del inglés, *gradient boosting machine*). Estos modelos utilizan el descenso de gradiente para minimizar los parámetros de un modelo: se calcula la pérdida y se actualizan los parámetros en la dirección de menor pérdida [RN10]. Sin embargo, no se actualizan los parámetros del modelo

de datos de tamaño mediano (aprox. 10.000 observaciones). Asimismo, [BLS+22] han realizado estudios comparativos que indican que los algoritmos basados en ensambles de bosques de árboles superan a los modelos de DL en tareas de aprendizaje supervisado, lo que sugiere que el progreso de la investigación en DL para datos tabulares se está estancando. Todavía no hay precisiones del motivo por el cual estos métodos superan al DL.

En [BLS+22] especulan con los motivos del limitado éxito del DL en este tipo de datos. El primero de ellos, refiere al preprocesamiento de datos tabulares: muchos de los desafíos para DNN en datos tabulares están relacionados con la heterogeneidad de los datos (por ejemplo, atributos categóricos y/ o continuos). Por lo tanto, algunas soluciones de DL los transforman en una representación homogénea más adecuada para las NN, como es el caso de los embeddings. En algunos casos la sobrecarga adicional es pequeña, y estas transformaciones pueden aumentar considerablemente el rendimiento.

Aún no está claro por qué para DL este tipo de datos resulta un desafío. En [GOV22] explican la superioridad de los modelos basados en árboles por múltiples situaciones. La primera de ellas es que las arquitecturas similares a MLP presentan debilidades a los atributos no informativos. Se muestran experimentos en los que la eliminación de las características no informativas reduce la brecha del rendimiento entre los MLP y otros modelos, mientras que agregar características no informativas amplía la brecha. Una hipótesis de por qué las MLP se ven perjudicadas por las características poco informativas estaría relacionada con la propiedad del método de aprendizaje de ser rotacionalmente invariable en el sentido propuesto por [Ng04]. Este concepto de rotación invariable supone que, ante un cambio de orden de los atributos, la distribución de las predicciones es la misma. En términos prácticos, para eliminar las características no informativas, un algoritmo invariante a la rotación tiene que encontrar primero el orden original -de existir ese orden- y luego seleccionar los atributos menos informativos. En el caso de las imágenes este orden es explícito, lo que no ocurre con los datos tabulares. En [GOV22] se propone un experimento en el que la eliminación de los atributos no informativos de cada conjunto de datos previo a la rotación disminuye el rendimiento de todos los modelos. Los autores concluyen que el hecho de que tipos

existente, sino los del siguiente árbol. El popular paquete XGBOOST (del inglés, *eXtreme Gradient Boosting*) se usa habitualmente tanto para aplicaciones a gran escala en la industria (para problemas con miles de millones de ejemplos) como para los ganadores de competencias en ciencia de datos.

diferentes de *embeddings* (en español, incrustaciones) parezcan mejorar el rendimiento sugiere que la presencia de ellos disminuye los efectos de la invariancia rotacional, lo que resulta clave para estas mejoras. El uso de *embeddings* consiste en representar un tipo de dato en un vector numérico, lo que ha permitido una mejora significativa en algunas tareas. Por ejemplo, *Word2vec* -propuesto por [MSC+13]- es un método sencillo para implementar *embeddings* de palabras.

Otra hipótesis de trabajo de estos mismos autores es que las funciones de destino en los conjuntos de datos tabulares no son suaves y las MLP tienen dificultades para adaptarse a estas funciones irregulares en comparación con los modelos basados en árboles.

En apoyo a esta hipótesis, puede observarse que otros autores llegan a resultados similares. Por ejemplo, [GRB22] concluyen que en comparación con los NNMP, las arquitecturas profundas tipo transformador[13] Se proponen dos: una codificación lineal de valores escalares y *embeddings* basados en activaciones periódicas. Estas propuestas transforman un valor escalar en una función real.

Los experimentos demuestran que estos agregados no solo son beneficiosas para los transformadores, sino también para otros métodos: las DNNM son competitivas con los transformadores cuando se entrenan con ellos. En algunos casos, los modelos de DL superan los modelos basados en máquinas de potenciación del gradiente para problemas de regresión y clasificación.

Sin embargo, es importante considerar que se enfocaron en conjuntos de datos de mediana y gran escala. Además, la evaluación muestra un sesgo explícito hacia problemas afines a estas máquinas. Otra advertencia que resulta relevante es que el experimento de control también podría haber incorporado el uso de los métodos propuestos, por ejemplo la combinación entre máquinas de potenciación del gradiente y la codificación lineal de los valores escalares. Así, los autores han demostrado que el uso de *embeddings* para características numéricas es un aspecto importante que merece ser estudiado con mayor detenimiento para datos tabulares con DL.

Para datos tabulares, las máquinas de vectores de soporte (del inglés: *Support Vector Machine*), los bosques aleatorios y el aumento de gradiente son las técnicas que presentan el mejor desempeño.

[13] Un transformador es un modelo de DL que adopta el mecanismo de autoatención, ponderando diferencialmente la importancia de los datos de entrada. Los transformadores fueron presentados por [VSP+17] y han mostrado un sólido desempeño en problemas de datos tabulares. Estos autores incorporan en el uso de modelos existentes métodos para datos tabulares.

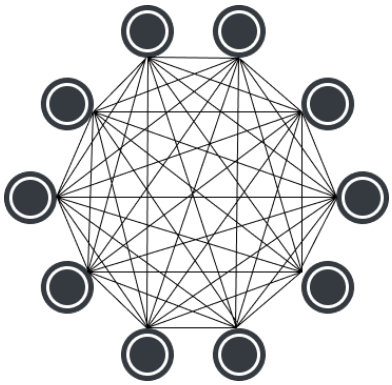


Figura 2.10: Esquema de arquitectura de red clásica Hopfield

En la red clásica de Hopfield [Hop82] cada unidad está conectada con las restantes. En la figura 2.10 se muestra un esquema de esta arquitectura dónde se observa la conexión. En [SGBNH22] se presento "Hopular", una arquitectura DL para conjuntos de datos de tamaño pequeño y mediano, en la que cada capa está equipada con redes Hopfield, basadas en las propuestas de [RSL$^+$21] y [WSP$^+$20]. La novedad de esta propuesta es que cada capa puede acceder directamente a la entrada original, así como a todo el conjunto de entrenamiento a través de los datos almacenados en las redes de Hopfield. Por lo tanto, Hopular puede actualizar paso a paso su modelo y la predicción resultante en cada capa, con algoritmos de aprendizaje iterativo estándar.

En experimentos con conjuntos de datos tabulares de tamaño pequeño con menos de 1000 muestras, Hopular supera a máquinas de potenciación del gradiente[14] , bosques aleatorios y SVM. En experimentos con datos tabulares medianos con alrededor de 10.000 observaciones, Hopular supera a varios algoritmos de máquinas de potenciación del gradiente[15]. Con base en tales resultados, Hopular podría ser considerada como alternativa a estos métodos para datos tabulares pequeños.

Otro modelo propuesto por [RDM22] es una arquitectura especializada que provee interpretabilidad de los resultados sobre conjuntos de datos tabulares y de imágenes. Una de las evaluaciones comparativas realizadas fue un caso de estudio sobre datos tabulares pequeños en problemas de regresión (menos de 10.000 observaciones). En coincidencia con otros autores, este experimento indicó superioridad de XGBoost.

[14]Se menciona a Gradient Boosting.
[15]Se menciona en particular a XGBoost, CatBoost y LightGBM.

Según estudios comparativos realizados por [BLS+22], se ha demostrado que los algoritmos basados en ensambles de bosques de árboles son más efectivos en tareas de aprendizaje supervisado en datos tabulares que los modelos de DL. Esto sugiere que la investigación en DL para este tipo de datos podría estar llegando a un punto en el que no se produce avance.

El DL en datos tabulares aún presenta desafíos. Se observa que no se ha trabajado lo suficiente en la definición de evaluaciones comparativas estandarizadas (del inglés, *benchmarks*) para problemas de regresión para datos tabulares [GOV22], como sí ocurre en el caso de la clasificación para el aprendizaje de pocas oportunidades [HAMS21]. En [GOV22] se propone una batería de conjuntos de datos con ciertas limitaciones: conjuntos de tamaño medio (más de 3.000 observaciones, sus atributos numéricos y heterogéneos). Por lo que esta cuestión seguirá en debate.

Otro desafío es el TL para datos tabulares mencionado por [BLS+22]. Mientras que TL se utiliza con éxito en aplicaciones de visión artificial (del inglés, *computer vision*) y aplicaciones de procesamiento del lenguaje natural, no hay formas eficientes y aceptadas de realizar el aprendizaje de transferencia para datos tabulares. Por lo tanto, una pregunta de investigación general puede ser cómo compartir conocimientos entre múltiples conjuntos de datos tabulares relacionados de forma eficiente.

Por otro lado, el aumento de datos para datos tabulares es todavía un reto. Aunque existen algunas técnicas de aumento de datos para datos tabulares, por ejemplo, SMOTE-NC [CBHK02], los modelos simples no logran capturar la estructura de dependencia de los datos. Sin embargo, las mejoras reportadas son marginales. Por lo tanto, es necesario trabajar en el futuro para encontrar transformaciones aleatorias sencillas pero eficaces para mejorar el entrenamiento sobre los conjuntos tabulares.

2.8. Conclusiones

No hay consenso respecto a cuando una base de datos es pequeña. La discusión parece estar centrada en la cantidad de parámetros de un modelo, penalizando modelos de gran volumen, relativizando otros aspectos que hacen al desempeño, por ejemplo el punto inicial.

Un conjunto de entrenamiento insuficiente, por otro lado, es un problema ineludible en

algunos dominios especiales. Cuando la recopilación de datos es compleja y costosa, se hace difícil construir un conjunto de datos a gran escala y de alta calidad.

Por estos motivos, es conveniente contar con herramientas y conocimiento específico sobre el planteo y modelado de NN cuando se cuenta con una base de datos reducida en su tamaño.

La construcción de NN con datos pequeños es un desafío, y este desafío resulta más complejo si se trata de datos tabulares. El estado del arte actual demuestra que los asuntos tratados en esta tesis son relevantes. Por lo tanto, es innovadora en tanto que afronta la intersección de estos dos desafíos.

En el Anexo A.1 se presentan otros tipos de NN y de entrenamiento relevantes para el estado del arte actual. Además, en el Anexo A.2 se detallan los recursos utilizados en esta tesis.

3. La división del conjunto de datos tabulares pequeños

Contenidos de este capítulo

En este capítulo se explora el supuesto de la división aleatoria del conjunto de datos -usual en un gran volumen de datos- con respecto a su validez en contextos de bases de datos pequeñas para problemas de regresión. Se argumenta sobre la conveniencia de esta práctica en el contexto de bases de datos pequeñas.

Se estudian los diferentes aspectos de la división de un pequeño conjunto de bases de datos tabular y cómo afecta las propiedades del MR y se introduce un algoritmo que pueda controlar el mecanismo de división siguiendo el criterio planteado por una función de distancia.

3.1. Introducción

"I suppose is tempting, if the only tool you have is a hammer,
to treat everything as if it were a nail" [1]
"The Psychology of Science"
– A. Maslow[2]

Grandes volúmenes de datos son necesarios para utilizar NN [BMR+20]. En consecuencia, si se trata de mejorar el desempeño de un sistema que tiene embebido un componente de DL [Doz16], puede seguirse alguno de estos enfoques:

- mejorar la estructura del modelo;
- mejorar la inicialización del modelo, garantizando que en etapas tempranas el gradiente tenga propiedades benéficas o construyendo grandes espacios de escasez o raleza (del inglés: *sparsity*);
- La mayoría de los algoritmos de aprendizaje involucran alguna forma de optimización, y es mediante el uso de un algoritmo de optimización más eficiente, que es posible encontrar mejores soluciones en un tiempo razonable.

Este capítulo se enfoca en la mejora de la inicialización del modelo. Se estudian los efectos que produce mejorar en particular una de estas condiciones: cómo la división de los datos es realizada. Se propone un algoritmo para optimizar esta división.

El objetivo principal de este capítulo es hacer notar que al dividir el conjunto de datos de manera aleatoria, surgen notables disparidades entre las distribuciones de los subconjuntos. Asimismo, se exploran los siguientes objetivos específicos propuestos con anterioridad:

- Se explora la validez del método que se aplica comúnmente a grandes volúmenes respecto a cómo se realiza la división del conjunto de datos en el caso de bases de datos tabulares pequeñas para problemas de regresión.

[1] "Supongo que es tentador pensar que, si la única herramienta que tienes es un martillo, puedes tratar cualquier cosa como si fuera un clavo."
[2] [Mas66]

- Se identifican las propiedades estadísticas de los conjuntos de datos que permitan prever un mejor desempeño. Se observa cómo afecta la varianza de la variable objetivo a la evaluación del modelo.

3.2. El error medio cuadrado

El MSE es una función que cumple diversos roles en el ámbito del DL. Uno de estos roles es el criterio de evaluación de modelos en problemas de regresión. Otro de los roles usuales es la función de pérdida, siendo una de las más frecuentemente utilizadas [HTFF17]. Se estudiará la composición de MSE en la construcción de MR en el contexto de bases de datos pequeñas.

Sea $y = f(x)$ un problema de regresión y sea $\widehat{y} = \widehat{f}(x)$ un MR que intenta resolver el problema planteado. Entonces se define el error medio cuadrado esperado, o simplemente error medio esperado (EEE) como:

$$\begin{aligned} EEE(\widehat{f}) &= E((\widehat{f}(x) - y)^2) \\ &= Var(\widehat{f}(x) - y) + E^2(\widehat{f}(x) - y) \\ &= Var(\widehat{f}(x)) + Sesgo^2(\widehat{f}(x)) \end{aligned} \tag{3.1}$$

El cálculo del MSE permite realizar una estimación sobre el valor EEE, ya que la esperanza por definición debe calcularse sobre todo el conjunto de los valores posibles, siendo este conjunto tal vez de cardinalidad infinita.

Sin embargo, en un escenario más complejo $y = f(x) + \varepsilon$, siendo ε una variable aleatoria de distribución normal[3] donde la $E(\varepsilon) = 0$ y $Var(\varepsilon) = \sigma^2$ [HTFF17]. Luego tenemos que:

[3]Se asume que $E(\varepsilon) = 0$, ya que, si fuera un valor diferente de cero, el problema $y = f(x) + \varepsilon$ podría ser reescrito como $y = f(x) + (\varepsilon - \mu + \mu) = y = f'(x) + (\varepsilon - \mu)$. Ahora la $E(\varepsilon - \mu) = 0$, donde $f'(x) = f(x) + \mu$.

En términos prácticos, de aquí se podría deducir que el MR va a incorporar dentro de su estimación el valor medio del epsilon.

Si fuera el caso de que la distribución de ε fuera diferente a una distribución normal. Podemos reescribir $y = f'(x) + (\varepsilon - \mu)$ como $y = f'(x) + (\varepsilon - \mu)/\sigma$, donde $f'(x) = (1/\sigma) * f'(x)$.

Por el Teorema del Límite Central [WMMY17] $(\varepsilon - \mu)/\sigma$ tendrá una distribución normal. Ocurre que de forma análoga al planteo anterior, se podría deducir que el MR incorpora dentro de su estimación el desvío de la media.

$$
\begin{aligned}
Var(y) &= Var(f(x) + \varepsilon) \\
&= Var(f(x)) + Var(\varepsilon) && [f(x) \text{e} \varepsilon \text{ son independientes}] \\
&= Var(\varepsilon) = \sigma^2 && [f(x) \text{ no es una variable aleatoria}]
\end{aligned}
$$

Este nuevo escenario redefine la Ec. 3.1, lo que implica que:

$$
\begin{aligned}
EEE(\widehat{f}) &= E((\widehat{f}(x) - y)^2) \\
&= Var(\widehat{f}(x) - y) + E^2(\widehat{f}(x) - y) \\
&= \ \sigma^2 + Var(\widehat{f}(x)) + Sesgo^2(\widehat{f}(x)) \\
&= Error\ irreducible + Var(\widehat{f}(x)) + Sesgo^2(\widehat{f}(x))
\end{aligned}
\tag{3.2}
$$

El sesgo es la diferencia entre la estimación promedio del modelo y el valor que se intenta aproximar. Un modelo con alto sesgo en valor absoluto muestra una tendencia a la sobreestimación o subestimación general del mismo según sea el signo. Un sesgo bajo sugiere una mejor aproximación sobre el proceso generador de los datos[4] (DGP: *Data Generating Process*). En contraposición, un alto sesgo sugiere un abordaje más simple al DGP.

Es de esperar que un MR tenga cierta varianza. Los MR que tienen una alta variación están fuertemente influidos por las características específicas de los datos utilizados para su elaboración.

De la ec. 3.2, es deseable que el MR tenga una varianza pequeña, como así también un pequeño sesgo. En términos prácticos, es posible estimar $Sesgo^2(\widehat{f}(x))$ a través del MBE. Un modelo con bajo sesgo y poca varianza indica que el modelo logra estimar con precisión el DGP. Que, en última instancia, es el ideal.

A continuación, se evidencia lo planteado en la ec. 3.2. Por un lado, en la evaluación de los MR y, por otro, en la construcción de los mismos cuando esta función es empleada como función de pérdida.

[4]Los datos de entrenamiento y prueba se generan mediante una distribución de probabilidad sobre conjuntos de datos denominada proceso generador de datos. Se asume que esta misma distribución es utilizada para generar cada observación del conjunto de entrenamiento y/ o validación o testeo.

3.2.1. Distorsiones en la evaluación de los modelos

En situaciones donde los datos abundan, el mejor enfoque es dividir los datos de forma aleatoria en tres partes: entrenamiento, validación y testeo [HTFF17, GBC16]. Cada uno de estos conjuntos persigue un objetivo diferente durante la construcción del MR. El conjunto de entrenamiento $\{\tau\}$ tiene la finalidad de construir el MR, mientras que la utilidad del conjunto de validación $\{\kappa\}$ es estimar el desempeño de diferentes modelos para elegir el mejor entre ellos. Por último, el propósito del conjunto de testeo $\{\phi\}$ es brindar una estimación del error de predicción sobre nuevos datos (error de generalización). Como se explicó en el Capítulo 2, idealmente estos datos deberían ser guardados con celo hasta el final del desarrollo del modelo [HTFF17, Bis06].

Sin embargo, en un contexto de bases de datos pequeñas, dividir los datos de esta forma no es recomendable. En primer lugar, debido a que las diferencias en las varianzas de los conjuntos de entrenamiento y testeo podrían tener un impacto en la evaluación de desempeño de los modelos cuando se utiliza MSE como medida de su rendimiento. Esta distorsión provendría del primer término que es independiente del MR $\widehat{f}$ indicado en la ec. 3.2 como *error irreducible*. Esto quiere decir que la distorsión no depende del método a través del cual fue construido el MR, sea una red neuronal, un bosque aleatorio u otro.

En un contexto donde $n \to \infty$ es esperable que la $Var(y_\tau) \ \to \ Var(y)$ por la Ley de los Grandes Números. Pero cuando tenemos una cardinalidad limitada, la convergencia empírica puede no satisfacerse. Tampoco podría asegurarse que $Var(y_\tau)$ y $Var(y_\phi)$ tiendan al mismo valor. De esta forma se agregaría una distorsión adicional a la evaluación del modelo, aún cuando el método de regresión pueda estimar con precisión el DGP. Este argumento sugiere la necesidad de considerar cómo es llevada a cabo la división de los datos en un contexto de bases de datos pequeña con el objetivo de no introducir un sesgo de selección[5]. Por lo que realizarla al azar podría tener sus riesgos, cosa que no es usual cuando se tienen grandes volúmenes de datos.

[5]Se refiere a una distorsión de un análisis estadístico, debido al método de recolección de muestras.

3.2.2. Una herramienta para la selección

Por lo expuesto, la partición de los conjuntos de datos sería recomendable realizarla aleatoriamente, lo que sugiere la necesidad de un instrumento que permita particionar los conjuntos de datos.

Además de la división aleatoria, otro enfoque de división de datos usual en escenarios de base de datos pequeños se conoce como validación cruzada sobre k-partes (del inglés: *k-fold cross validation*). Sin embargo, esta técnica tiene como contrapartida un mayor coste computacional: el procedimiento se basa en la idea de repetir el entrenamiento y el testeo k veces en subconjuntos disjuntos obtenidos al azar.

El error de generalización[6] puede ser estimado promediando el error de testeo a través de los k intentos. En el intento i, el i-ésimo subconjunto de los datos es usado como conjunto de testeo y el resto de los datos es utilizado como conjunto de entrenamiento.

Se propone encontrar una partición del conjunto de datos que minimice las diferencias de las varianzas: esto es un problema de optimización combinatoria que tiene una complejidad NP-Completo. Esta categoría de complejidad puede ser resuelta por una máquina de Turing no determinista en tiempo polinomial [AB09]. Tal clase de problemas contiene numerosas instancias de búsqueda y optimización, para los que se desea saber si hay una cierta solución o si existe una mejor solución que las conocidas. En la sección siguiente se presenta un algoritmo evolutivo[7] que permite resolver el problema.

3.3. Un algoritmo evolutivo para particionar el conjunto de datos

Inspirados en la Teoría de la Evolución Natural, estos algoritmos guían la búsqueda estocástica de la solución haciendo evolucionar a un conjunto de estructuras de modo iterativo. La idea subyacente común de estas técnicas es que, dada una población de individuos dentro de algún

[6] Además, esta técnica posee la contrapartida adicional de subestimar el error de generalización [HTFF17]. Por lo que, a pesar de la falta de datos, podría ser una decisión utilizar validación cruzada.

[7] Más adelante en el texto, se discriminará entre la familia de algoritmos evolutivos, resultando el algoritmo propuesto en un algoritmo genético.

entorno que tiene recursos limitados, la competencia por esos recursos provocaría la selección natural (supervivencia del más apto). Esto a su vez promovería un aumento en la aptitud de la población. En la ejecución de un algoritmo evolutivo, la población de individuos representa un conjunto de soluciones a un problema.

Dada una función a maximizar, podemos crear aleatoriamente un conjunto de soluciones candidatas, es decir, elementos del dominio de la función. Entonces, es posible aplicar una función de calidad a estos como una medida abstracta de aptitud: cuanto mayor sea este valor, será mejor la solución. Sobre la base de estos valores de aptitud, algunos de los mejores candidatos serán elegidos para recrear la próxima generación. Esto se hace aplicando sobre los individuos un mecanismo de cruza y/ o mutación. La cruza o recombinación es un operador que se aplica a dos o más candidatos seleccionados (los llamados padres), produciendo nuevos candidatos.

La mutación se aplica a un candidato y da como resultado un nuevo candidato con el que se incorpora nueva información. Por lo tanto, ejecutar las operaciones de cruza y mutación en los padres conduce a la creación de un conjunto de nuevos candidatos (la descendencia). Luego, estos tendrán la oportunidad de competir según su aptitud (y posiblemente por su antigüedad) con los mayores por un lugar en la próxima generación. Este proceso de selección favorece a los mejores individuos. Este favoritismo puede determinarse dada su aptitud, calculada por una función específica (del inglés: *fitness*). Cada uno de los ciclos de mutación y selección se define como generación. Idealmente, luego de un cierto número de generaciones se espera que el mejor individuo o campeón esté razonablemente próximo a la solución buscada. En el Anexo B.1 se muestra con mayor detalle la mecánica indicada a través de un algoritmo.

Siguiendo el criterio de [ES15], en función de la forma que tomen los individuos, el algoritmo tendrá diferentes nombres[8]. Se llamará algoritmo genético (GA: Genetic Algorithm) si los individuos tienen la forma de cadenas (secuencias de caracteres) sobre un alfabeto finito. En caso de que los individuos sean vectores de valores reales, se llamarán estrategias de evolución. Y se llamará programación evolutiva clásica en caso de ser máquinas de estados finitos, ó programación genética en el caso de que sean árboles de decisión.

[8]La terminología contemporánea denota el campo por computación evolutiva. Los algoritmos involucrados son algoritmos evolutivos y se consideran como subáreas pertenecientes a las variantes de algoritmos: la programación evolutiva, las estrategias de evolución, los algoritmos genéticos y la programación genética.

Se propone un algoritmo genético que pueda ser utilizado como una herramienta de partición del conjunto de datos. Este algoritmo recibe como parámetros de entrada un conjunto de datos y una lista de porcentajes que indica la proporción del tamaño de cada partición deseada, en términos de la cantidad de subconjuntos y su tamaño. En este caso, partición se define como un subconjunto perteneciente al conjunto de partes del conjunto de datos original. La intersección entre estos subconjuntos es vacía. Adicionalmente, todos los elementos pertenecen a algún subconjunto.

El conjunto de datos es un arreglo de observaciones numéricas ó vectores de R^n de longitud m. La partición se representa como una secuencia de longitud arbitraria l tal que cada elemento e satisface la inecuación $0 \ \leq e \leq 1$ y la secuencia debe sumar uno ($\sum e_i = 1$). El resultado devuelto será una secuencia s de longitud m en la que cada elemento tomará un valor perteneciente al rango $[0,\ l-1]$. Luego, cada elemento de la secuencia determina la partición a la que pertenece esa observación, satisfaciendo la cardinalidad indicada para ese subconjunto.

La población inicial se construye como una permutación aleatoria de la secuencia ordenada de índices que respeta la proporción indicada por l. Cada individuo[9] está representado por una lista de longitud m.

El algoritmo incluye operadores de mutación y cruza. Podría producirse un desbalance en la proporción de los conjuntos al momento de cruzarlos, lo que daría lugar a individuos que se distancian de la cantidad de observaciones que debe tener cada conjunto. Estos individuos que no respetan el criterio planteado se definen como no factibles.

El cruce monopunto elige una posición aleatoria para ambas secuencias, uniendo la parte anterior de un individuo con la parte posterior de otro. En la fig. 3.1 se puede apreciar de manera esquemática cómo el operador de cruce monopunto produce nuevos individuos. La posición que se utiliza para crear los nuevos participantes es elegida de forma aleatoria con una distribución uniforme para cada posición, por lo tanto la probabilidad de elegir cada posición es equitativa.

En el caso de la cruza, se planteó un cruce monopunto definido ad hoc, de tal forma que incluya la reparación de los individuos cuando no se mantienen las proporciones de las particiones

[9]Dos conceptos relevantes relacionados al individuo son los de genotipo y fenotipo. El fenotipo corresponde a una característica del individuo, el genotipo es una representación del fenotipo. Un ejemplo clásico es el color de ojos: mientras que el color verde es un fenotipo su representación genómica es el fenotipo.

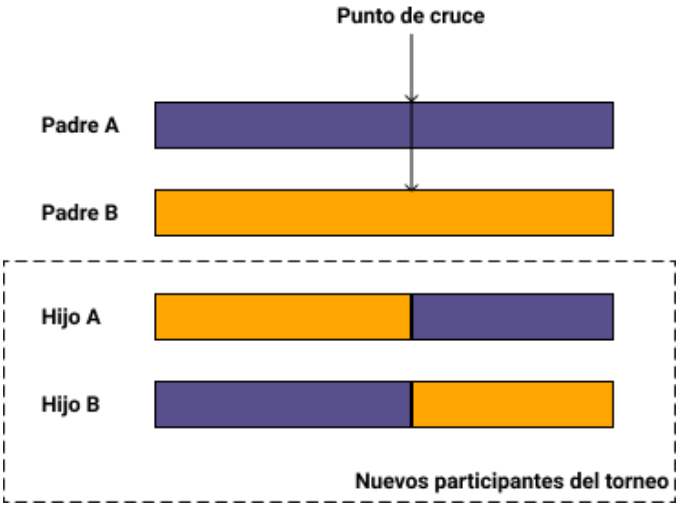


Figura 3.1: Operador de cruce monopunto

(individuo no factible). La reparación restablece el balance, eligiendo aleatoriamente posiciones entre las partes que restablezcan el equilibrio. Así, no se producirán individuos no factibles en este procedimiento.

Una vez que el individuo fue seleccionado para su mutación, el operador genético se resuelve mediante el intercambio del contenido de dos posiciones elegidas eventualmente. La probabilidad de mutar un gen[10] es proporcional a la longitud del individuo, es de esperar no más de una mutación por cada uno de ellos.

El operador de selección utilizado se conoce como *tournament* o torneo. La mecánica de este operador para la siguiente generación se realiza mediante una comparación de dos individuos: se selecciona el mejor individuo entre los elegidos al azar para participar del torneo. Esto se repite la cantidad de veces necesaria hasta lograr una nueva población produciendo el reemplazo total de la anterior. El menos apto de los individuos de cada generación perderá todos los torneos. Además, el algoritmo propuesto implementa elitismo, por lo que el campeón -el mejor individuo- se mantiene dentro de la población en todo momento. De esta manera, la mejor solución encontrada no se pierde de generación a generación.

El algoritmo cuenta con dos funciones de aptitud. La primera está basada en la distancia[11]

[10]Aquí es relevante distinguir gen y alelo. Un alelo es uno de los posibles valores que puede tener un gen. Este concepto es análogo al de una variable y el alelo al de su valor. En el algoritmo propuesto, los alelos pueden tomar algún valor dentro del rango $[0, l-1]$ y cada posición de la lista representa un gen.

[11]Una distancia o métrica es una función matemática para la cual pueda probarse las siguientes propiedades: no negatividad, simetría y desigualdad triangular.

de Wassertein o de movimiento de tierra. Esta distancia se define como una función entre distribuciones de probabilidades en un espacio métrico M dado. Puede interpretarse como la cantidad mínima de "trabajo" requerido para transformar una pila de tierra en otra. "Trabajo" se mide como el peso de una distribución que debe moverse multiplicado por la distancia en la que tiene que ser desplazado.

La aplicación de esta distancia en el DL tiene antecedentes en [ACB17], en la que fue utilizada para estabilizar el entrenamiento de modelos generativos [GPAM+14]. Teóricamente[12], esta función se define sobre dos distribuciones u y v:

$$l_1(u, v) = inf_{\pi \in \Gamma(u,v))} \int_{\mathbb{R}\times\mathbb{R}} |x - 1| d\pi(x, y) \tag{3.3}$$

Donde $\Gamma(u, v)$ es el conjunto de distribuciones de probabilidad sobre cuyas marginales u y v sobre el primer y segundo factor respectivamente.

En el segundo caso, la función de aptitud es calculada con base en una definición de compromiso de la distancia de Mahalanobis[13] propuesta por [Cua19].

$$(\mu_1 - \mu_2)' \left[\frac{1}{2}(\Sigma_1 + \Sigma_2)\right] (\mu_1 - \mu_2) \tag{3.4}$$

Donde μ_i representa el vector de medias y Σ_i la matriz de varianzas-covarianzas de la partición dada y el conjunto de datos íntegro. En la implementación, en lugar de los parámetros poblacionales se emplearon los estadísticos muestrales. Para este cálculo, se tomaron en consideración las medias de cada uno de los atributos y la matriz de varianza-covarianza que relaciona los valores de la distribución conjunta de las variables del conjunto de observaciones.

[12] En términos prácticos su método de cálculo puede variar. Para esta tesis se empleó la implementación de la librería SciPy (v. 1.7.1). El material de referencia se puede consultar en [RTC17] y en
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html

[13] Esta decisión está fundada en que la Distancia de Mahalanobis supone que las matrices de covarianzas son iguales, y tal cosa no puede asegurarse. En efecto, el objetivo del algoritmo es lograr que las matrices de covarianzas sean lo más próximas unas a otras.
Un cambio de escala de una variable X_j es una transformación $Y_j = aX_j$; donde a es una constante. Esta distancia tiene algunas propiedades de interés: tiene en cuenta la correlación entre las variables y es invariante por transformaciones lineales no singulares de las variables, en particular cambios de escala. Por lo cual una normalización de los datos no cambiaría la distancia entre ellos.

3.4. Resultados y discusión

El algoritmo fue implementado[14] en el lenguaje Python usando la librería DEAP (acrónimo de *Distributed Evolutionary Algorithms in Python*) en su versión v.1.0. Para evaluar el desempeño del algoritmo se diseñó un conjunto de experimentos comparándolo con una búsqueda aleatoria.

En las siguientes cuadros se resumen los resultados obtenidos de la evaluación[15]. La medida presentada es la razón entre el algoritmo propuesto y la búsqueda aleatoria.

La evaluación fue realizada en dos tipos de DGP diferentes que se caracterizan por tener distintas curtosis. El primero de estos, una distribución uniforme. El segundo, una distribución normal donde la media y la varianza fueron calculadas aleatoriamente para cada atributo de la evaluación.

En teoría de probabilidad y estadística, la curtosis es una medida de la "cola" de la distribución de probabilidad de una variable aleatoria de valor real. La curtosis de cualquier distribución normal es 3, una distribución con un valor inferior producirá menos valores atípicos y menos extremos que la distribución normal.

Se define como observación atípica aquella para la cual alguno de sus atributos es un valor atípico siguiendo el criterio utilizado por el diagrama de cajas y bigotes [DKLM05]. En la fig. 3.2, se considera un valor atípico el que se encuentra 1,5 veces la distancia del rango intercuartílico (IQR, del inglés *interquartile range*). Vemos la caja, cuyo borde superior corresponde al valor del tercer cuartil (Q3) y el borde inferior al primer cuartil (Q1). Dentro de la caja se encuentran el 50 % de las ocurrencias. Además, se marca con una línea gruesa el valor de la mediana. La altura de la caja es el IQR. Por encima y debajo se ven lo que se conoce como bigotes, estos dos límites son los umbrales para los valores atípicos. La distancia de IQR rango intercuartílico se define como:

$$IQR = Q_3 - Q_1 \tag{3.5}$$

Entonces si el valor q,

[14] Detalles de implementación pueden encontrarse en https://deap.readthedocs.io/en/master

[15] El algoritmo genético fue evaluado con el valor de sus hiperparámetros por defecto. Cantidad de generaciones= 50, Individuos de la población = 30, probabilidad de cruce=0.60

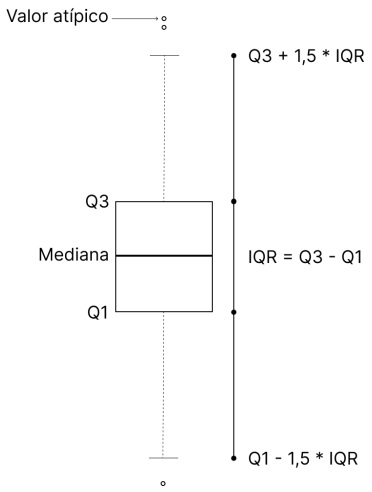


Figura 3.2: Diagrama de cajas

$$q < Q_1 - 1{,}5 * IQR \text{ o } Q_3 - 1{,}5 * IQR > q \tag{3.6}$$

se considera valor atípico.

Para cada tamaño de conjunto de datos y distribución se generaron 36 conjuntos. En las cuadros 3.1 y 3.2 se presentan los resultados para las distancias de Mahalanobis y Wassertein respectivamente, para la división de datos que se haría para una validación cruzada en k partes.

En el caso de el cuadro 3.1 se observan los resultados de la evaluación en forma de razón entre la aptitud de las mejores soluciones reportadas utilizando la función de compromiso de distancia de Mahalanobis. Mientras que el cuadro 3.2 se observan los resultados de la evaluación en forma de razón entre la aptitud de las mejores soluciones reportadas utilizando la distancia de Wassertein. En ambos casos, el máximo beneficio parece estar cuando el tamaño del conjunto se encuentra comprendido en el rango $[10^3, 10^4]$.

En términos generales, se observa un mejor desempeño de los datos uniformemente distribuidos en comparación con los normales, lo que sugiere que la curtosis presente en el conjunto de datos supone alguna dificultad en resolver el problema: la presencia de valores atípicos supone un obstáculo. A su vez, en ambos casos se observa que entre 100 y 1000 observaciones parece encontrarse el punto de máximo beneficio de la utilización del método de división de los datos propuesto.

Cantidad de observaciones	Ratio	
	Uniforme	Normal
100	56,04 % (+/- 9,48 %)	56,90 % (+/- 10,44 %)
1000	68,27 % (+/- 13,78 %)	70,08 % (+/-14,20 %)
10.000	51,01 % (+/- 24,89 %)	48,37 % (+/- 24,68 %)
100.000	31,48 % (+/- 31,39 %)	28,66 % (+/- 24,53 %)

Cuadro 3.1: Partición en k partes realizada con la pseudo distancia de Mahalanobis

Cantidad de observaciones	Ratio	
	Uniforme	Normal
100	25,43 % (+/- 5,67 %)	20,95 % (+/- 5,57 %)
1.000	28,20 % (+/-6,45 %)	23,47 % (+/- 5,41 %)
10.000	17,78 % (+/-8,38 %)	15,92 % (+/- 4,32 %)
100.000	11,22 % (+/-7,80 %)	9,62 % (+/- 5,99 %)

Cuadro 3.2: Partición en k partes realizada con la pseudo distancia de Wassertein

En las cuadros 3.3 y 3.4 se presentan los resultados para las distancia de Mahalanobis y Wassertein respectivamente, para la división de datos que se haría para una validación cruzada (en tres partes y proporciones de 70 %, 15 % y 15 % respectivamente). También muestra que el punto de mayor beneficio se encuentra entre las 1000 y 10000 observaciones para ambas distancias.

En el cuadro 3.3 se observan los resultados de la evaluación en forma de razón entre la aptitud de las mejores soluciones reportadas utilizando la función de compromiso de distancia de Mahalanobis. Por otro lado, en el cuadro 3.4 se observan los resultados de la evaluación en forma de razón entre la aptitud de las mejores soluciones reportadas utilizando la distancia de Wassertein. Otra vez, para ambas distancias, el máximo beneficio parece estar cuando el tamaño del conjunto se encuentra comprendido en el rango $[10^3, 10^4]$.

Es importante notar que a diferencia de lo que ocurre con otras normas, no es posible comparar los resultados de las distancias entre ellas, debido a que no se ha podido encontrar una demostración de la relación de orden que podría existir entre estas distancias.

La distancia de Mahalanobis parece tener un mejor desempeño cuando la división de datos es para validación cruzada, tanto en la distribución normal como uniforme. Esto parece mostrar

Cantidad de observaciones	Ratio	
	Uniforme	Normal
100	59,78 % (+/- 8 ,55 %)	60,17 % (+/- 12,72 %)
1.000	84,74 % (+/- 4,39 %)	80,24 % (+/- 3,30 %)
10.000	70,16 % (+/- 19,27 %)	82,31 % (+/- 9,40 %)
100.000	53,77 % (+/- 28,55 %)	62,16 % (+/- 23,62 %)

Cuadro 3.3: Partición en tres partes realizada con la pseudo distancia de Mahalanobis

Cantidad de observaciones	Ratio	
	Uniforme	Normal
100	26,41 % (+/- 2,60 %)	21,96 % (+/- 3,20 %)
1.000	32,58 % (+/- 4,61 %)	26,89 % (+/- 3,60 %)
10.000	21,89 % (+/- 6,31 %)	16,09 % (+/- 8,47 %)
100.000	15,70 % (+/- 7,09 %)	12,09 % (+/- 7,64 %)

Cuadro 3.4: Partición en tres partes realizada con la pseudo distancia de Wassertein

cierta resistencia a los valores atípicos. Es decir, que de utilizar validación cruzada se recomienda esta distancia.

La distancia de Wassertein tiene un mejor desempeño cuando la distribución de los datos es uniforme. Esto sugiere que en presencia de valores atípicos tendrá un desempeño menor que el esperado, independientemente del mecanismo de división que se emplee.

3.4.1. Tiempos de ejecución

Las corridas se ejecutaron en una máquina Samsung modelo NP270E5E-EXP con un procesador Intel Core i7 con una capacidad de memoria 8 GB DDR3-SDRAM.

En el cuadros 3.5 y 3.6 se presentan los tiempos expresados en segundos de ejecución del algoritmo propuesto teniendo en cuenta la función de distancia empleada y la cardinalidad de los conjuntos tanto en observaciones como en atributos.

Ambas distancias muestran una gran diferencia en cuanto a los tiempos de ejecución. La distancia de Wassertein muestra sensibilidad a la cantidad de observaciones y atributos, mientras que la distancia de Mahalanobis parece mostrar mayor dependencia de la cantidad de observaciones, sin responder a los atributos que no parecen una condición de crecimiento del tiempo de

Tiempo promedio de la búsqueda mediante el algoritmo genético en segundos				
Distancia de Mahalanobis				
Cantidad de atributos	Cantidad de observaciones			
	100	1.000	10.000	100.000
1				
3	5.71	10.56	38.45	352.51
5	5.68	10.53	38.05	324.15
7	5.41	10.46	39.48	348.09
9	5.53	10.45	40.34	343.60
Promedio general	5.58	10.50	39.08	342.09

Cuadro 3.5: Tiempos de ejecución expresados en segundos para la distancia de Mahalanobis

Tiempo promedio de la búsqueda mediante el algoritmo genético en segundos				
Distancia de Wassertein				
Cantidad de atributos	Cantidad de observaciones			
	100	1.000	10.000	100.000
1	5.17	9.00	47.68	555.45
3	6.27	12.17	82.82	984.84
5	7.25	15.82	116.62	1448.77
7	8.20	19.24	147.00	1862.06
9	8.81	22.87	180.59	2320.29
Promedio general	7.14	15.82	114.94	1434.28

Cuadro 3.6: Tiempos de ejecución expresados en segundos para la distancia de Wassertein

ejecución. Se trata, entonces, de un factor a tener en cuenta para sopesar los beneficios de una y otra.

3.5. Conclusiones

En este capítulo se exploró el supuesto de que la división aleatoria del conjunto de datos es la estrategia usual cuando los datos abundan. Sin embargo, se mostraron los riesgos -al menos teóricos- que conlleva realizar la partición del conjunto de datos de forma aleatoria cuando la base de datos es pequeña, ya sea para validación cruzada en k partes o cross-validation.

Quedaron plasmadas las diferencias que se observan cuando la división de datos se realiza de forma aleatoria o controlada. Estas diferencias fueron calculadas mediante dos funciones de distancia de distribuciones de probabilidad (Mahalanobis y Wassertein). Se expusieron las

razones de estas distancias y cómo ellas varían dependiendo de las propiedades estadísticas del conjunto de datos, en particular de la curtosis. Ahora bien, no es prudente afirmar a priori que estas distancias tengan un impacto en la construcción del MR.

El análisis y la experimentación expuestos sugieren un criterio alternativo para definir cuándo una base de datos es pequeña. Recordemos que a los efectos de esta tesis, se presume una base de datos pequeña cuando la cantidad de observaciones disponibles es menor a la cantidad de parámetros que requiere el mínimo modelo posible de una cierta arquitectura. El criterio alternativo se basaría en el mecanismo de la división de los datos, dejando fuera de la discusión los parámetros en cuestión y por ende el MR. Este criterio estaría dado por el punto de inflexión en el cual la Ley de los Grandes Números se vuelve predominante, es decir el punto en el cual es indistinto el beneficio entre la búsqueda aleatoria y la búsqueda guiada por el GA. En términos de la experimentación propuesta, cuando la razón tiende a cero o se vuelve negativa. Asimismo, también se observa que el punto de máximo beneficio se encuentra cuando el tamaño del conjunto de datos tiene entre 10^3 y 10^4 observaciones, independientemente de su curtosis.

3.6. Trabajos futuros

La experimentación se concentró en atributos numéricos de las observaciones. Sin embargo, requieren una mención especial los atributos ordinales o categóricos. El primer reparo refiere a la transformación de las variables categóricas que puede ser necesaria, en función del MR que se requiera. En el caso de las NN, esta transformación debe ser llevada a cabo. Una de estas transformaciones posibles se conoce como representación distribuida (conocida también como *one hot encoding*) [GBC16, Hin86]. Por ejemplo, el atributo altura -con valores posibles como *alto*, *medio* ó *bajo*- sería representado por tres variables binarias -es_alto, es_medio y es_bajo- donde sólo una de ellas puede tomar el valor uno y el resto tendrá el valor cero[16]. Esto lleva a preguntarse si serían necesarias otras funciones de distancia específicas para estos atributos. Como por ejemplo, la distancia de Hamming [Cua19]. Otra alternativa a considerar, es utilizar el

[16]En esta representación, uno de los valores del atributo podría darse por implícito. Es decir, podríamos omitir una de las variables binarias y representarla implícitamente cuando el resto de las variables toma el valor cero. En el ejemplo planteado, podríamos omitir es_bajo y representarlo cuando es_alto = es_medio = 0. De esta forma, se alteraría la cantidad de parámetros.

coeficiente de similaridad definido por [Gow71] para observaciones mixtas, entre otras.

Existen, además, múltiples distancias que podrían ser evaluadas para conjuntos de datos diversos. Como por ejemplo el caso de la distancia de Sokal-Sneath que se encuentra específicamente definida para variables binarias [Cua19].

Por otro lado, la transformación implica un cambio en la relación entre cantidad de observaciones y parámetros del modelo. Esta relación es la que -por la definición dada en Capítulo 2- determina si se trata de una base de datos pequeña o no. Se plantea la duda de que si el conjunto de datos posee variables discretas o categóricas, sea factible alcanzar los mismos niveles de generalización con la misma cantidad de observaciones en el conjunto de datos. Estos interrogantes en torno a los atributos categóricos se estudiarán en trabajos futuros.

Otro de los aspectos a considerar en próximos experimentos es cómo afecta la falta de simetría -momento 3 de una variable aleatoria (en inglés: *skewness*), ya que la experimentación se realizó sobre DGP con distribuciones simétricas.

4. La división controlada del conjunto de datos en NN

Contenidos de este capítulo

En este capítulo se estudian los efectos del control de la división de los datos en la evaluación del modelo cuando se trata de un MR construido con una NN. Se intenta identificar si este control produce una estimación más precisa del error de generalización, para esto el DGP debe ser conocido. Por ese motivo se utilizan generadores artificiales de datos que poseen propiedades estadísticas similares a condiciones de laboratorio.

Luego, se estudia la división controlada de conjuntos de base de datos tabulares pequeños de dominio público, de los que se desconoce su DGP. La pregunta de investigación es: ¿hay diferencia en cuanto al error de testeo entre la división controlada y la aleatoria?

Todos los experimentos de este capítulo se originan en un mismo modelo, evitando así las consecuencias de partir de diferentes puntos iniciales.

4.1. Introducción

En el capítulo 3 se observó que, cuando la división del conjunto de datos es dejada al azar, se producen grandes diferencias entre las distribuciones de los subconjuntos. En este capítulo se estudia los efectos que podrían producir estas diferencias, ya que estos conjuntos se utilizan para construir el MR cuando es una NN.

Recordemos que en situaciones en las que los datos abundan, la manera recomendada de dividirlos es aleatoriamente, independientemente de si se utiliza validación cruzada o un esquema de validación cruzada en k-fold [GBC16, HTFF17]. En un esquema de validación cruzada cada uno de los subconjuntos cumple un rol diferente en la construcción de los modelos. El primero de estos roles es la definición del MR y es llamado de entrenamiento: (τ) [HTFF17]. Mientras que otra de las partes se ocupa de estimar el desempeño de diferentes modelos para elegir el mejor entre ellos, conocido como validación (κ)) [HTFF17]. Por último, la función del subconjunto de testeo ((ϕ)) es la estimación del EEE a través de MSE (error de generalización) [HTFF17].

4.2. La evaluación de los modelos de NN con base de datos pequeñas tabulares

Este capítulo se pregunta si la división controlada produce modelos de NN con mayor poder de generalización. En general, se desconoce el DGP, por lo que el poder de generalización es estimado. Como ya se dijo, el error de testeo es una estimación del poder de generalización de los modelos [HTFF17]. Es este error el que se utiliza para realizar la evaluación de los modelos y guía la decisión respecto de cuál es el que posee un mejor desempeño. Por este motivo, la pregunta original de investigación puede descomponerse en dos fases: cuál es la precisión en la estimación del poder de generalización y cuál de los esquemas produce menores errores de testeo.

Entonces, la división controlada de (X, y) ¿estima con mayor precisión el error de generalización que la división aleatoria? ¿Produce errores de testeo más pequeños que la división aleatoria? Es posible determinar el grado de precisión de esta estimación en ambos esquemas si y sólo si se conoce el DGP.

Se tratará de encontrar respuesta a estas preguntas con el foco puesto en si se presentan diferencias en las etapas tempranas de entrenamiento de las NN. En [FSM20] se exponen los cambios que se producen en NN tomando como referencia dos arquitecturas para el procesamiento de imágenes (ResNet y Cifar10). En la fig. 4.1 se grafica mediante una línea de tiempo lo que ocurre en las etapas tempranas del entrenamiento de las NN.

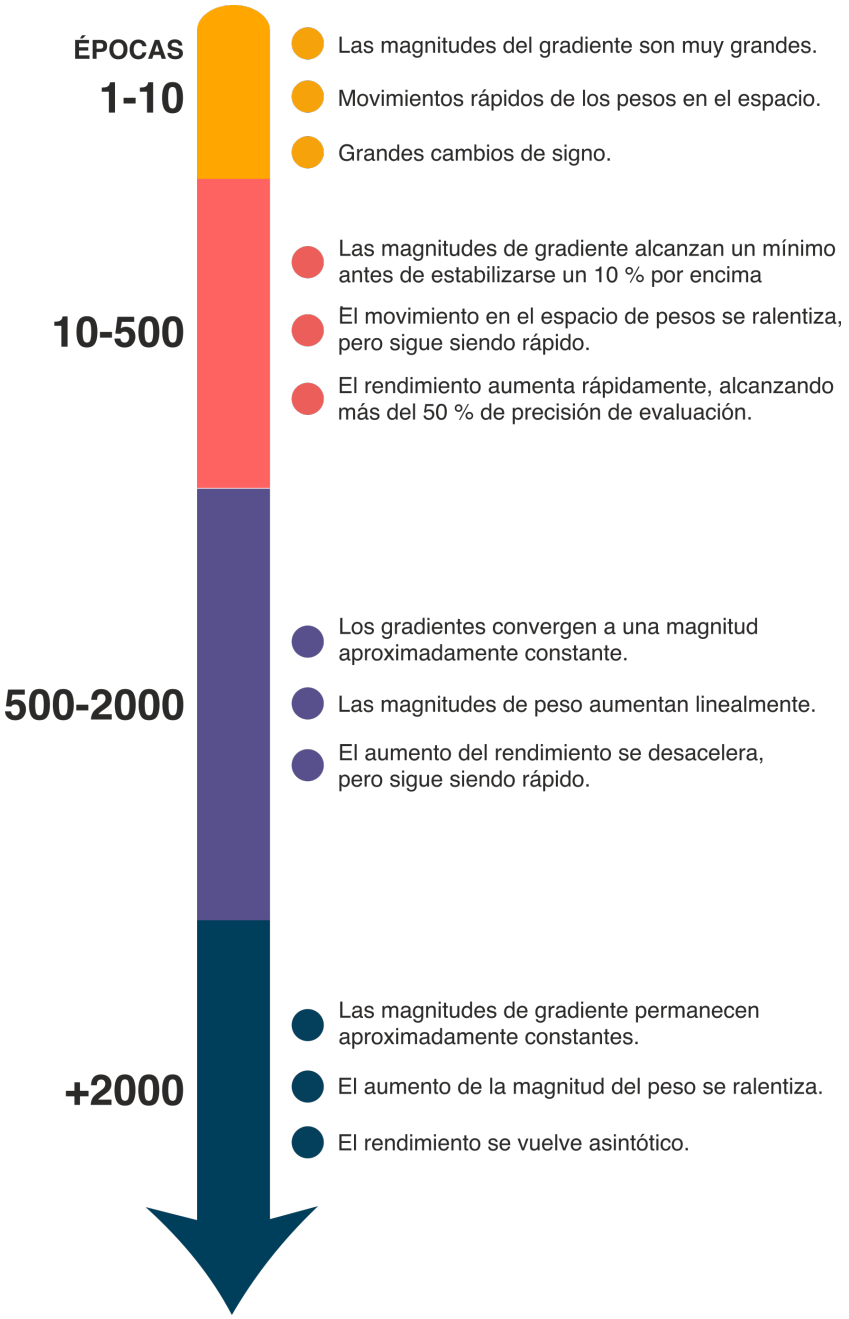


Figura 4.1: Eventos que ocurren en las etapas tempranas del entrenamiento

En las primeras épocas del entrenamiento se suceden cambios que tienen efecto a largo

plazo. Entre la época 10 y la 500 se producen movimientos de los pesos en el espacio, además el desempeño aumenta rápidamente, llegando al 50 % en su precisión de evaluación.

Mientras que en el periodo de la época 500 a 2000, el desempeño comienza a desacelerarse, aunque todavía es rápido, y los gradientes convergen a una magnitud constante. Más allá de la época 2000, los gradientes se mantienen constantes y el desempeño se vuelve asintótico.

4.3. Datos, métodos y materiales

Consideremos los siguientes contextos para la partición del conjunto de datos:

- división aleatoria, siendo esta la línea de base para la comparación,
- división controlada, ya sea en un esquema de validación cruzada o en k partes.

Se plantean los siguientes interrogantes para los distintos contextos:

- Experimento A - La división controlada de (X, y) ¿estima con mayor precisión el error de generalización que la división aleatoria?
- Experimento B - La división controlada de (X, y) ¿produce errores de testeo menores que la división aleatoria?

En consideración de los hallazgos de [FC18], en los que la existencia de tickets de lotería[1] de inicialización (LT: *Lottery Ticket*) en algunas redes hace que el entrenamiento sea particularmente efectivo debido a sus conexiones y pesos iniciales, es que todos los experimentos parten de una misma red inicial, con la intención de neutralizar el problema de punto inicial que esto supone.

La calidad de un modelo predictivo está condicionada críticamente por la configuración de sus hiperparámetros, aunque no se tiene una comprensión acabada de cómo los valores de estos interactúan entre ellos afectando el modelo [LJD+16]. Este proceso de selección de los valores apropiados se denomina ajuste de hiperparámetros. Se define hiperparámetro como una variable que gobierna el proceso de entrenamiento y/ o la selección de la topología de un modelo de NN. Para cada uno de los experimentos se tomaron consideraciones especiales respecto a este asunto.

[1] Este asunto se retoma más adelante (Ver Cap. 5)

En la inferencia estadística, la prueba de hipótesis es un procedimiento que puede determinar si los atributos asumidos en la población estadística son compatibles con los observados en la muestra de población. Mediante esta teoría, se trataron los problemas estadísticos considerando una cierta hipótesis nula H_0 y una hipótesis alternativa H_1, y aplicamos tales parámetros a un cierto número de experimentos.

La forma en la que se reportan los resultados de test de hipótesis de esta tesis es indicando el nivel de significación específico en el que la hipótesis nula es rechazada. Este punto -llamado p-valor- y conocido también como nivel de significación observado, es el más pequeño nivel de significación por el cual H_0 sería rechazada cuando se utiliza un procedimiento de prueba de hipótesis.

Esto quiere decir que una vez que se ha determinado el p-valor, la conclusión a un nivel particular α resulta de comparar ambos:

1. p-valor $\leq \alpha \Rightarrow$ rechazar H_0 al nivel α

2. Caso contrario, no es posible rechazar H_0 al nivel α

Mientras más pequeño es el p-valor, más contradictorios son los datos con la hipótesis nula. Los niveles tradicionales de significación son 0.10, 0.05 y 0.01 [Dev07]. Mientras más estricto sea el criterio con el error, más pequeño deberá ser el nivel de significación. En esta tesis se indica en los resultados el nivel de significación más bajo con el cual se rechaza la hipótesis nula (el p-valor).

En el caso del Experimento A, se definen diferentes escenarios para estas preguntas. Un escenario es la combinación de un DGP, el criterio de división y una función de distancia utilizada para realizar la división controlada del conjunto de datos. Ya que la naturaleza de esta pregunta supone que se conoce el DGP, los conjuntos de datos serán generados artificialmente con la propiedad de ser no lineales: Friedman 1 (ec.4.1) , Friedman 2 (ec.4.2) y Friedman 3 (ec.4.3). En el caso de Friedman 1 ([Fri91][2]; [Bre96][3]) el proceso generador se basa en polinomios y funciones trigonométricas (seno). Mientras que Friedman 2 (F2) implica multiplicaciones e inversas de las entradas. Por último, Friedman 3 (F3) es similar a la anterior, pero además incluye trigonometría (la función arco tangente). Se puede observar que F1 es una combinación lineal de

[2]Según Google Scholar citado en 10.522 referencias al 30/abril/2023

[3]Según Google Scholar citado en 33.046 referencias al 30/abril/2023

valores acotados, por lo que la función tiene un mínimo y un máximo global. En el caso de F2, se implican potencias, raíces e inversas de las entradas que poseen rangos muy dispares. En el caso de $X_2 = 0$ la función no estaría definida, ya que se trata de la raíz de un número negativo. F3 también posee entradas con rangos muy dispares en sus escalas. Además de no estar definido si $X_0 = 0$, dado que se produce una indefinición matemática debido a una división por cero.

Si bien estos conjuntos de datos poseen complejidades y rangos dispares, todos ellos comparten la propiedad de que los atributos de entrada son independientes y uniformemente distribuidos en los rangos definidos. Estas condiciones resultan más favorables cuando se realiza la división del conjunto al azar, dado que por su distribución todos los percentiles son homogéneos.

$$\begin{gathered} F_1(X_0, X_1, X_2, X_3, X_4) = 10 * sin(\pi X_0 X_1) + 20(X_2 - \frac{1}{2})^2 + 10X_3 + 5X_4 \\ 0 \leq X_i \leq 1 \text{ tal que } 0 \leq i \leq 4 \end{gathered} \tag{4.1}$$

$$\begin{gathered} F_2(X_0, X_1, X_2, X_3) = X_0^2 + \sqrt{X_1 X_2 - \frac{1}{(X_1 X_3)^2}} \\ 0 \leq X_0 \leq 100 \\ 40 * \pi \leq X_1 \leq 560 * \pi \\ 0 \leq X_2 \leq 1 \\ 1 \leq X_3 \leq 11 \end{gathered} \tag{4.2}$$

$$\begin{gathered} F_3 = arctan(\frac{X_1 X_2 - \frac{1}{X_1 X_3}}{X_0}) \\ 0 \leq X_0 \leq 100 \\ 40 * \pi \leq X_1 \leq 560 * \pi \\ 0 \leq X_2 \leq 1 \\ 1 \leq X_3 \leq 11 \end{gathered} \tag{4.3}$$

En este caso, no se realiza HPO, lo que permite cuantificar la labilidad de las NN con relación al punto inicial. Este asunto ha sido discutido por [FC18], en particular la existencia de puntos iniciales favorables.

En el caso del experimento B, se utilizaron otros conjuntos de datos de los cuales se desconocen los DGP. En contraposición a los conjuntos del experimento A, estos tienen la propiedad de que sus atributos no se encuentran uniformemente distribuidos y presentan correlaciones entre ellos.

En particular, se trata de considerar si no hay diferencias entre ambos esquemas de división del conjunto de datos en términos del poder de generalización. El desconocimiento del DGP solo permite estimar el poder de generalización mediante el error de testeo [HTFF17].

No es la intención del trabajo entrar en detalles específicos sobre los problemas de aplicación expresados por estos conjuntos, por lo que se presentan sucintamente los datos utilizados haciendo uso de un gráfico: un mapa de calor dado por la matriz de correlación. En el Anexo C.2 se ofrece una descripción con mayor detalle de los datos correspondientes al experimento. La correlación de Pearson es independiente de la escala de medida de las variables y tiene rango definido en $[-1, 1]$. De manera menos formal, podemos definir este coeficiente como un índice que permite medir el grado de relación lineal entre las variables[4]. Si es próximo a 0, no hay relación lineal, pero esto no significa necesariamente que las variables sean independientes: es posible que haya una relación no lineal entre ellas. Si $r = -1$ ó $r = 1$, entonces hay una correlación perfecta, ya sea negativa ó positiva.

En el apéndice C.2.1 se presenta un conjunto de datos de regresión múltiple que se relaciona con la eficiencia energética presentada en [TX12], compuesto por 768 observaciones de 8 atributos de entrada y 2 de salida. En este caso particular, las relaciones perfectas se dan en los pares variables: (Y_1, Y_2), (X_4, X_5), (X_1, X_2). Otras relaciones relevantes próximas a los extremos son: (X_1, Y_1), (X_1, Y_2), (X_1, X_4), (X_1, X_5), (X_2, X_4), (X_4, Y_1) y (X_4, Y_2).

En el apéndice C.2.2 se presenta un conjunto de datos relacionado a la hidrodinámica de los yates [OLG07] compuesto por los resultados de 308 experimentos realizados. La estimación de la resistencia residual de los yates a vela es útil para estimar la potencia propulsora requerida. Las relaciones positivas más relevantes se dan en (Y_1,X_6), (X_2,Y_1) y (X_2,X_6). El par (X_2,X_4) muestra cierta correlación negativa.

En el apéndice C.2.3, se presenta de conjunto de datos que posee 517 observaciones con 13

[4]Siempre y cuando estas variables sean cuantitativas y continuas.

atributos, de los cuales uno es de salida, que fuera presentado por [CM07]. Dentro de los atributos de entrada, cuatro de ellos corresponden al tipo categórico, por este motivo serán excluidos del conjunto de datos, describiendo solamente los numéricos. En este conjunto se observa que los atributos están notablemente correlacionados. El sentido común dicta que cuanto más seco y ventoso es un ambiente, más propicio es para la propagación del fuego. Esto también se observa en este conjunto de datos marcado por las relaciones negativas entre (wind, area_log) y (RH, area_log).

En el último en el apéndice C.2.4, se presenta un conjunto de datos referido al asentamiento del concreto [Yeh07]. Este conjunto de datos está compuesto por 103 observaciones con 7 atributos de entrada y 3 de salida. En este caso se trata de un problema de regresión múltiple. Se observan dos correlaciones intensas: una positiva (entre SP y *Slag*) y la otra negativa (entre *Water* y *Coarse Aggr.*)

En este experimento B, se explora qué ocurre en las etapas tempranas del entrenamiento teniendo en cuenta los diferentes esquemas de división de los datos. Recientes estudios han mostrado que aspectos importantes del aprendizaje de las NN ocurren en etapas tempranas del entrenamiento [FSM20]. En este experimento el foco está puesto en el error de testeo en las primeras etapas entre los esquemas a fin de descubrir si hay alguna diferencia sustancial entre ellas.

Además, se realiza HPO que es compartida con todos los modelos. La optimización se realizó con un algoritmo propio de HPO que se presentará en el Capítulo 5.

4.4. Resultados y discusiones

4.4.1. Experimento A: La división controlada de (X, y) ¿estima con mayor precisión el error de generalización que la división aleatoria?

En general no se observa diferencia entre los dos esquemas de división en estos conjuntos de datos.

En este experimento se entrenaron 480 redes por 1000 épocas cada una. Podemos considerar 6 cohortes, cada una de ellas compuesta por el par DGP y esquema de entrenamiento (validación o validación cruzada en k partes).

Para cada cohorte se entrenaron NN con la misma arquitectura de base sobre conjuntos de datos de 1000 observaciones. Teniendo en consideración que se posee el DGP, es posible controlar el error irreducible asociado. A su vez, dentro de cada cohorte se generaron distintos conjuntos de datos con una varianza del error irreducible con los siguientes porcentajes: 0.00 %, y 5.00 %.

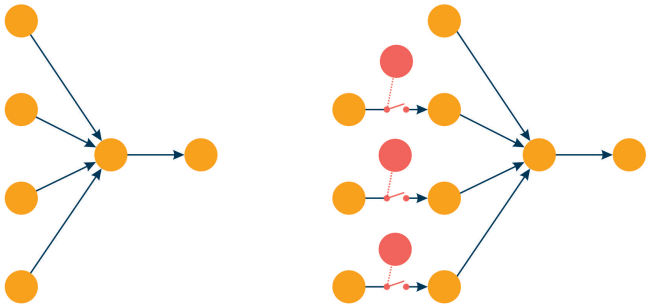
Todas las NN comparten la misma arquitectura compuesta por cinco capas: tres de ellas ocultas tienen 700, 500 y 300 unidades respectivamente. Esta arquitectura poseía un total de 11.742 parámetros (de los cuales 8.132 son entrenables). La función de activación es la lineal rectificada con filtraciones [GBC16] (Ver 2.3.1).

Recordemos que por definición se trata de una base de datos pequeña si nos encontramos ante una relación determinada entre el número de pesos (parámetros) y la cardinalidad del conjunto de observaciones. La ec. 4.4, por la definición planteada se trata de un problema de base de datos pequeña.

$$p_{min} > \frac{m}{100} = p_{min} > \frac{1000}{100} = p_{min} > 10 \tag{4.4}$$

Un problema central en el ML es cómo hacer que un algoritmo funcione bien, no solo con los datos de entrenamiento, sino también en las nuevas entradas. Como se mencionó en el Cap. 2, muchas estrategias utilizadas están diseñadas explícitamente para reducir el error de testeo, posiblemente a expensas de un mayor error de entrenamiento [HTFF17]. Estas estrategias se conocen colectivamente como regularización. En este sentido se utilizó deserción (del inglés: *dropout*), presentada por [SHK$^+$14]. La deserción entrena el conjunto que consta de todas las subredes que pueden formarse, eliminando unidades sin salida de una red base subyacente, como se ilustra en la fig.4.2 tomada de [SHK$^+$14]. En la mayoría de las NN actuales, basadas en una serie de transformaciones y no linealidades, podemos eliminar efectivamente una unidad de una red multiplicando su valor de salida por cero.

Para la optimización fue utilizado el algoritmo SGD, este y sus variantes son probablemente



(a) red clasica (b) red con deserción

Figura 4.2: Esquema arquitectónico de la deserción

los más utilizados. Un parámetro crucial de este algoritmo es la LR. Otro parámetro de gran interés es el momento (del inglés: *momentum*). Para estos parámetros se dieron los valores recomendados por defecto respectivamente: 0.0001 y 0.9. Además, no se empleó aceleración del gradiente de Nesterov [SMDH13]. Por último, el tamaño del lote de observaciones utilizado para la actualización de los pesos (en inglés, conocido como *batch size*) es de 32 observaciones.

El experimento fue escrito en Python, sobre la base de librerías Tensorflow[5] y ejecutado en Colab[6].

Escenario		Error porcentual promedio entre de testeo y generalización			
Conjunto de datos	Criterio	División aleatoria		División controlada	
F1	Validación cruzada	-1.25 %	(+ / -) 8.37 %	-1.43 %	(+ / -) 8.36 %
	K-fold	0.42 %	(+ / -) 10.04 %	-1.31 %	(+ / -) 9.32 %
F2	Validación cruzada	-4.58 %	(+ / -) 16.55 %	-8.05 %	(+ / -) 12.36 %
	K-fold	-4.82 %	(+ / -) 19.77 %	4.47 %	(+ / -) 24.25 %
F3	Validación cruzada	-0.11 %	(+ / -) 14.22 %	0.29 %	(+ / -) 12.98 %
	K-fold	2.25 %	(+ / -) 28.54 %	-4.45 %	(+ / -) 23.62 %

Cuadro 4.1: Error porcentual promedio del error de testeo y generalización sin ruido irreducible

En los cuadros 4.1 y 4.2, se puede observar el error porcentual promedio entre el error de testeo y generalización. En la cuadra 4.1 se reportan los errores correspondientes al DGP sin error irreducible. Mientras que la cuadra 4.2 indica estos errores con el máximo del error irreducible

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

[6]Colab permite escribir y ejecutar código Python en un navegador. Es recomendado por Google para el ML, análisis de datos y proyectos educativos. Técnicamente, Colab es un servicio de script alojado en Jupyter que no requiere configuración y brinda acceso gratuito a los recursos informáticos, incluidas las GPU.

Escenario		Error porcentual promedio entre de testeo y generalización			
Conjunto de datos	Criterio	División aleatoria		División controlada	
F1	Validación cruzada	-7.96 %	(+ / -) 10.33 %	-8.35 %	(+ / -) 10.46 %
	K-fold	-11.88 %	(+ / -) 10.10 %	-4.54 %	(+ / -) 10.81 %
F2	Validación cruzada	-9.01 %	(+ / -) 12.45 %	-6.34 %	(+ / -) 14.09 %
	K-fold	-6.11 %	(+ / -) 38.50 %	-8.53 %	(+ / -) 35.39 %
F3	Validación cruzada	-10.39 %	(+ / -) 16.01 %	-13.46 %	(+ / -) 9.63 %
	K-fold	-7.49 %	(+ / -) 22.55 %	-8.06 %	(+ / -) 18.99 %

Cuadro 4.2: Error porcentual promedio del error de testeo y generalización con ruido irreducible del 5 %.

(5 %).

En general, no se observa diferencia entre los dos esquemas de división en estos conjuntos de datos. Recordemos que ellos tienen propiedades estadísticas muy particulares y poco usuales fuera de laboratorio. Como es de esperar, a medida que crece el error irreducible aumenta el error de generalización. Este error fue calculado sobre una muestra de 30.000 observaciones del DGP correspondiente, por lo que se puede considerar una estimación muy precisa de este, que debería ser calculada sobre todo el dominio.

Retomando a la pregunta de investigación: se realizaron tests de hipótesis en relación con el error porcentual entre el testeo y generalización. En el cuadro 4.3 se reportan los p-valores para las cohortes en cuestión, cada cohorte incluye todos los valores de error irreducible evaluados. Estos tests indican la medida en la que un esquema distorsiona la evaluación del modelo. Salvo una cohorte, estos esquemas no rechazan la hipótesis nula de igualdad de medias. Por lo que, en principio, se podría afirmar que para estos conjuntos de datos -en estas condiciones- es indistinto un esquema u otro.

Se evaluaron tests de hipótesis de igualdad de medias e igualdad de varianzas. En el Anexo J se reportan los test de hipótesis comparativos de la división controlada o aleatoria.

4.4.2. Experimento B: La división controlada de (X, y) ¿produce menores errores de testeo que la división aleatoria?

La respuesta es que en algunas ocasiones la división controlada puede producir redes con errores menores (tanto MSE como MAE). Por lo que, en estos casos podría concluirse que

Escenario		$H_0 : \mu_{aleatoria} = \mu_{controlada}$ $H_1 : \mu_{aleatoria} \neq \mu_{controlada}$	
Conjunto de datos	Criterio	MAE	MSE
F1	Validación cruzada	0.65	0.81
	K-fold	0.41	0.34
F2	Validación cruzada	0.09	0.12
	K-fold	0.60	0.29
F3	Validación cruzada	0.91	0.24
	K-fold	0.99	0.69

Cuadro 4.3: Test de hipotesis para las poblaciones apareadas: contraste de division aleatoria con division controlada

la división controlada presentaría mayor poder de generalización en las etapas tempranas del entrenamiento.

Este experimento compara la división aleatoria con la división controlada en las distintas etapas tempranas del entrenamiento para los conjuntos de datos antes mencionados.

Recordemos que este experimento se realiza sobre conjuntos de datos que tienen propiedades estadísticas distintas del experimento anterior. Además, el DGP es desconocido. Por lo que no es posible calcular el error de generalización y las conclusiones que se obtengan se basarán en el error de testeo.

A diferencia del experimento anterior, los hiperparámetros fueron definidos mediante un algoritmo de HPO que será presentado en el Cap 5. De esta forma, para cada conjunto de datos todas las redes comparten los mismos hiperparámetros que fueron definidos como óptimos en un proceso previo.

Para cada uno de los conjuntos de datos presentados se entrenaron 150 redes para distintas combinaciones de función de distancia para división controlada y cantidad de épocas de entrenamiento (500, 1000 y 3500): 25 redes por cohorte.

En los cuadros 4.4, 4.5, 4.6 y 4.7 se observan los test de hipótesis sobre dos muestras no independientes de NN donde se compara, con respecto al mismo punto inicial, el desempeño de la división controlada en contraste con la división aleatoria, reportándose el p-valor asociado para los distintos conjuntos de datos. Se analizan las posibilidades en relación con el rendimiento medio y la igualdad de las varianzas para los conjuntos de datos reportados anteriormente.

Se colorean las celdas en virtud del valor de significancia mínima que rechaza la hipótesis

nula. El gris oscuro corresponde a 0.01, el gris medio a 0.05 y el gris claro a 0.10. En contraste, el color blanco no rechaza la H_0. En este cuadro se compara el esquema de división aleatoria con la controlada.

Etapa	$H_0 : \mu_{aleatoria} > \mu_{controlada}$ $H_1 : \mu_{aleatoria} \leq \mu_{controlada}$		$H_0 : \mu_{aleatoria} = \mu_{controlada}$ $H_1 : \mu_{aleatoria} \neq \mu_{controlada}$		$H_0 : \mu_{aleatoria} < \mu_{controlada}$ $H_1 : \mu_{aleatoria} \geq \mu_{controlada}$	
	MAE	MSE	MAE	MSE	MAE	MSE
Entrenamiento	0.8359	0.8478	0.3282	0.3044	0.1641	0.1522
Validación	0.5605	0.6118	0.8789	0.7764	0.4395	0.3882
Testeo	0.4922	0.4901	0.9844	0.9803	0.5078	0.5099

Cuadro 4.4: Test de hipótesis para las poblaciones apareadas: contraste de división aleatoria con división controlada para el conjunto de datos ENB.

Etapa	$H_0 : \mu_{aleatoria} > \mu_{controlada}$ $H_1 : \mu_{aleatoria} \leq \mu_{controlada}$		$H_0 : \mu_{aleatoria} = \mu_{controlada}$ $H_1 : \mu_{aleatoria} \neq \mu_{controlada}$		$H_0 : \mu_{aleatoria} < \mu_{controlada}$ $H_1 : \mu_{aleatoria} \geq \mu_{controlada}$	
	MAE	MSE	MAE	MSE	MAE	MSE
Entrenamiento	0.0041	0.0030	0.0082	0.0061	0.9959	0.9970
Validación	0.7312	0.9547	0.5377	0.0905	0.2688	0.0453
Testeo	0.9973	0.9922	0.0055	0.0156	0.0027	0.0078

Cuadro 4.5: Test de hipótesis para las poblaciones apareadas: contraste de división aleatoria con división controlada para el conjunto de datos Forest Fires.

Etapa	$H_0 : \mu_{aleatoria} > \mu_{controlada}$ $H_1 : \mu_{aleatoria} \leq \mu_{controlada}$		$H_0 : \mu_{aleatoria} = \mu_{controlada}$ $H_1 : \mu_{aleatoria} \neq \mu_{controlada}$		$H_0 : \mu_{aleatoria} < \mu_{controlada}$ $H_1 : \mu_{aleatoria} \geq \mu_{controlada}$	
	MAE	MSE	MAE	MSE	MAE	MSE
Entrenamiento	0.0135	0.0265	0.0269	0.0530	0.9865	0.9735
Validación	0.9835	0.9834	0.0330	0.0331	0.0165	0.0166
Testeo	0.9910	0.9886	0.0180	0.0229	0.0090	0.0114

Cuadro 4.6: Test de hipótesis para las poblaciones apareadas: contraste de división aleatoria con división controlada para el conjunto de datos Slump.

La primera columna indica la etapa del entrenamiento del modelo. La segunda ofrece precisiones sobre la etapa y la medida en las cuales se calculó el test de hipótesis. Los siguientes pares de columnas[7] refieren a la inferencia estadística sobre las medias.

[7]Es importante notar que el valor de algunas celdas del cuadro carece de importancia. Por ejemplo, en el caso de que la hipótesis nula para igualdad de medias no sea rechazada (valores reportados en columna 4 y 5), no tendría sentido reportar los test de cola (columnas 2,3, 6 y 7).

Etapa	$H_0 : \mu_{aleatoria} > \mu_{controlada}$ $H_1 : \mu_{aleatoria} \leq \mu_{controlada}$		$H_0 : \mu_{aleatoria} = \mu_{controlada}$ $H_1 : \mu_{aleatoria} \neq \mu_{controlada}$		$H_0 : \mu_{aleatoria} < \mu_{controlada}$ $H_1 : \mu_{aleatoria} \geq \mu_{controlada}$	
	MAE	MSE	MAE	MSE	MAE	MSE
Entrenamiento	0.4967	0.3053	0.9934	0.6107	0.5033	0.6947
Validación	0.7478	0.8153	0.5043	0.3694	0.2522	0.1847
Testeo	0.9034	0.8874	0.1932	0.2253	0.0966	0.1126

Cuadro 4.7: Test de hipótesis para las poblaciones apareadas: contraste de división aleatoria con división controlada para el conjunto de datos Yacht.

En principio, se observan dos patrones de comportamiento: aquellos conjuntos en los cuales la división controlada no reporta beneficios y aquellos en los sí. Dicho de otra forma, en los no se rechaza la hipótesis nula contra los que sí lo hacen.

En el caso de Yacht y ENB no se puede rechazar la hipótesis nula. Parece darse el caso de que las medias de MSE y MAE del testeo son iguales. Por lo que es posible concluir que no hay diferencias sustanciales para estos conjuntos de datos entre un esquema y otro.

Sin embargo, el escenario para Forest Fires y Slump corresponde a los grupos en los que, para el conjunto de testeo, se rechaza la hipótesis nula de igualdad de medias entre MSE y MAE. Esto implica diferencias importantes en los modelos entre un esquema y otro. Para entrenamiento, testeo y validación se rechaza H_0, en consecuencia se presenta una diferencia entre ambos esquemas.

Paradójicamente, los conjuntos de entrenamiento de la división aleatoria muestran en promedio errores menores (MAE y MSE). Luego, en los conjuntos de validación y testeo se revierte la relación, mostrando que la división controlada posee errores menores demostrables estadísticamente. Recordemos que el error de testeo es una estimación del EEE, lo que sería un ejemplo de que en algunos casos la división controlada puede producir redes con menores errores (tanto MSE como MAE). Por lo que puede concluirse que la división controlada para este caso, presenta mayor poder de generalización en las etapas tempranas del entrenamiento.

En principio, es posible conjeturar una diferente composición del MSE en términos de sesgo y varianza.

4.5. Conclusiones

A partir de los resultados del experimento A, se puede concluir que el error de testeo aproxima el error de generalización en los mismos órdenes de magnitud, independientemente del criterio de división de los datos.

Es relevante considerar que este experimento se realizó sobre DGP que producían atributos de entrada uniformemente distribuida e independientes. Lo cual representa el mejor escenario para el esquema de división aleatoria. Esto relaciona las propiedades estadísticas del conjunto con propiedades del entrenamiento en NN.

Además, los resultados del experimento A permiten interpretar los resultados del experimento B como aproximaciones del error de testeo al error de generalización, en principio, sin reparos respecto a su precisión dado que se desconoce el DGP.

El experimento B mostró que para algunos conjuntos de datos la división controlada producía errores de testeo más pequeños que la división realizada al azar, por lo tanto, daría lugar a modelos con mayor poder de generalización.

4.6. Trabajos futuros

Lo expuesto en este capítulo propone nuevos interrogantes que serán explorados en próximos trabajos.

En primer término, se evaluará si se presentan diferencias en los errores en las etapas finales del entrenamiento. Asimismo, se estudiarán los pesos que producen un esquema y otro con la intención de determinar sus diferencias. Se indagará sí esta misma situación se presenta cuando los datos se dividen en k partes.

En la experimentación propuesta se planteó la división considerando (X,y), en próximos trabajos se observará si se reportan los mismos beneficios considerando solo X o solo y. Además de la evaluación de otras funciones de distancias y otros tipos de datos diferentes de los números reales (variables binarias, categóricas, imágenes, etc).

El algoritmo de optimización Adam, según [KB15], es recomendado para escenarios donde se posee grandes volúmenes de datos. Se intentará determinar si con este algoritmo se obtienen

resultados similares a los presentados.

Por último, en relación con los DGP utilizados en la experimentación propuesta, se observan limitaciones en cuanto a las posibilidades experimentales. Algunas de estas modificaciones podrían acercarlos a condiciones menos experimentales. Se propondrán opciones de generación que incluyan progresivamente otras complejidades, como por ejemplo distribuciones de probabilidades no uniformes en los atributos de entrada o la inclusión de valores atípicos. Otra propiedad que aumenta la dificultad del problema, consiste en las relaciones que pueden observarse entre los atributos, generando dependencias entre ellos para acotar la independencia para diseñar DGP que se aproximen a las condiciones que presentan los problemas aplicados.

5. Optimización de hiperparámetros para bases de datos pequeñas

Contenidos de este capítulo

En este capítulo se introduce un algoritmo híbrido de HPO basado en recocido simulado. Se realiza una evaluación comparativa de su desempeño con una búsqueda aleatoria en bases de datos pequeñas.

Además, se intenta identificar una dinámica o mecanismo de compensación entre los hiperparámetros que gobiernan el entrenamiento de los modelos.

5.1. Introducción

El proceso de optimización automática de hiperparámetros o simplemente optimización hiperparámetros (HPO) pretende reducir la intervención de experto en el ciclo de desarrollo de un sistema de ML. A cambio de reducir esta intervención, el HPO exige una gran cantidad de recursos computacionales. Las metas de los HPO son varias [FH19]: en primer lugar, mejorar la precisión y eficiencia del entrenamiento de NN [MDB18]; luego, reducir los costos del trabajo de los expertos en IA y reducir el umbral de investigación y desarrollo.

El término hiperparámetro refiere a un parámetro cuyo valor no puede ser actualizado durante el entrenamiento. Estos pueden participar en la construcción de la arquitectura ó en la efectividad del entrenamiento del modelo. Se define hiperparámetro como una variable que gobierna el proceso de entrenamiento y/ o la selección de la topología de un modelo de NN.

Algunos de estos hiperparámetros pueden afectar el tiempo y el gasto de memoria. Otros afectan la calidad del modelo producto del entrenamiento y su capacidad para inferir resultados correctos cuando se implementa en nuevas entradas.

El proceso de selección de los valores apropiados se denomina ajuste de hiperparámetros. La optimización automática del ajuste de hiperparámetros se ha vuelto cada vez más necesaria debido a dos tendencias crecientes en el desarrollo de modelos de DL [YZ20]: el aumento de escala de estos modelos [TL19], y, paradójicamente, el desarrollo de pequeños modelos con menores cantidades de pesos y parámetros [MZZS18, SHZ+18, TL19].

La calidad de un modelo predictivo está condicionada críticamente por la configuración de sus hiperparámetros, aunque no se tiene una comprensión acabada de cómo los valores de estos interactúan entre ellos afectando el modelo [LJD+16].

La estrategia propuesta en esta tesis está dada por un algoritmo propio de optimización basado en recocido simulado (SA: *Simulated Annealing*). Esta técnica fue presentada por [KGJV83] como de propósito general que permite hacer una cantidad mínima de suposiciones. Se emplea a menudo en funciones con muchos mínimos locales debido a su capacidad para escapar de ellos [GP18].

El recocido o templado es un proceso en el que un material metálico es calentado y luego se enfría, haciéndolo más maleable. Cuando está caliente, los átomos del material tienen más

libertad para moverse y, mediante un movimiento aleatorio, tienden a asentarse en posiciones más adecuadas. Un enfriamiento lento lleva el material a un estado cristalino ordenado. Contrariamente, un enfriamiento rápido y brusco provoca defectos en el material porque se ve obligado a asentarse en su estado actual.

El SA, inspirado en la metalurgia, es uno de los métodos metaheurísticos más simples y conocidos para abordar problemas de optimización global en cajas negras difíciles, cuya función objetivo no se proporciona explícitamente y solo se puede evaluar a través de una simulación costosa en computadora. En esta simulación, la variable de temperatura es utilizada para controlar el grado de estocasticidad durante la búsqueda aleatoria (RS: Random Search).

La temperatura comienza alta, lo que permite que el proceso se mueva libremente por el espacio de búsqueda, con la esperanza de que en esta fase se encuentre una buena región con el mejor mínimo local. Luego, la temperatura se reduce lentamente[1], disminuyendo la estocasticidad y obligando a la búsqueda a converger al mínimo. Esta técnica es un método clásico de búsqueda de la optimización metaheurística, cuya convergencia asintótica hacia un óptimo global ha sido probada [GP18].

Esta convergencia está dada en ciertas condiciones: el algoritmo de SA posee la propiedad de convergencia estocástica hacia un óptimo global siempre que se proporcione un diagrama de caída de temperatura infinitamente largo con pasos de caída infinitamente pequeños. Este esquema es puramente teórico y se intentará en la práctica acercarse a este ideal mientras permanezca dentro de tiempos razonables de ejecución[GP18].

Simulated Annealing (SA) es uno de los métodos metaheurísticos más simples y conocidos para abordar problemas de optimización global en cajas negras difíciles, cuya función objetivo no se proporciona explícitamente y solo se puede evaluar a través de una simulación costosa en computadora. Se utiliza ampliamente en aplicaciones de la vida real.

5.2. Antecedentes

El ajuste manual de hiperparámetros puede funcionar muy bien cuando el programador tiene un buen punto de partida (como podría ser otro modelo a emplear con transferencia de

[1] En la implementación propuesta el esquema de enfriamiento utilizado es el geométrico en ambas etapas.

conocimiento), o cuando se posee experiencia en la definición de valores de hiperparámetros para NN aplicadas a tareas similares. Sin embargo, esto no siempre es una opción. En este caso, los algoritmos automatizados pueden ser útiles para determinar estos valores. A continuación se presentan estrategias automáticas reconocidas.

La HPO puede ser una tarea que tenga una resolución tan compleja como se quiera. La búsqueda por cuadrícula y la RS se encuentran entre las menos sofisticadas.

En la búsqueda por cuadrícula, para cada hiperparámetro se selecciona un conjunto finito de valores a explorar. El algoritmo luego entrena un modelo para cada combinación de valores en el producto cartesiano del conjunto de valores para cada hiperparámetro individual. El experimento que produce el error más pequeño en el conjunto de validación se devuelve como la mejor combinación de hiperparámetros. Este algoritmo es de $O(n^m)$ donde n son los hiperparámetros y m es la cantidad de valores a evaluar.

La RS es una alternativa a la búsqueda por cuadrícula: simple de programar, más conveniente de usar y que converge mucho más rápido a buenos valores de los hiperparámetros [BBBK11]. La RS procede de la siguiente manera: para cada hiperparámetro se define una distribución marginal. Por ejemplo: un Bernoulli o multinoulli para hiperparámetros binarios o discretos, o una distribución uniforme en una escala logarítmica para hiperparámetros de valor real positivo.En este caso, no se debe discretizar ni agrupar los valores de los hiperparámetros, de modo que es posible explorar un conjunto más grande de valores evitando costos computacionales adicionales. La RS puede ser exponencialmente más eficiente que una búsqueda por cuadrícula cuando hay varios hiperparámetros que no afectan fuertemente la medida de rendimiento.

El ajuste de hiperparámetros de caja negra es popular en la industria y la academia, los ejemplos incluyen a Google Vizier [GSM$^+$17], Hyperopt [BBBK11] , Spearmint [SLA12]. Estos enfoques asumen la existencia de una métrica en un conjunto de validación. La optimización basada en modelos secuenciales, es una familia de métodos que construye un modelo de la métrica de validación con hiperparámetros como entrada [HHLB11]. Este modelo es generalmente entrenado en los ensayos previos de los hiperparámetros. Una categoría de algoritmos basado en modelos secuenciales es la Optimización Bayesiana que construye un modelo probabilístico de la función anterior. Un relevamiento detallado sobre la optimización bayesiana se puede encontrar

en [SSW+16].

La búsqueda eficiente para la HPO ha sido hegemonizada por los métodos de optimización bayesianos [BBBK11, SLA12, HHLB11]. Dentro de la novedad, se encuentra el algoritmo de hiperbanda presentado por [LJ18], como una técnica de propósito general que permite hacer una cantidad mínima de suposiciones.

5.3. HPO basada en recocido simulado

El algoritmo propuesto recorre el espacio de búsqueda de dos maneras: exploración y explotación, ambas controladas por la temperatura. En la exploración, cualquier punto del espacio controlado de búsqueda es factible. Progresivamente, a medida que la temperatura disminuye, también lo hace el espacio de búsqueda.

En contraposición, durante la explotación los cambios se limitan a un entorno reducido del valor del parámetro. El cambio de fase entre exploración y explotación se da cuando se encuentra un óptimo pseudo-global, en un intento de refinar la configuración hallada. Este óptimo representa el mejor candidato encontrado hasta el momento, es decir el mejor óptimo local.

Téngase en cuenta que reevaluar una misma configuración en múltiples oportunidades es un modo implícito de evaluar diferentes puntos iniciales. Esto es de particular interés, ya que estos algoritmos suelen ser controlados por la cantidad de intentos que realizan. El algoritmo propuesto se detiene luego de agotar los intentos estipulados. Además, el algoritmo incluye una lista tabú[2]: los puntos ya visitados son evitados con algún grado de probabilidad. De esta forma, se previene cierto grado de reevaluación, lo cual resulta conveniente.

En el caso de las NN, el punto inicial se refiere a la configuración de los pesos previa al entrenamiento. Recordemos la existencia del LT, que no es más que el descubrimiento de un punto inicial extremadamente favorable. Por este motivo, se permite la repetición de una configuración, aunque es poco probable que esto ocurra.

Otra particularidad del algoritmo planteado es que se establece un criterio de prioridad de selección de los HP con la intención de incorporar, en la medida de lo posible, el saber hacer

[2]El principio básico de la lista tabú es permitir durante un período determinado la búsqueda local siempre y cuando existan movimientos que permitan encontrar el óptimo local. Se previene la evaluación de sitios que ya han sido examinados mediante el uso de una estructura de datos que registra la historia reciente.

del experto. De tal modo que si el operador desconoce la relevancia de un HP sobre otro, puede mantener la distribución uniforme de los mismos. Pero si se conociera la mayor relevancia de un HP sobre los otros, puede indicarse al HPO propuesto mediante una mayor probabilidad de selección. Esto es implementado a través de la generación, previa al comienzo del enfriamiento, de una lista de parámetros a evaluar que respete el criterio de prioridades establecido.

$$T_p(k) = \frac{T_0}{k} \tag{5.1}$$

El algoritmo posee un control adaptativo de la temperatura con relación al parámetro que se está optimizando. Este control está inspirado en la propuesta de [Ing11] . De esta forma, la temperatura, en lugar de ser un valor real, es un vector de temperaturas en el espacio D-dimensional de parámetros. Algunas de estas dimensiones son continuas y otras discretas. En la fig. 5.1 puede verse una representación gráfica de este concepto de la corrida 85 correspondiente al conjunto de datos F1 para 50 intentos. En este experimento la probabilidad de cada parámetro es uniforme. Cada una de las barras corresponde al vector de temperatura en el momento-intento *i* . De esta forma, en cada iteración el valor de temperatura de uno de los parámetros es disminuído siguiendo el criterio de la ec. 5.1 conocida como recocido rápido (FA: Fast anneling) [Ing11]. Durante la etapa de explotación la temperatura del parámetro no es actualizada. En algunos casos, el valor tomado del parámetro provoca que la red retorne un "no número" (NaN: Not a number) como resultado de la evaluación de su función de pérdida. En este caso el valor del parámetro es reiniciado a su temperatura máxima inicial, implementando así un mecanismo de recalentamiento. En el Apéndice D se describe con detenimiento el algoritmo del HPO propuesto.

5.4. Datos, métodos y materiales

En primer lugar, se realiza un análisis del desempeño del HPO en comparación con una RS de HP para establecer una línea de base con respecto a su desempeño (Experimento C). Luego, nos enfocamos en el estudio de la dinámica de HP que gobiernan la construcción del modelo haciendo uso del algoritmo presentado. Se proponen dos estudios aplicados a bases de datos pequeñas: uno sobre la arquitectura (Experimento D) y otro referido a la dinámica de la

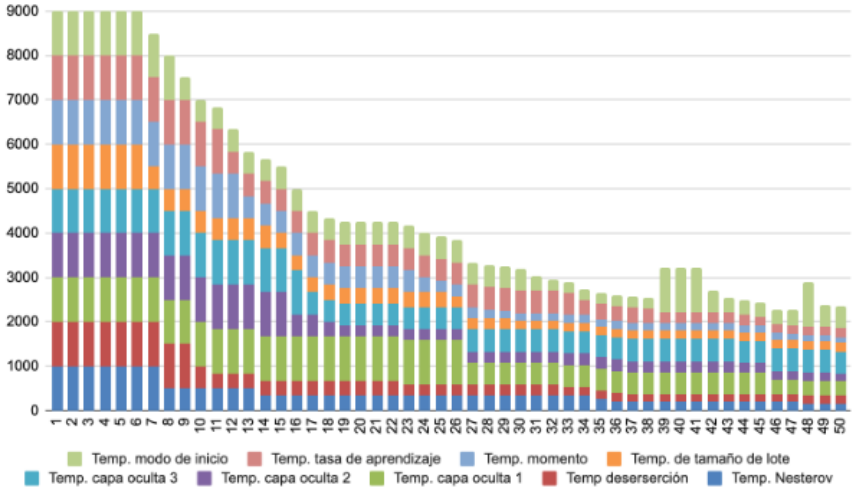


Figura 5.1: Evolución del vector de temperaturas

regularización y el algoritmo de aprendizaje (Experimento E).

La presencia de un ticket de lotería supone la posibilidad de un "ticket del infierno", es decir un punto inicial con una configuración suficientemente desfavorable de la cual no sería posible recuperarse. En el caso del análisis del desempeño del HPO no se toman consideraciones especiales respecto a este asunto, a diferencia de los experimentos D y E.

En el experimento C todos los HP tienen las mismas probabilidades de ser seleccionados, de forma tal que no se introduce saber experto y la comparación se realiza de modo imparcial.

En el caso de los Experimentos D y E se consideró tanto la influencia del LT como del "ticket del infierno": se ordenaron las redes por su desempeño en términos del MSE y se eliminaron del análisis las redes del primero y último decil.

En todos los casos se utilizan los DGP presentados para el Experimento A del Capítulo 4 (ver sección 4.3). Para el estudio C, se entrenaron 90 redes para cada esquema de HPO, 30 NN correspondientes a cada F_i con $1 \leq i \leq 3$. Para el estudio D y E, se entrenaron 60 NN correspondientes a 20 por cada F_i.

En el caso del estudio C, las NN provenían del espacio de búsqueda que puede ser apreciado en el cuadro 5.1. Por otro lado, en el estudio D, las NN provenían de otro espacio de búsqueda, con la particularidad de que no todos los parámetros poseían la misma probabilidad de selección. Los detalles de estos valores pueden consultarse en el cuadro 5.2. Análogamente al experimento

D, el experimento E posee su propio espacio de búsqueda y asignación de relevancia de HP indicados en el cuadro 5.3. En ambos casos, para un conjunto de datos de 1000 observaciones.

El criterio de asignación de probabilidades se realizó dando preponderancia a los HP en estudio y a la LR. Por otro lado, se asignaron probabilidades de selección marginales a otros HP para permitir su optimización en caso de que el valor inicial fuera perjudicial.

5.5. Resultados y discusión

En el experimento C, como se mencionó anteriormente, la distribución de probabilidades entre los parámetros es uniforme. Mientras que en los experimentos D y E la probabilidad de selección se concentra en algunos de ellos. En el caso de D, en el diseño de la arquitectura. Mientras que en el último, la combinación de la tasa de deserción y el tamaño del lote es prioritaria.

5.5.1. Experimento C: ¿Es el algoritmo de optimización de HP propuesto de mejor rendimiento que la búsqueda aleatoria?

La respuesta es afirmativa, a partir de los 50 intentos. El desempeño del HPO comienza a ser superior a partir de los 50 intentos para todos los conjuntos de datos evaluados. El recocido simulado tiene un desempeño cercano al 16 % en promedio en términos del MSE obtenido del mejor modelo hallado. En la 5.4, se exponen los valores medianos y promedios utilizados para la función de pérdida empleada. A partir del cuadro, se observa que a medida que aumentan los intentos la mediana del HPO propuesto disminuye.

En este estudio todos los HP tienen las mismas probabilidades de ser seleccionados, de forma tal que no se introduce ningún sesgo proveniente del saber hacer del especialista, asegurando una comparación imparcial.

En las fig. 5.2, un diagrama de cajas y bigotes muestra la dispersión de los valores obtenidos para cada uno de los conjuntos de datos, evidenciando la evolución según la cantidad de intentos. Cada uno agrupa diferentes temperaturas. Entrando en detalle con respecto al funcionamiento

Hiperparámetro	Tipo	Rango de búsqueda
Cantidad de unidades en capa oculta El algoritmo permite optimizar la cantidad de unidades en cada una de las capas. En este caso los modelos poseen tres capas.	entero	$[400, 1000]$
Tasa de deserción Esta tasa establece aleatoriamente, en cada época de entrada de la unidad en 0, lo que ayuda a evitar el sobreajuste entrenamiento, el valor de entrada de la unidad en 0, lo que ayuda a evitar el sobreajuste (técnica mencionada en el Cap 4.).	real	$[0{,}25, 0{,}90]$
Tamaño de lote El tamaño del lote se refiere a la cantidad de observaciones utilizadas en cada iteración o época del entrenamiento para computar la actualización de los pesos.	entero	$[2, 64]$
Modo de inicialización de los pesos Distribución estadística que determina el valor inicial de los pesos. Estos valores se seleccionan de diferentes distribuciones aleatorias. La primera de ellas es una distribución uniforme que obtiene sus valores del rango [-0.5,0.5]. La segunda opción considerada es una distribución normal con media 0 y desvío estándar 0.05.	conjunto discreto	[distribución uniforme, distribución normal]
LR inicial La LR es ajustada a medida que avanza el entrenamiento siguiendo una disminución exponencial a una tasa del 96 % calculada sobre 100.000 épocas.	real	$[0{,}0000001, 0{,}01]$
Momento Cuando el momento es diferente de cero, este se combina con la LR para la determinación de los nuevos pesos.	real	[0.90, 0.99]
Nesterov Este es una extensión del algoritmo de optimización de descenso de gradiente en el que la actualización se realiza empleando la derivada parcial de la actualización proyectada en lugar del valor variable actual derivado [SMDH13].	Booleano	[Verdadero, Falso]
Tamaño del conjunto de datos		1000 observaciones
Función de pérdida		MSE

Cuadro 5.1: Condiciones de entrenamiento y espacio de búsqueda a optimizar en el experimento C.

del HPO propuesto, en la tabla coloreda como un mapa de calor se puede observar cómo el desempeño aumenta a medida que aumenta la temperatura y crecen los intentos.

El comportamiento observado en la 5.4. no presenta ninguna sorpresa, considerando la demostración teórica de convergencia del recocido simulado a medida que la caída de temperatura

Hiperparámetro	Espacio de búsqueda		Probabilidad de selección (%)
	Tipo	Rango	
Configuración de unidades en capas ocultas uno, dos y tres respectivamente	Tipo discreto	$\{A = (500, 400, 300), B = (400, 400, 400), C = (300, 400, 500)\}$	68.96
Tasa de deserción	Tipo real	$25\,\%, 75\,\%$	3.44
Tamaño de lote	Tipo discreto	$\{A = 4, B = 16, C = 32, D = 64\}$,	34.78
Modo de inicialización de los pesos	Tipo conjunto discreto	[distribución uniforme, distribución normal]	3.44
LR	Tipo real	$[0{,}0000001, 0{,}01]$	17.24
Nesterov	Tipo conjunto discreto	$Verdadero, Falso$	3.44

Cuadro 5.2: Distribución de probabilidades para la selección de los HPs para Exp D

Hiperparámetro	Espacio de búsqueda		Probabilidad de selección (%)
	Tipo	Rango	
Cantidad de unidades en capa oculta 1	Tipo entero	[400, 1000]	1.82
Cantidad de unidades en capa oculta 2	Tipo entero	[400, 1000]	1.82
Cantidad de unidades en capa oculta 3	Tipo entero	[400, 1000]	1.82
Combinación de tasa de deserción y tamaño de lote respectivamente	Tipo discreto	$\{A = (0{,}25, 4), B = (0{,}50, 32), C = (0{,}75, 64), D = (0{,}75,\ 4), E = (0{,}25, 64)\}$	72.27
Modo de inicialización de los pesos	Tipo conjunto discreto	[distribución uniforme, distribución normal]	1.82
LR	Tipo real	$[0{,}0000001, 0{,}01]$	18.18
Nesterov	Tipo conjunto discreto	$Verdadero, Falso$	1.82

Cuadro 5.3: Distribución de probabilidades para la selección de los HPs para Exp E

tiende a ser infinitamente larga con pasos de caída infinitamente pequeños.

Por otro lado, el desempeño de SA no es consistente en los 20 intentos. Si bien SA tiene asegurada la convergencia, esta propiedad se da cuando la cantidad de intentos tiende a infinito, por lo que en un presupuesto de recursos más limitado podría no verse reflejada esta ventaja, lo cual se observa en este caso.

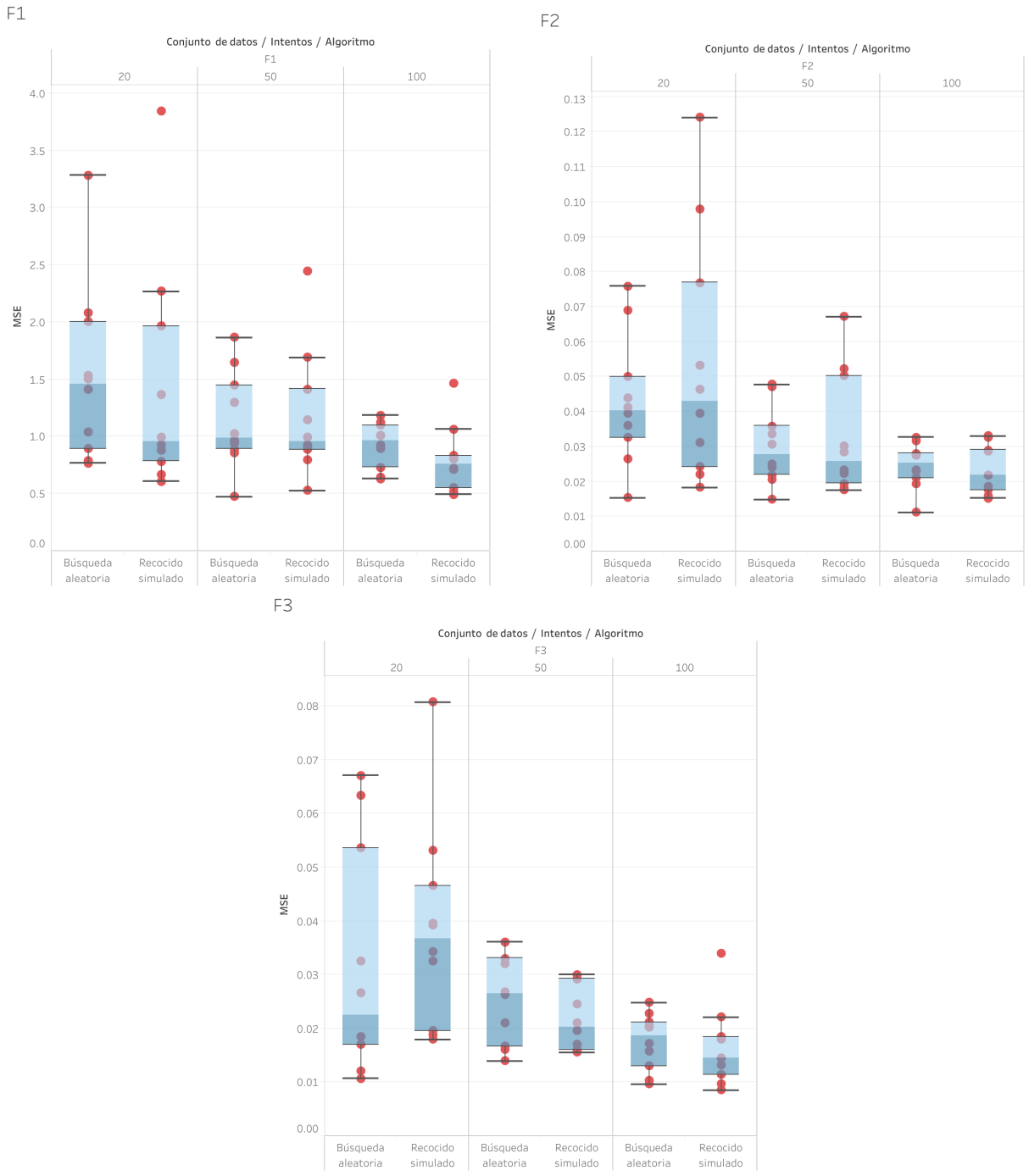


Figura 5.2: Diagrama de cajas y bigotes

5.5.2. Experimento D: ¿Se observa preferencia por alguna arquitectura emergente?

La respuesta es negativa, pues no parece haber preferencia por un modelo. Se busca si surge algún patrón en la arquitectura de las capas ocultas, buscando algo similar a la forma de un embudo, donde una capa posee menos unidades que la otra o se mantiene en valores similares. En

Intentos	Conjutos de datos	MSE Mediana			MSE Promedio		
		Búsqueda aleatoria	Búsqueda recocido simulado	%	Búsqueda aleatoria	Búsqueda recocido simulado	%
20	F1	1.46	0.96	34.40 %	1.53	1.07	29.82 %
	F2	0.04	0.04	-6.34 %	0.04	0.05	-21.99 %
	F3	0.02	0.04	-62.97 %	0.03	0.04	-14.36 %
Promedio general				-11.64 %			-2.18 %
50	F1	0.99	1.07	-7.88 %	0.99	0.96	7.50 %
	F2	0.02	0.03	-15.60 %	0.03	0.03	23.55 %
	F3	0.03	0.02	31.81 %	0.03	0.02	17.46 %
Promedio general				2.77 %			16.17 %
100	F1	0.97	0.77	20.16 %	0.92	0.76	17.19 %
	F2	0.03	0.02	11.84 %	0.02	0.02	11.10 %
	F3	0.02	0.01	22.36 %	0.02	0.01	17.19 %
Promedio general				18.12 %			15.16 %

Cuadro 5.4: Resultados agrupados de la función de pérdida MSE para el Experimento C.

la fig. 5.3 se observan los modelos evaluados: los mismos poseen la misma cantidad de unidades distribuidas de diferente forma en las distintas capas ocultas.

En el cuadro 5.5 se exponen las medidas MSE obtenidas para todas las corridas que fueron ejecutadas con presupuesto de 150 intentos. Observemos que los valores obtenidos en esta tabla no son directamente comparables con los reportados en el cuadro 5.4, debido a que poseen diferentes distribuciones de probabilidad. En la fig. 5.4 se expone la distribución porcentual de la evaluación de las combinaciones estudiadas. Esto muestra que cada una de ellas ha sido evaluada en órdenes de magnitud similares.

Por último, en el cuadro 5.6 se reportan los valores del mejor óptimo local encontrado luego del proceso de optimización de HP. En ninguno de los casos se muestra una preferencia marcada por una de las configuraciones. Sin embargo, se observa que la configuración B muestra promedios menores de MSE.

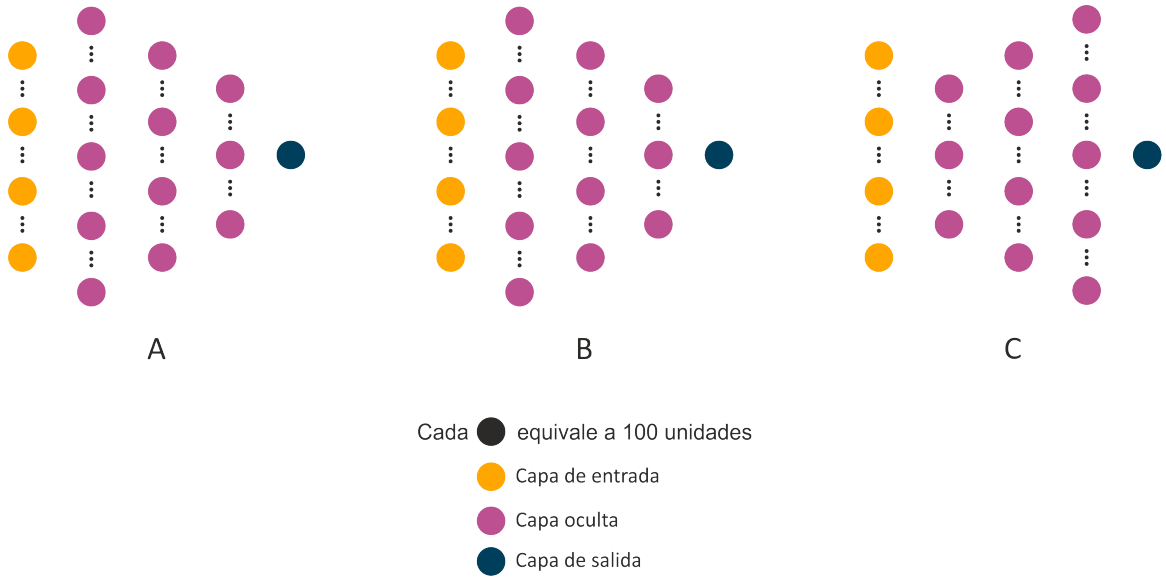


Figura 5.3: Modelos arquitectónicos evaluados

Conjunto de datos	Mediana	MSE			
		Todos los deciles		Deciles en estudio	
		MSE promedio	Desvío standard	MSE Promedio	Desvío standard
F1	0.7042	0.7333	0.1442	0.6955	0.0956
F2	0.0122	0.0127	0.0129	0.0126	0.0109
F3	0.0128	0.0152	0.0166	0.0149	0.0117

Cuadro 5.5: Comparación de MSE entre el general y los cuartiles a considerar para Experimento D

5.5.3. Experimento E: ¿El tamaño del lote condiciona la tasa de regularización?

La motivación inicial de este experimento ha sido observar la posible relación entre la regularización dada por el tamaño del lote y la tasa de deserción. Volviendo a la pregunta planteada, la respuesta es: hay un condicionamiento entre estos hiperparámetros.

En el cuadro 5.7 se presentan los errores promedio para todas las corridas que fueron ejecutadas con presupuesto de 50 intentos. En esta tabla pueden observarse: la mediana, el promedio del MSE y su desvío estándar para todos los deciles y los deciles en consideración. En la fig. se muestra la distribución porcentual de evaluación de las combinaciones estudiadas, tanto en general cómo en su desagrupación por los distintos DGP. En el segundo cuadrante se observa

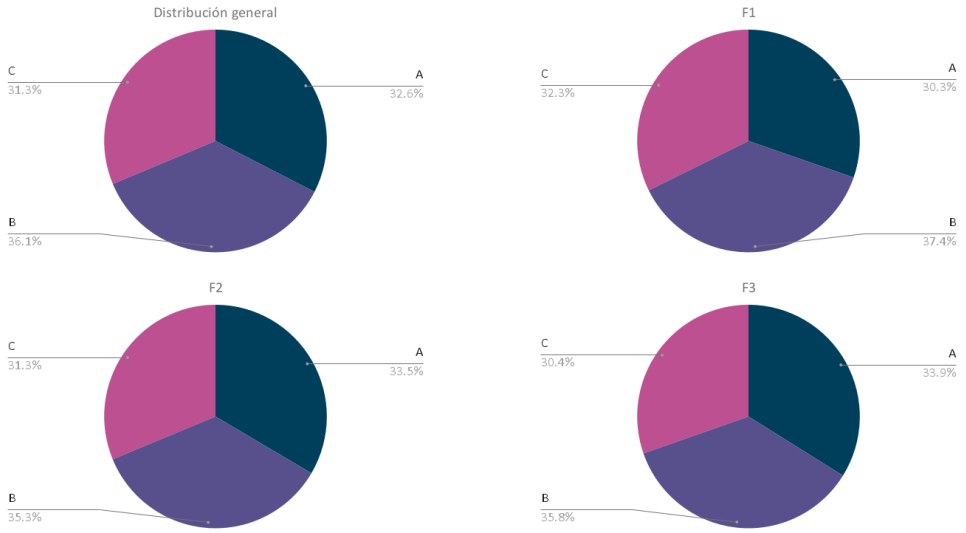


Figura 5.4: Distribución porcentual de la evaluación de las combinaciones estudiadas.

Conjunto de datos	Configuración óptima	MSE promedio	Desvío standard
F1	A 9	0.8218	0.3852
	B 10	0.6559	0.2829
	C 1	0.7092	
F2	A 7	0.0125	0.0046
	B 6	0.0125	0.0047
	C 7	0.0130	0.0039
F3	A 9	0.0158	0.0067
	B 4	0.0133	0.0039
	C 7	0.0155	0.0077

Cuadro 5.6: Comparación de MSE entre el general y los cuartiles a considerar para experimento D

la distribución de configuraciones para el DGP F1, en el tercero F2, y por último F3.

En el cuadro 5.8 se reportan los valores del mejor óptimo local encontrado en el proceso de optimización de HP, que muestra el predominio de la combinación E (tamaño de lote de 64 observaciones y un porcentaje de 25 % de tasa de deserción) por sobre las otras. Esta es la combinación que minimiza MSE del conjunto de validación con mayor frecuencia, mostrando preferencia casi unánime por un aumento del lote en desmedro de una disminución de la tasa de deserción.

En la fig. 5.6 se muestran las combinaciones que resultaron en un mejor óptimo local en algún momento de la ejecución. La combinación B refiere a un tamaño de lote de 32 observaciones y

Conjunto de datos	Mediana	MSE			
		Todos los deciles		Deciles en estudio	
		MSE promedio	Desvío standard	MSE Promedio	Desvío standard
F1	0.6379	0.6236	0.1783	0.6206	0.1312
F2	0.0195	0.0219	0.0084	0.0203	0.0046
F3	0.0180	0.0178	0.0064	0.0176	0.0044

Cuadro 5.7: Comparación de MSE entre el general y los cuartiles a considerar.

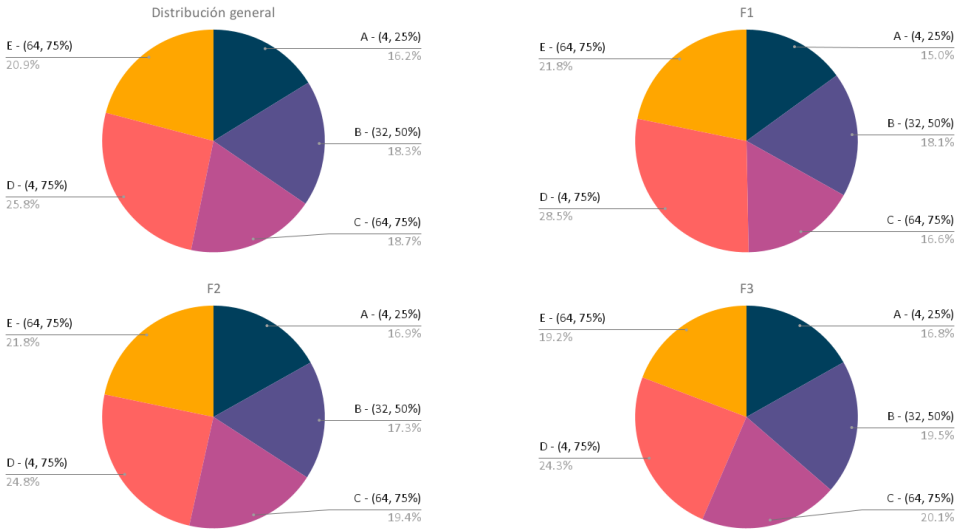


Figura 5.5: Distribución porcentual de la evaluación de las combinaciones estudiadas.

Conjunto de datos	Configuración óptima		MSE promedio	Desvío standard
F1	E	20	0.6236	0.1783
F2	B	1	0.0198	
	E	19	0.0220	0.0086
F3	B	1	0.0201	
	E	19	0.0177	0.0065

Cuadro 5.8: Valores del mejor óptimo local encontrado

una tasa de deserción del 50 %. Estos dos puntos muestran que, ante el aumento del tamaño de lote, se tiende a una disminución en la tasa de deserción. Luego, la combinación C se refiere a un tamaño de lote de 64 observaciones y una tasa de deserción del 75 %. En este caso, un aumento en relación con la combinación B de ambos HP. Las combinaciones restantes tienen una menor participación.

La hipótesis inicial era que pequeñas tasas de deserción permitirían mayores tamaños de

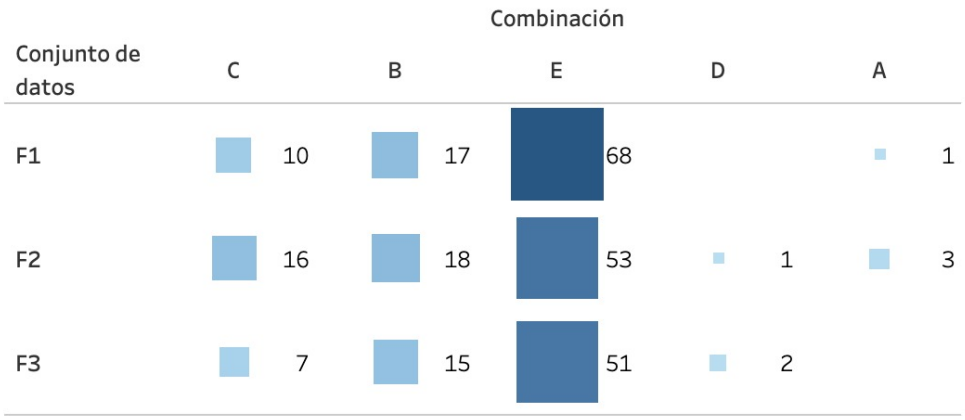


Figura 5.6: Combinaciones que resultaron en un mejor óptimo local en algún momento de la ejecución

lote: a medida que aumenta la tasa de deserción, los tamaños de lote tenderían a disminuir. La dinámica de estas fuentes de regularización tal vez requiera una explicación más extensa.

Un aspecto de los algoritmos de ML que los diferencia de los algoritmos de optimización general es que la función objetivo generalmente se descompone como una suma entre los ejemplos de entrenamiento. Los algoritmos de optimización para el ML normalmente calculan cada actualización de los parámetros en términos de un valor esperado de la función de costo estimada, usando solo un subconjunto de los términos de la función de costo total. Este subconjunto es llamado lote.

Tanto la tasa de deserción como el tamaño del lote son conocidos mecanismos de regularización. Específicamente, los lotes pequeños pueden ofrecer un efecto de regularización [WM03], quizás debido al ruido que agregan al proceso de aprendizaje. El entrenamiento con un tamaño de lote tan pequeño podría requerir una LR más pequeña para mantener la estabilidad debido a la gran variación en la estimación del gradiente.

5.6. Conclusiones

El algoritmo presentado en este capítulo puede considerarse como un enfoque mixto entre una optimización manual y una automática, debido a que incluye la posibilidad de incorporar un criterio en la selección de cada hiperparámetro, lo que permite sumar el entendimiento del

experto sobre la relación entre los hiperparámetros.

Se mostró que la propuesta presenta un mayor desempeño en promedio que la línea de base a partir de los 50 intentos, lo que sugiere que la decisión sobre qué estrategia de optimización utilizar estaría atada al presupuesto en recursos de ejecución.

Los algoritmos de HPO a menudo tienen sus propios hiperparámetros, aunque estos hiperparámetros pueden ser más fáciles de elegir[3]. En la estrategia propuesta, el algoritmo no cuenta con hiperparámetros, lo que representa una ventaja contra otros que sí los poseen.

Se trabajó en la dinámica de interrelación de hiperparámetros, experimentando con algunas de estas relaciones restringidas a bases de datos pequeñas. Se exploró el diseño de la arquitectura y se mostró lo que ocurre entre el tamaño de lote y la tasa de deserción.

5.7. Trabajos futuros

Otro aspecto de particular importancia es el desempeño del algoritmo de HPO propuesto en comparación con los clásicos y con otros más novedosos. Se comparará a futuro si la propuesta tiene mejores resultados para NN que hiperbanda o una optimización bayesiana sobre una biblioteca de referencia para la DL (HPOlib), presentada por [EFH+13] en un extenso estudio más allá de los problemas de regresión.

El algoritmo propuesto posee solo un esquema de enfriamiento. En próximos trabajos se realizará una tarea comparativa de diferentes esquemas.

La selección adecuada de este parámetro supone un nuevo nivel de complejidad en la selección de los HP correspondientes al modelo final. Por lo que sería apropiado que se incluya la selección de este esquema de forma dinámica. Es una decisión de diseño mantener el algoritmo libre de HP propios.

Además, en próximas versiones se evaluará la incorporación de un mecanismo de calentamiento más elaborado, sin necesidad de optimizar este hiperparámetro. Como el herrero, que en oportunidades necesita volver a calentar la pieza, algunos problemas podrían requerir de recalentamiento. Es didáctico ejemplificar el asunto como si fuera un terreno con múltiples accidentes

[3]Cabe observar que es posible lograr un rendimiento aceptable en una amplia gama de tareas usando los mismos hiperparámetros secundarios para todas las tareas.

Figura 5.7: Principios de recocido simulado

topográficos. En la fig. 5.7 se puede observar el planteo. Se desea encontrar el valle más bajo en este terreno (mínimo global), por lo que el HPO abordaría este problema de manera análoga a una pelota que rebota sobre las montañas de valle en valle. Comenzaríamos con una temperatura alta, lo que permitiría que la pelota rebote muy alto y pueda sobrepasar cualquier montaña para acceder a cualquier valle, dados suficientes rebotes. A medida que la temperatura se vuelve relativamente baja, la pelota no puede rebotar tan alto, y puede asentarse y quedar atrapada en valles relativamente más pequeños. Este mismo principio controlaría la temperatura. Si luego de una cantidad de intentos fallidos no se ha logrado una mejora, la temperatura comenzaría a subir probabilísticamente. Por el contrario, si progresivamente se han encontrado soluciones que mejoran con el tiempo, la temperatura decrecería [Moi02].

También se replanteará un manejo explícito del punto inicial, ya que la misma configuración de hiperparámetros puede verse infra o sobrevalorada dependiendo de este punto. Se realizará un análisis de perturbación con el objetivo de cuantificar la sensibilidad de las NN respecto al punto inicial y cómo esto puede mitigarse.

Que en el Experimento D no hayan aparecido claras preferencias por ninguno de los diseños arquitectónicos, sugiere aprovechar este hecho para simplificar el espacio de búsqueda, manteniendo la cantidad de unidades constantes para todas las capas (configuración B), y en su lugar concentrar los esfuerzos en optimizar la cantidad de unidades. Más adelante se realizará un estudio comparativo de este enfoque.

En cuanto al experimento E, se plantea el interrogante sobre las consecuencias a largo plazo de la combinación predominante. En etapas tempranas del entrenamiento mostró ser la más eficiente; sin embargo, esto puede resultar contraproducente en etapas tardías del entrenamiento, resultando en modelos con mayor tendencia a la subestimación del error de testeo.

6. Conclusiones

Contenidos de este capítulo

En este capítulo se establecen las conclusiones derivadas del trabajo realizado, sintetizando los resultados y hallazgos de la investigación. Asimismo, se destacan avances y novedades introducidas en el campo de estudio en el que se realiza la investigación. Por último, se presenta las nuevas direcciones y áreas de trabajo que puedan ser exploradas en una etapa posterior a la presentación de esta tesis.

6.1. Conclusiones

"I would rather have questions that can't be answered
than answers that can't be questioned."[1]
– Richard Feynman

En esta tesis se propuso como objetivo global obtener un mejor desempeño de DNN para problemas de regresión con base de datos tabulares pequeñas. Con el propósito de dilucidar este interrogante, se estudiaron modelos de RN profundas con bases de datos tabulares pequeñas, lo que ha representado un doble desafío.

Actualmente, estos modelos dependen fuertemente del supuesto de la disponibilidad de grandes volúmenes de datos. Por lo cual, esto constituye una limitación fundamental.

Un conjunto de datos insuficiente es un problema inevitable en algunos dominios. Además, parece no haber acuerdo en cuándo una base de datos es pequeña. Esta discusión parece estar centrada en la cantidad de parámetros de un modelo, relativizando otros aspectos que hacen al desempeño, como es el caso del punto inicial. Esta definición penaliza los modelos de gran volumen, lo que, según el estado del arte actual, plantea un conflicto con la evidencia existente respecto a que los modelos sobredimensionados muestran mayor poder de generalización.

La definición con la que se ha trabajado es perfectible, ya que sólo considera los aspectos cuantitativos del conjunto de datos, sin tener en cuenta aspectos cualitativos. Por estos motivos, la discusión está lejos de llegar a un consenso respecto la cuestión de fondo sobre cuándo una base de datos es pequeña.

Otras de las conclusiones derivadas del trabajo abordado refiere a la forma en la cual el conjunto de datos es dividido previo al entrenamiento. Es posible concluir que para algunos conjuntos de datos la división controlada produce errores de testeo más pequeños que la división realizada al azar, por lo tanto, daría lugar a modelos con mayor poder de generalización.

Dado que los modelos de DL suelen tener un gran número de hiperparámetros que deben ajustarse para obtener un rendimiento óptimo, el ajuste apropiado de estos puede marcar la diferencia entre un modelo con un rendimiento inaceptable y un rendimiento notable.

[1] "Prefiero tener preguntas que no puedan ser respondidas a respuestas que no puedan ser cuestionadas.". [Fey81]

Aunque se dispongan de algunos algoritmos de HPO para DL, aún no se ha dicho la última palabra al respecto. A menudo, estos algoritmos tienen sus propios hiperparámetros. Aunque estos hiperparámetros pueden ser más fáciles de elegir, esto resulta un problema tautológico. En el HPO propuesto el algoritmo no cuenta con hiperparámetros, lo que representa una ventaja contra otros.

Debido a que incluye la posibilidad de incorporar un criterio en la selección de cada hiperparámetro, puede considerarse como un enfoque mixto entre una optimización manual y una automática, lo que permite sumar el entendimiento del experto sobre la relación entre los hiperparámetros. Se trabajó en la dinámica de interrelación de hiperparametros, experimentando con algunas de estas relaciones restringidas a bases de datos pequeñas. Esto permitió exponer alguna de ellas, lo que supone una forma de conocer aspectos teóricos.

6.2. Principales contribuciones

Se formalizó una definición de base de datos pequeña que supone una relación implícita entre la cantidad de parámetros y su desempeño. Esta definición se centra en la cantidad de observaciones del conjunto de datos en relación con la arquitectura del modelo, sin considerar otros aspectos que se saben relevantes.

Se propuso un algoritmo evolutivo para particionar el conjunto de datos que permita encontrar la división que más se asemeje al conjunto original. Esta similitud es medida por dos funciones: distancia de Wassertein y distancia de Mahalanobis. La experimentación expuso cómo la partición aleatoria produce diferencias entre los subconjuntos y cómo, a medida que crece el tamaño del conjunto de datos, la partición controlada se vuelve innecesaria. La experimentación mostró además en qué rango de tamaño del conjunto resulta más beneficiosa.

Se estudió el efecto de la división aleatoria y controlada de conjuntos de datos pequeños tabulares en modelos de DL en las etapas tempranas del entrenamiento. Se concluyó que para algunos conjuntos de datos la división controlada produce un menor error de testeo, lo que implicaría mayor poder de generalización en las etapas tempranas del entrenamiento.

Se exploró el supuesto de que la división aleatoria del conjunto de datos es la estrategia usual cuando los datos abundan. Se comparó entre el mecanismo aleatorio y el controlado por el

algoritmo evolutivo. Se observó que el punto de máximo beneficio de la división controlada por el algoritmo se encuentra entre 103 y 104 observaciones del conjunto de datos.

Se planteó un algoritmo de optimización de hiperparámetros de DL basado en el recocido simulado que puede considerarse como un enfoque mixto entre una optimización manual y una automática, debido a que incluye la posibilidad de incorporar un criterio de prioridad en la selección de los hiperparámetros. Además, se estudió la interrelación de algunos hiperparámetros en bases de datos pequeñas tabulares.

Se trabajó en la dinámica de interrelación de hiperparámetros, experimentando con algunas de estas relaciones restringidas a bases de datos pequeñas.

6.3. Trabajos futuros

En base al trabajo efectuado, se vislumbran interesantes perspectivas para continuar las investigaciones realizadas, tanto en investigación teórica como aplicada.

El TL se refiere al paradigma de ML en el que un algoritmo extrae conocimiento de uno o más escenarios de aplicación para ayudar al rendimiento en un escenario de destino. Este tipo de aprendizaje ablanda la hipótesis de que los datos de entrenamiento deben ser independientes y distribuidos de manera idéntica con los datos de validación y testeo. Para este enfoque no es necesario que los datos de entrenamiento y testeo provengan del mismo PGD. Dado que TL es una herramienta que permitiría sobreponerse a la escasez de datos para algunos problemas. Sin embargo, su aplicación no es sencilla, y aún le queda un largo camino para lograr su objetivo de aplicación efectiva a los problemas aplicados planteados. El TL utiliza no solo los datos en el dominio de la tarea de destino como entrada para el algoritmo de aprendizaje. Además, hace uso de los procesos de aprendizaje en el dominio de origen, los modelos y la descripción de la tarea.

En términos de investigación aplicada, el problema de investigación refiere al desarrollo de un MR mediante TL para la estimación de concentraciones de material particulado (PM) para la ciudad de Bahía Blanca.

El caso planteado para la ciudad de Bahía Blanca podría tratarse como TL profundo basado en el modelo. Este enfoque reutiliza parcialmente la RN que se entrenó previamente en el dominio de origen, incluida su estructura y sus pesos, para que ellos formen parte de la RN que

se construyo para el dominio de destino. Supone que el mecanismo de procesamiento es similar, y es un proceso de abstracción iterativo y continuo. Las capas frontales de la red se pueden tratar como un extractor de características y las características extraídas son versátiles [TSK+18].

El desarrollo de este conocimiento puede llevarse a cabo con un enfoque orientado al área de Computación Científica. Esto implica el tratamiento riguroso desde dos puntos de vista: ingenieril y computacional. En este contexto, se podría analizar los criterios con los cuales son elegidos - y consecuentemente compatibilizados en un mismo espacio de representación- los dominios de origen y destino. Este es un problema de aplicación de base de datos pequeñas tabulares que apela al TL para la construcción del modelo siguiendo directrices inusuales para este problema. Por lo que resultaría de interés construir un modelo de estimación de PM10 para otra ciudad de referencia a partir de múltiples estaciones. A partir de ese modelo, podrá construirse un modelo para estimar las concentraciones de dicho contaminante para la ciudad de Bahía Blanca.

Construir NN con datos pequeños es un desafío que se vuelve más complejo cuando los son datos tabulares. El uso de emebddings ha reportado grandes avances; sin embargo, algunos embeddings son sensibles al cambio de dominio. Por lo que, para su uso en problemas de TL requerirían mayor capacidad de adaptación a nuevos dominios. La última palabra sobre estos asuntos no se ha dicho y es de esperar nuevos avances.

Además, en esta tesis se han identificado las siguientes oportunidades de investigación futura.

Durante la experimentación de la división del conjunto de datos, se enfocó en los atributos numéricos de las observaciones, sin embargo, es importante destacar los atributos ordinales o categóricos. Una consideración clave se refiere a la necesidad de transformar las variables categóricas según los requisitos del modelo que se esté utilizando. Esta consideración plantea la pregunta de si se necesitan otras funciones de distancia específicas para estos atributos.

En cuanto a la transformación de los datos, es importante destacar que esto implica un cambio en la relación entre la cantidad de observaciones y los parámetros del modelo. Esta relación, como se definió en el Capítulo 2, es lo que determina si se trata de una base de datos pequeña o no. Se plantea la cuestión de si es posible lograr los mismos niveles de generalización con la misma cantidad de observaciones en el conjunto de datos si este contiene variables discretas o

categóricas. En futuros experimentos, también se deberá tener en cuenta cómo la asimetría y otras propiedades estadísticas del conjunto de datos pueden influir en su partición.

En relación con el impacto de la división controlada en la evaluación de los modelos, se examinarán las posibles diferencias en los errores que pudieran presentarse en las últimas etapas del entrenamiento. También se investigará si estos mismos resultados se presentan cuando se divide el conjunto de datos en k partes. Se evaluará si los mismos beneficios reportados en la experimentación propuesta se aplican al considerar solo X o solo y.

Asimismo, el algoritmo de optimización Adam es recomendado para grandes volúmenes de datos, y se evaluará si se obtienen resultados similares a los presentados en la experimentación con el mismo.

Se observaron limitaciones en cuanto a las posibilidades experimentales de los PGD utilizados. Se propondrán opciones de generación que incluyan complejidades progresivas, como distribuciones de probabilidades no uniformes en los atributos de entrada o la inclusión de valores atípicos. Además, se abordarán las relaciones observadas entre los atributos, generando dependencias entre ellos para diseñar PGD que se aproximen a las condiciones de los problemas aplicados.

En relación con el HPO propuesto, uno de los aspectos más importantes es el rendimiento del mismo en comparación con los clásicos y otros más innovadores. En futuros trabajos se comparará si la propuesta tiene mejores resultados para NN que hiperbanda o una optimización bayesiana sobre una biblioteca de referencia para DL en un estudio que va más allá de los problemas de regresión.

Se ha decidido mantener el algoritmo libre de HP propios y, dado que solo tiene un esquema de enfriamiento, se realizará una tarea comparativa de diferentes esquemas en futuros trabajos. La selección adecuada de este parámetro supone un nuevo nivel de complejidad en la selección de los HP correspondientes al modelo final. Por lo tanto, sería apropiado que se incluya la selección de este esquema de forma dinámica.

Como el herrero -que en ocasiones necesita volver a calentar la pieza- algunos problemas podrían requerir recalentamiento. En futuras versiones se evaluará la incorporación de un mecanismo de calentamiento más elaborado al actualmente propuesto. También se replanteará

el manejo explícito del punto inicial, ya que la misma configuración de hiperparámetros puede verse infravalorada o sobrevalorada según este punto. Se realizará un análisis de perturbación con el objetivo de cuantificar la sensibilidad de las NN respecto al punto inicial y cómo esto puede mitigarse.

En líneas generales, considero fundamental la profundización de estudios sobre:

- **desarrollo de un MR mediante TL para la estimación de concentraciones de PM para la ciudad de Bahía Blanca**
- **el desarrollo de embeddings que soporten TL:** algunos embeddings son sensibles al cambio de dominio. Por lo que, para su uso en problemas de TL requerirían mayor capacidad de adaptación a nuevos dominios.
- **el diseño de PGD que se aproximen a escenarios factible:** el diseño de PGD que se aproximen a escenarios factibles posibilitaría el estudio de los modelos en relación con el poder de generalización y proveerían un marco para la estandarización de las evaluaciones de modelos y experimentos. ,
- **algoritmos de optimización específicos para base de datos pequeñas:** sería posible combinar el algoritmo evolutivo como mecanismo de selección de los lotes en el algoritmo de optimización. De esta forma sería un algoritmo específico para base de datos pequeñas.
- **algoritmos de optimización de hiperparámetros:** la incorporación otros mecanismos de enfriamiento y calentamiento manteniendo el mismo libre de hiperparámetros propios.

A. Anexos del Capítulo 2

A.1. Otros tipos de modelos

Propuestas por [Elm90], en las NN recurrentes, las unidades pueden tener conexiones recursivas a través del tiempo, por lo que poseen un estado interno. En estas redes el orden en el cual la experiencia es presentada tiene un impacto en el aprendizaje. En la figura 2.4.2.a se puede observar esta arquitectura. Sin embargo, presenta inconvenientes al momento de su entrenamiento [Hoc98, PMB13, Han18]. Una de las alternativas para sobreponer este inconveniente es la propuesta de las redes de memoria de largo y corto plazo (conocidas por la sigla LSTM - *long short term memory*) [HS97]. En estas redes cada unidad tiene una celda de memoria y tres compuertas: entrada, salida y olvido; la función de estas compuertas es custodiar el flujo de la información deteniendo o permitiendo la misma. Otro modelo recurrente es el de las unidades de compuertas recurrentes (GRU: *Gated Recurrent Unit*) propuestas por [CGCB14]. Esta arquitectura es una variación de las LSTM, dónde las unidades poseen una compuerta menos; las compuertas son llamadas de reinicio y actualización.

Una mención relevante son los mecanismos de atención, originalmente propuestos por [VSP+17]. Estos permiten modelar dependencias sin tener en cuenta su distancia en las secuencias de entrada o salida. Una función de atención se puede describir como un diccionario de pares clave-valor a una salida, con la particularidad de que los valores y la salida son todos vectores. Un mecanismo particular de atención llamado autoatención o intra atención (del inglés, *self-attention* e *intra-attention*, respectivamente) utiliza diferentes posiciones de una sola secuencia para calcular una representación de la misma secuencia.

Aunque existen excepciones en las que se implementan con redes recurrentes [PTDU16] (su principal uso es una arquitectura conocida como transformadores (del inglés, *transformers*) en los cuales la recursión no está presente. El transformador sigue esta arquitectura general usando autoatención apilada y capas puntuales completamente conectadas (análogas a las utilizadas en las MLP).

Propuestos por [BK88], los autocodificadores (AE: Auto Encoders) son similares a las MLP sin embargo poseen una aplicación radicalmente distinta y constituyen una arquitectura diferente. La idea básica de los AE es codificar la información, comprimirla automáticamente y de allí su nombre. La arquitectura de la red puede ser vista como un reloj de arena, donde las capas ocultas o medias poseen menor cantidad de unidades que la capa de entrada y salida; es en estas capas intermedias dónde la información es comprimida. Los AE se pueden construir de forma simétrica, de tal manera que los pesos que comunican la capa de entrada con las capas intermedias son los mismos que comunican esta última con la capa de salida.

Los AE dispersos (AED), pueden ser considerados como opuestos a los AE, utilizadas para extraer muchas características pequeñas de un conjunto de datos [RPCC06]. En lugar de aprender cómo representar un conjunto de experiencia en un espacio reducido de unidades, se trata de codificar esta información en un mayor espacio.

Por último, de arquitectura similar a los AE, los autocodificadores variacionales (VAE: Variational AutoEncoder), tienen un objetivo diferente: aproximar una distribución de probabilidad a partir de la experiencia de entrada dada [KW13].

Los modelos adversarios propusieron en su momento un nuevo marco para estimar modelos generativos a través de un proceso competitivo similar a un juego [GPAM$^+$14].

Se entrenan simultáneamente dos modelos: un modelo generativo G y un modelo discriminativo D. Los autores presentan este ejemplo para ilustrar la forma en la que estos modelos son entrenados: el modelo generativo puede considerarse análogo a un equipo de falsificadores que intenta producir una moneda falsa y utilizarla sin ser detectada, mientras que el modelo discriminativo es análogo a la policía, que intenta detectar esta. La competencia en este juego impulsa a ambos equipos a mejorar sus métodos hasta que las falsificaciones son indistinguibles de los artículos genuinos.

Técnicamente, se entrenan simultáneamente dos modelos: un modelo generativo G que captura la distribución de datos, y un modelo discriminativo D que estima la probabilidad de que una muestra provenga de los datos de entrenamiento en lugar de G. El objetivo de G es maximizar la probabilidad de que D cometa un error. Mientras que el objetivo de D es determinar si una muestra es de la distribución del modelo G o de la distribución de datos.

Se definen G y D mediante dos MLP interconectadas. Todo este conjunto puede entrenarse con los mecanismos usuales de las redes perceptrón multicapa.

A.1.1. Otros tipos de entrenamiento

Es usual tener grandes cantidades de datos de entrenamiento sin etiquetar y relativamente pocos datos de entrenamiento etiquetados, el aprendizaje semi-supervisado ofrece la oportunidad de aprender también de los datos no etiquetados [GBC16].

Una de las técnicas relevantes de aprendizaje no supervisado es el pre-entrenamiento *greedy* de las capas (del inglés, *greedy layer-wise unsupervised pretraining*).

Esta técnica recibe el nombre de greedy porque supone un pre entrenamiento capa a capa, optimizando cada una de ellas de forma independiente, de una a la vez, en lugar de optimizar todas las piezas de forma conjunta. Avanza entrenando la capa k-ésima mientras se mantienen fijas las anteriores. Se considera no supervisado porque cada capa se entrena con un algoritmo de esta característica.

Se denomina pre entrenamiento porque generalmente es un primer paso antes de que se aplique un algoritmo de entrenamiento conjunto para ajustar con precisión todas las capas juntas. Esta primera fase -en el contexto del aprendizaje supervisado- es una forma de inicialización de parámetros. La fase de aprendizaje supervisado puede implicar un ajuste fino supervisado de toda la red aprendida anteriormente.

Este esquema de entrenamiento general es independiente del algoritmo aprendizaje no supervisado o del tipo de modelo se emplee. En muchas tareas, esta técnica puede producir mejoras sustanciales para las tareas de clasificación [RPCC06, BLPL06] . En muchas otras, sin embargo, no confiere un beneficio o incluso causa un daño notable.

A.2. Recursos utilizados en esta tesis

A.2.1. Lenguajes de programación

El lenguaje de programación utilizado para todo el desarrollo de los temas tratados en esta tesis es Python. Es un lenguaje de propósito general, interpretado y de alto nivel, con un

sistema de tipo dinámico y con un recolector de basura integrado (del inglés: *Garbage Collector*). Se destaca su legibilidad lograda a través del uso de sangría significativa. Además, admite múltiples paradigmas de programación, incluida la programación estructurada (particularmente procedimental), orientada a objetos y funcional.

Las librerías más relevantes[1] utilizadas para el DL están realizadas en este lenguaje.

A.2.2. Librerías utilizadas

Se mencionan las librerías más importantes utilizadas como pilar de los algoritmos y experimentos implementados:

- DEAP[2] (Distributed Evolutionary Algorithms in Python): framework de computación evolutiva para la creación rápida de prototipos y pruebas de ideas. El objetivo que persigue es hacer que los algoritmos sean explícitos y sus estructuras de datos comprensibles al momento de la implementación.
- Tensorflow[3]: framework de código abierto de extremo a extremo para el ML. Posee un ecosistema integral y flexible de herramientas, bibliotecas y recursos comunitarios que permiten a los investigadores impulsar los límites del estado del arte en ML, facilitando la extensión de estos límites.
- SKLEARN: Simple and efficient tools for predictive data analysis y el los generadores de los datasets
- Librerías para el desarrollo de gráficos: Seaborn

A.2.3. Entornos de desarrollo

Los entornos de desarrollo se dividen en dos grupos: los utilizados para el desarrollo local y el utilizado para la ejecución en la nube.

- Local

[1] Estas librerías son PyTorch (https://pytorch.org) y Tensorflow (https://www.tensorflow.org).
[2] DEAP: https://deap.readthedocs.io/en/master/
[3] Tensorflow (https://www.tensorflow.org)

- Spyder[4]: entorno científico gratuito y de código abierto escrito en Python para Python y diseñada para usos científicos, ingenieriles y análisis de datos.
- Pycharm[5]: entorno de desarollo profesional para proporcionar las herramientas necesarias para el desarrollo con fines comerciales.

- No local: Google Colab. Çolab.[es] un producto de Google Research que facilita la escritura y la ejecución de código en Python en el navegador. Particularmente apropiado para tareas de aprendizaje automático. Técnicamente hablando, Colab es un servicio de cuaderno alojado de Jupyter Notebooks que no requiere configuración. Se usó su versión Pro, que facilita el acceso a máquinas virtuales con alta capacidad de memoria y hardware específico (en sus versiones GPU T4 o P100) para acelerar el desarrollo de modelos de DL.

[4] *https://www.spyder-ide.org*
[5] https://www.jetbrains.com/pycharm

B. Anexos del Capítulo 3

B.1. Esquema de algoritmo evolutivo

Algoritmo evolutivo. Elaboración propia con base en Perez Serrada (1990)

```
Algoritmo Evolutivo
Datos de entrada:
   problema
Datos de salida:
      población: lista de individuos
Datos auxiliares
      generación: número entero
      población: lista de individuos
   descendencia: lista de individuos
   aptitud_población: lista de números reales
   aptitud_descendencia: lista de números reales
Comienzo
      generacion <- 0
      población <- InicializarPoblacion(problema)
      aptitud_poblacion <- EvaluarFitness(población)
      para t en el rango(0, max_generación) hacer:
            descendencia <- Seleccionar reproductores(población)
            descendencia <- Transformar(descendencia)
            aptitud_descendencia <- EvaluarFitness(descendencia)
            poblacion <- SeleccionSobrevivientes(población,
                                 aptitud_poblacion,
                                 descendencia,
                                 aptitud_descendencia)
            aptitud_poblacion <- EvaluarFitness(población)
```

Si CumpleConCondiciónTerminación(población,

aptitud_poblacion)

entonces retornar(población)

Fin del algoritmo

B.2. Esquema del algoritmo evolutivo propuesto

Esquema del algoritmo evolutivo propuesto.

Algoritmo Evolutivo

Datos de entrada:

ConjuntoDeDatos

Particion

Distancia

TamañoPoblacion

CantGeneraciones

ProbabilidadCruce

ProbMutacion

Datos de salida:

Campeon

Datos auxiliares

Poblacion

Descendencia

individuo_1, individuo_2

Comienzo

Crea una población de soluciones aleatorias

poblacion <- InicializarPoblacion(ConjuntoDeDatos, Particion)

campeon <- Ninguno

```
Para g en el rango(1,CantGeneraciones):

    # Selecciona un subconjunto de soluciones de la población actual para reproducirse.
    # Las soluciones con mejores calificaciones tienen una mayor
    # probabilidad de ser seleccionadas.
  desendencia <- Clonar(poblacion)

  # Genera nuevas soluciones mediante la combinación de las
  # soluciones seleccionadas. Esto se puede hacer utilizando
  # operadores de cruce y mutación.
  para individuo_1, individuo_2 en desdendencia:
      Si numero aleatorio < ProbabilidadCruce:
          cruzar(individuo_1, individuo_2)

    probMutacion = 1 - (0.95 * ( g/ CantGeneraciones)
    para individuo en desdendecia:
        Si numero aleatorio < probMutacion:
            mutar(individuo)

    Descendencia <- Mejores individuos (descendencia)
    Si el mejor individuo Población <      Campeón
        # nuevo campeón
        Campeón <- Nueva mejor solución

    # Reemplazo total de la población con las nuevas soluciones generadas
    Poblacion <- Desdendencia

retornar Campeón

Fin del algoritmo
```

B.3. Esquema del algoritmo de búsqueda aleatoria

Esquema del algoritmo evolutivo propuesto.

```
Algoritmo Evolutivo
Datos de entrada:
    ConjuntoDeDatos
    Particion
    Distancia
    CantRepeticiones
Datos de salida:
    Campeon
Datos auxiliares
    Poblacion
    Descendencia
    individuo_1, individuo_2

Comienzo
    # Crea una población de soluciones aleatorias
    poblacion <- InicializarPoblacion(ConjuntoDeDatos, Particion)

    campeon <- Ninguno
    aptitud_campeon <- +Infinito

    Para g en el rango(1,CantGeneraciones):

        #Genera un nuevo estado aleatorio
        candidato <- generar_particion_aleatoria(ConjuntoDatos, Particion)

        # Evalúa el nuevo estado generado
        Si(Distancia= "Mahalanobis")
            entonces aptitud_candidato <- aptitudMahalanobis(Candidate,
```

ConjuntoDatos)

sino aptitud_candidato <- aptitudWassertein(Candidate,

ConjuntoDatos)

#Si el nuevo candidato es mejor que el campeón actual,

#actualiza con la nueva partición generada

#

Si(aptitud_candidato < aptitud_campeon):

entonces aptitud_campeon <- aptitud_candidato

Campeón <- candidato

retornar Campeón

Fin del algoritmo

B.4. Implementación realizada

```
#imports


import random
import pandas as pd
import numpy as np
from math import inf
from numpy.linalg import inv
from deap import base
from deap import creator
from deap import tools


from scipy.stats import wasserstein_distance
import pickle
```

```
class GASplitter:
    def __init__(self, dataset, partition, distance):
        #que pasa si hay un partition[i] == 0 ?
        #dataset.shape
        if(sum(partition) != 1.0):
            raise ValueError('Partition sum must be 1')
        else:
            self.partition = partition
            self.par_elements = np.ones(len(partition))
            elements_left = dataset.shape[0]
            for e in range(len(partition)):
                elements_to_assign = int(round(self.partition[e] * dataset.
                if(elements_left > elements_to_assign):
                    self.par_elements[e] = int(elements_to_assign)
                    elements_left -= elements_to_assign
                #debería ser el caso de la ultima iteracion
                #y en casos especiales
                else:
                    self.par_elements[e] = int(elements_left)
                    elements_left -= elements_left
                if((elements_left > 0) and (e == len(partition)-1)):
                    self.par_elements[e] += int(elements_left)
            self.par_elements = list(map(int, self.par_elements))
            if(sum(self.par_elements) != dataset.shape[0]):
                raise ValueError('Elements without assingment')

        self.distance = distance
```

```
        self.logbook = None
        self.dataset = dataset.copy(deep=True)
        self.dataset_cov = np.cov(self.dataset.values.T)
        self.dataset_mean = self.dataset.values.mean(axis=0)
        self.popsize= 30
        self.pop = None
        self.NGEN = 50
        #probabiliadad de cruce
        self.CXPB = 0.60
        #propabilidad de mutación
        self.MUTPB = 0.0333
        self.champion = None

    def correct(self, ind, position, par_prior, par_posterior ):

        if(position == 1):
            indBefore_position = np.array(ind[0])
        else:
            indBefore_position = np.array(ind[0:position])
        indAfter_position = np.array(ind[position:len(ind)])

        idxBefore_position = np.where(indBefore_position == par_prior)
        idxAfter_position = np.where(indAfter_position == par_posterior

        idxBefore_position = random.sample(list(idxBefore_position[0]),
        idxAfter_position = random.sample(list(idxAfter_position[0]), k
        idxAfter_position = [e + position for e in idxAfter_position]
        idxBefore_position = idxBefore_position[0]
        idxAfter_position = idxAfter_position[0]
```

```
        ind[idxBefore_position] = par_posterior
        ind[idxAfter_position] = par_prior

        return(ind)

    def cxSpecialOnePoint(self, ind1, ind2):

        a_father =  ind1.copy()
        other_father = ind2.copy()

        cxpoint = random.randint(1, len(ind1)-1)
        part_size = len(ind1[cxpoint:len(ind1)])
        partsCountInd1 = np.zeros(len(self.par_elements))
        partsCountInd2 = np.zeros(len(self.par_elements))

        #Cuento los elementos que estan en cada particion
        for i in range(cxpoint, len(ind1)):
            partsCountInd1[ind1[i]] += 1
            partsCountInd2[ind2[i]] += 1

        #who is going to be correct
        who = random.randint(0,99)
        if(who % 2 == 0):
            ind = ind1
            changes = partsCountInd1 - partsCountInd2

        else:
            ind = ind2
```

```
        changes = partsCountInd2 - partsCountInd1

    repair = np.where(changes != 0)

    while(np.not_equal(repair,0).any()):

        more_partition = (np.where(changes>0))[0][0]
        less_partition = (np.where(changes<0))[0][0]

        ind = self.correct(ind, cxpoint, less_partition,
         more_partition)

        changes[more_partition] = changes[more_partition] - 1
        changes[less_partition] = changes[less_partition] + 1

        repair = np.where(changes !=0)

    if(who % 2 == 0):
        ind1 = ind
    else:
        ind2 = ind

    ind1[0:cxpoint], ind2[0:cxpoint] \
        = ind2[0:cxpoint].copy(), ind1[0:cxpoint].copy()

    if(np.array_equal(a_father, ind1)):
        self.toolbox.mutate(ind1)

    if(np.array_equal(other_father, ind2)):
```

```
            self.toolbox.mutate(ind2)

        return ind1, ind2

    def mahalanobisBetweenPopulations(self, mu1,mu2, cov1, cov2):
        #Are the same population?
        if(np.equal(mu1,mu2).all() and np.equal(cov1,cov2).all()):
            return float(0)

        #Different populations
        difMedias = mu1 - mu2
        difMedias = np.reshape(difMedias,
         (1,difMedias.shape[0]))
        difCovarianzas = 0.5 * (cov1 + cov2)
        mahalanobisDist = np.matmul(np.matmul(difMedias,
           inv(difCovarianzas)), difMedias.T)
        return np.sqrt(float(mahalanobisDist))

    def fitnessMahalanobis(self, partition):
        n_partitions = max(partition)
        distance = 0

        for i in range(n_partitions+1):
            partition_filter = pd.DataFrame(partition) == i
            partition_data = self.dataset
             .loc[partition_filter.values.T[0]]
            partition_mean = partition_data.values.mean(axis=0)
            partition_cov = np.cov(partition_data.values.T)
            distance += self.mahalanobisBetweenPopulations(
```

```
            self.dataset_mean,
            partition_mean,
            self.dataset_cov, partition_cov)
        return (distance, )

    def fitnessWassertein(self, partition):
        n_partitions = max(partition)
        distance = 0

        for i in range(n_partitions+1):
            partition_filter = pd.DataFrame(partition) == i
            partition_data = self.dataset
             .loc[partition_filter.values.T[0]]
            columns = self.dataset.shape[1]
            for c in range(columns):
                distance += wasserstein_distance(
                 self.dataset.values[:,c],
                 partition_data.values[:,c])
        return (distance, )

    def init(self):

        toolbox = base.Toolbox()
        self.toolbox = toolbox

        #creo la secuencia de indices
        sequence = []
        for s in range(len(self.partition)):
            sub_sequences = np.ones(int(self.par_elements[s])) * s
```

```
            sub_sequences = map(int,sub_sequences)
            sequence.extend(sub_sequences)

        #Creo el individuo
        creator.create("FitnessMin", base.Fitness, weights=(-1.0, ))
        creator.create("Individual", list, fitness=creator.FitnessMin)
        #range(IND_SIZE) debe ser reemplazado por el con la cadena
        # fija de indices e ind_size por la cantidad de columnas
        self.toolbox.register("indices",
          random.sample, sequence, len(sequence))
        self.toolbox.register("individual",
         tools.initIterate, creator.Individual,
         self.toolbox.indices)
        # define the population to be a list of individuals
        self.toolbox.register("population", tools.initRepeat,
         list, toolbox.individual)
        self.toolbox.register("mate", self.cxSpecialOnePoint)
        self.toolbox.register("mutate", tools.mutShuffleIndexes,
         indpb=(1/self.dataset.shape[0]))
        self.toolbox.register("select", tools.selTournament, tournsize=2)
        if(self.distance == 'mahalanobis'):
            self.toolbox.register("evaluate", self.fitnessMahalanobis)
        else:
            self.toolbox.register("evaluate", self.fitnessWassertein)

    def fit(self):

        self.init()
```

```
self.pop = self.toolbox.population(n= self.popsize)

# Elitisim
self.champion = float('inf')
if(True):

    stats = tools.Statistics(
     key=lambda ind: ind.fitness.values)
    stats.register("min", np.min)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("max", np.max)

    self.logbook = tools.Logbook()

    #La mutacion varia con las generaciones

    for g in range(self.NGEN):
        # Select the next generation individuals
        offspring = self.toolbox.select(self.pop,
         len(self.pop))

        # Clone the selected individuals
        offspring = list(map(self.toolbox.clone, offspring))
        #print(offspring)

        # Apply crossover on the offspring
        for child1, child2 in zip(offspring[::2],
         offspring[1::2]):
```

```
    if random.random() < self.CXPB:
        self.toolbox.mate(child1, child2)
        del child1.fitness.values
        del child2.fitness.values

# Apply mutation on the offspring
self.MUTPB = 1 - (0.95 * (g/self.NGEN))
for mutant in offspring:
    if random.random() < self.MUTPB:
        self.toolbox.mutate(mutant)
        del mutant.fitness.values

# Evaluate the individuals with an invalid fitness
invalid_ind = [ind for ind in offspring
 if not ind.fitness.valid]
fitnesses = self.toolbox.map(self.toolbox.evaluate,
  invalid_ind)
for ind, fit in zip(invalid_ind, fitnesses):
    ind.fitness.values = fit

# Elitisim
fitnesses = self.toolbox.map(self.toolbox.evaluate,
 offspring)
for ind, fit in zip(offspring, fitnesses):
    if((ind.fitness.values)[0] < self.champion):
        self.champion = (ind.fitness.values)[0]
        self.champion_ind = ind

worst_ind =0
```

```
                j = 0
                for ind, fit in zip(offspring, fitnesses):
                    if((ind.fitness.values)[0] >
                      ((offspring[worst_ind]).fitness.values)[0]):
                        worst_ind = j
                    j = j + 1

                # The population is entirely replaced by the offspring
                self.pop[:] = offspring
                self.pop[worst_ind] = self.champion_ind

                # Saving statistics
                record = stats.compile(self.pop)
                record["MUTPB"] = self.MUTPB
                #print(record)
                self.logbook.record(gen=g, **record)
            print(record)

def split(self):
    if(self.champion == None):
        self.fit()
    return self.champion_ind
```

C. Anexos del Capítulo 4

C.1. Curvas de Aprendizaje

Una curva de aprendizaje se utiliza para evaluar el rendimiento de un modelo a medida que se entrena con más datos. Se traza un gráfico en el que:

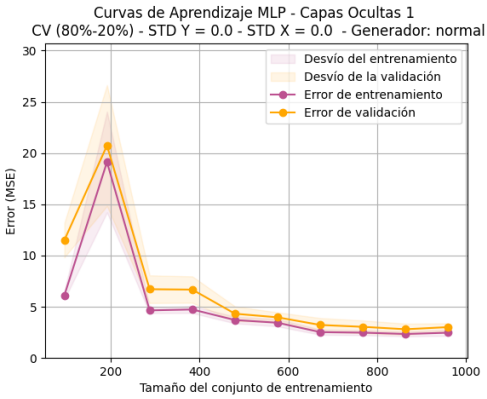
- El eje X suele representar la cantidad de datos de entrenamiento.
- El eje Y suele mostrar algún tipo de medida de error o precisión.

La interpretación depende del contexto en el que se esté utilizando la curva, pero en el caso de aprendizaje automático, se buscan generalmente dos cosas:

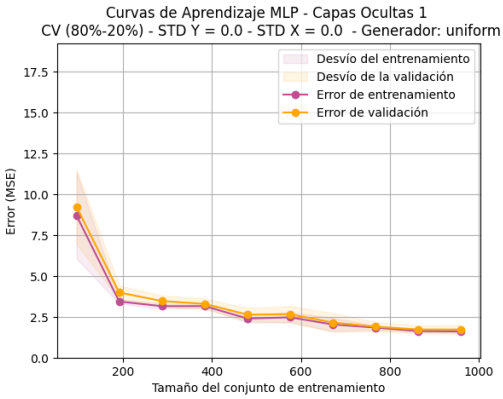
- Convergencia: Si el error en el conjunto de entrenamiento y el conjunto de validación tiende a estabilizarse o converger a medida que se agregan más datos, esto puede indicar que agregar aún más datos de entrenamiento podría no ser beneficioso.
- Brecha entre entrenamiento y validación: Si hay una gran brecha entre los errores del conjunto de entrenamiento y de validación, esto puede indicar un problema de sobreajuste (el modelo se desempeña bien en los datos de entrenamiento pero no en datos no vistos). Por otro lado, si ambos errores son altos y cercanos entre sí, puede indicar un problema de subajuste (el modelo es demasiado simple para los datos y no está aprendiendo bien).

C.1.1. GPD naïve:simple sin ruido

Friedman 1

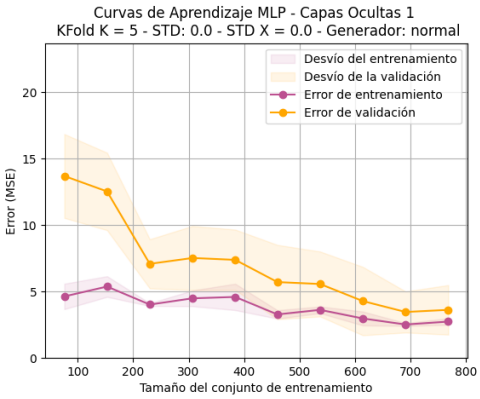


(a) Distribución normal

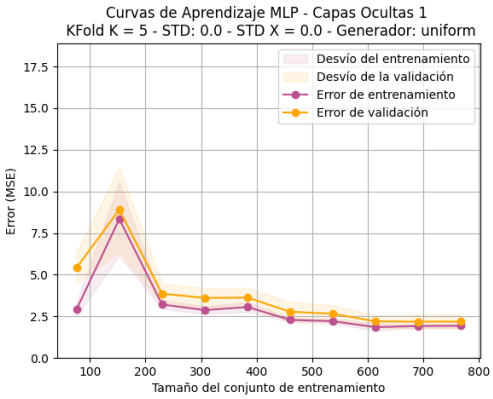


(b) Distribución uniforme

Figura C.1: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

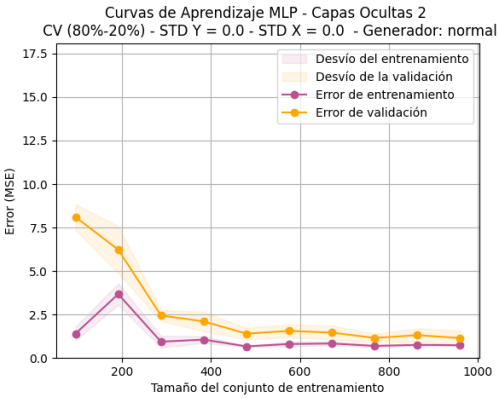


(a) Distribución normal

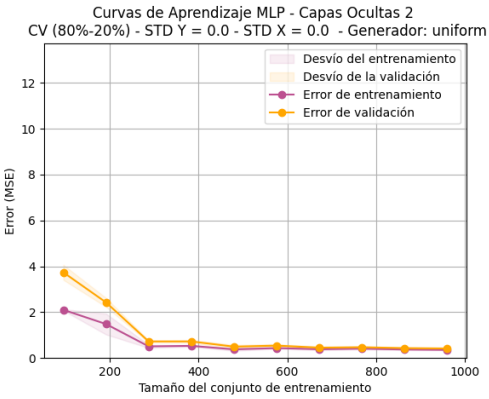


(b) Distribución uniforme

Figura C.2: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

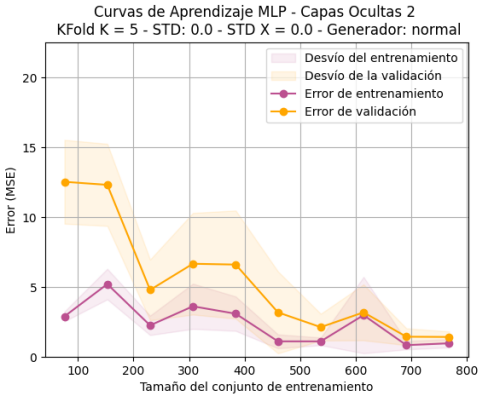


(a) Distribución normal

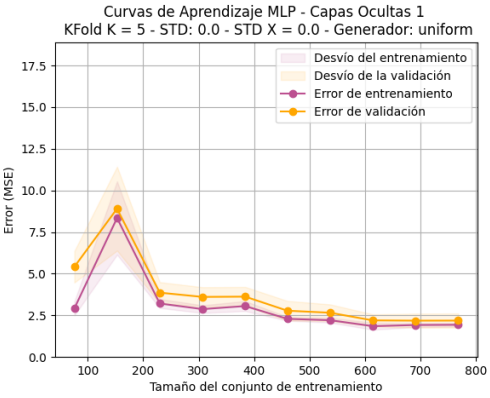


(b) Distribución uniforme

Figura C.3: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

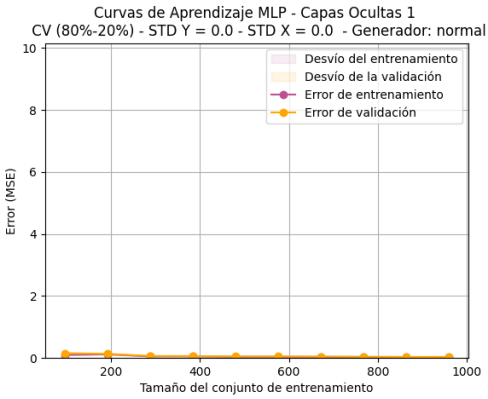


(a) Distribución normal

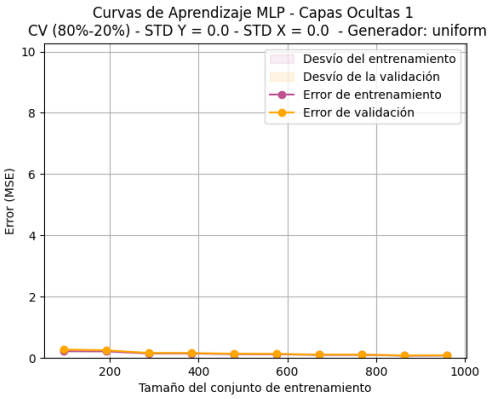


(b) Distribución uniforme

Figura C.4: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

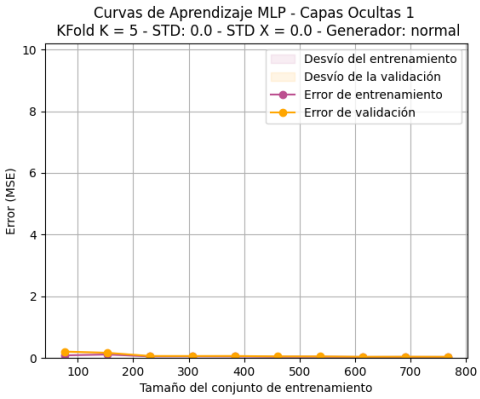


(a) Distribución normal

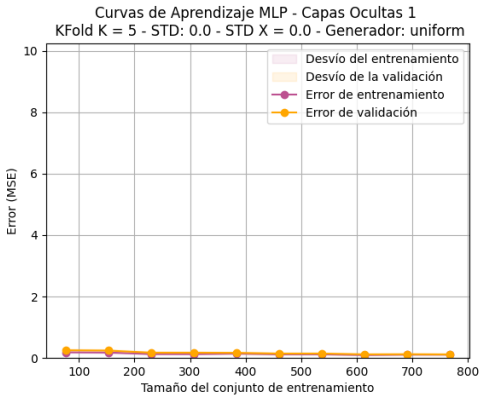


(b) Distribución uniforme

Figura C.5: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

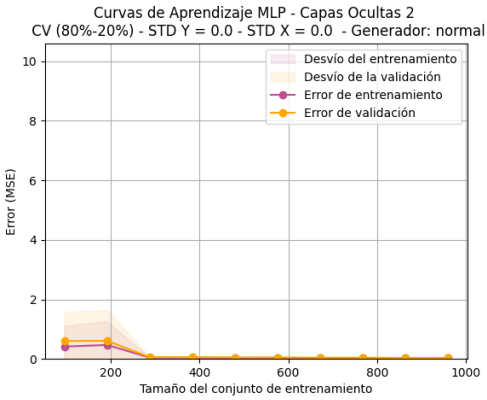


(a) Distribución normal

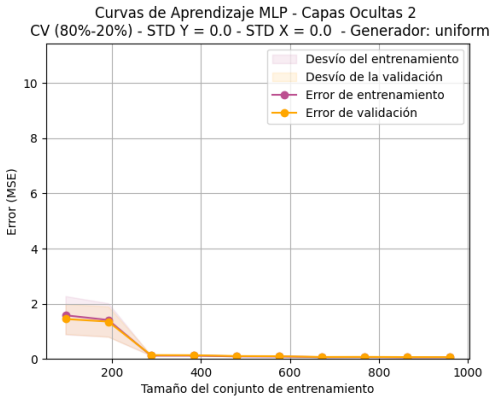


(b) Distribución uniforme

Figura C.6: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

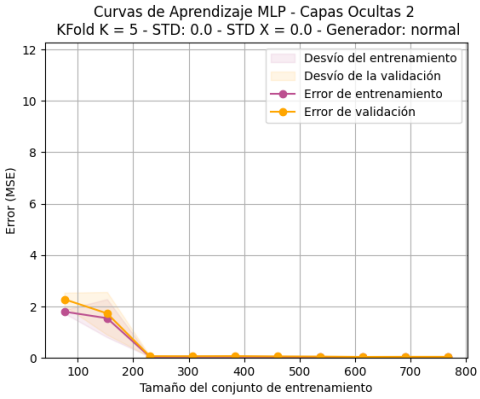


(a) Distribución normal

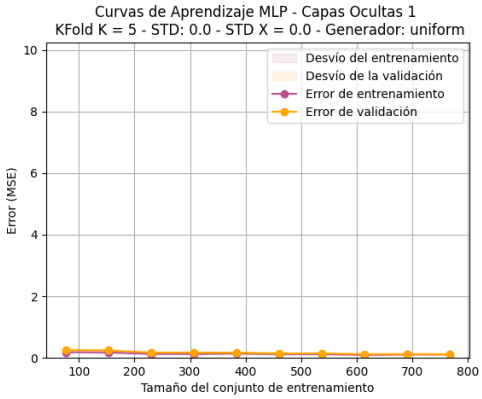


(b) Distribución uniforme

Figura C.7: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

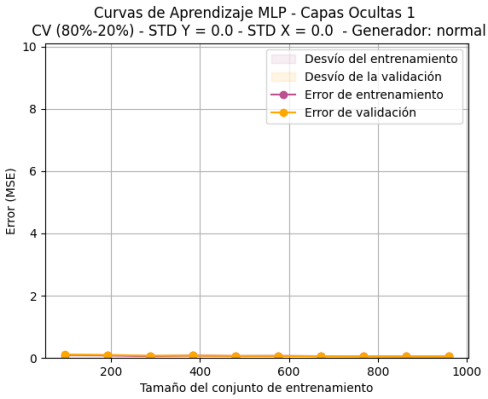


(a) Distribución normal

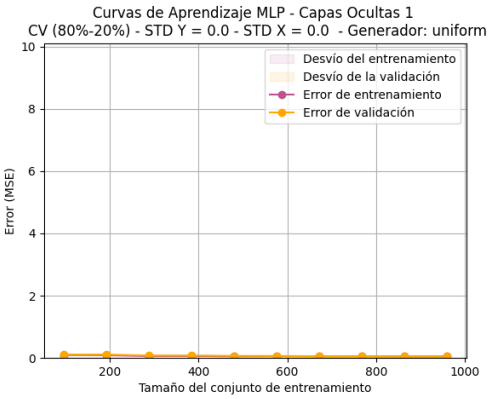


(b) Distribución uniforme

Figura C.8: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

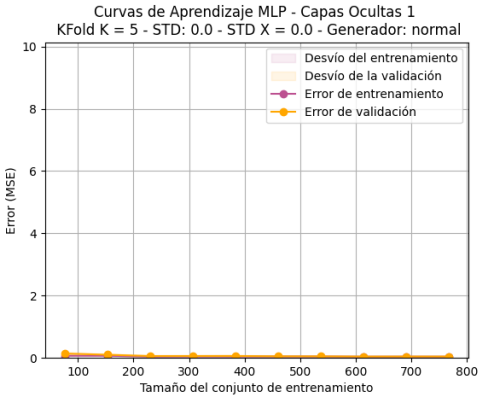


(a) Distribución normal

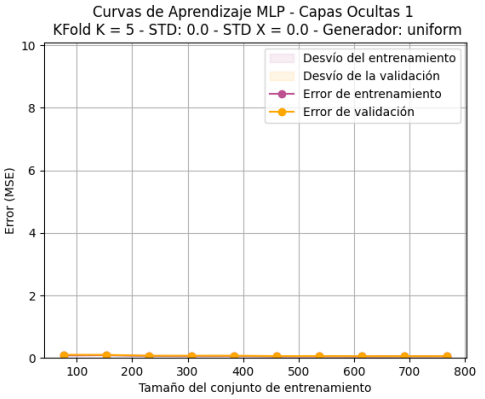


(b) Distribución uniforme

Figura C.9: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

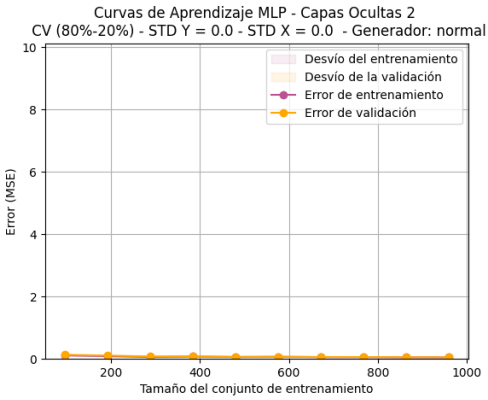


(a) Distribución normal

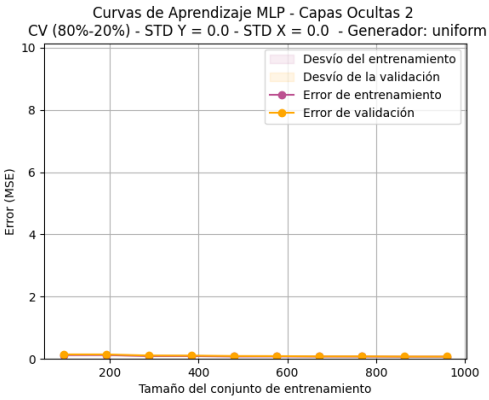


(b) Distribución uniforme

Figura C.10: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

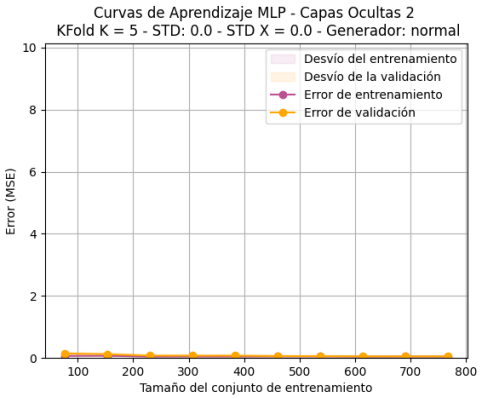


(a) Distribución normal

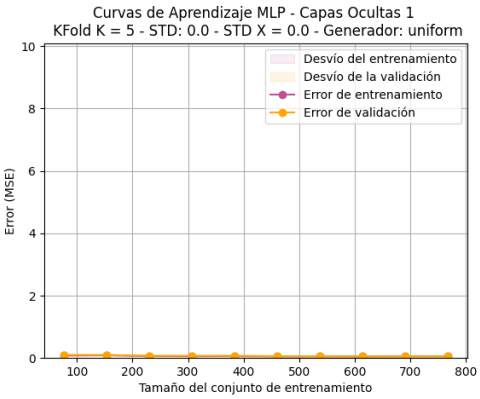


(b) Distribución uniforme

Figura C.11: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

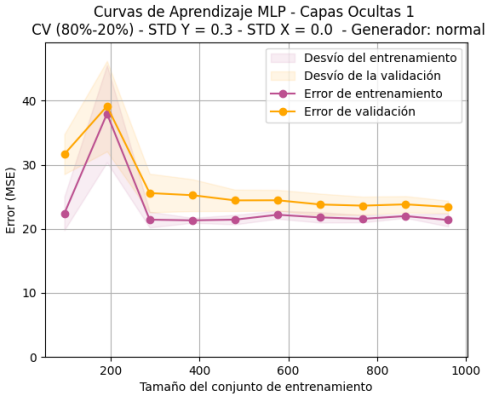


(a) Distribución normal

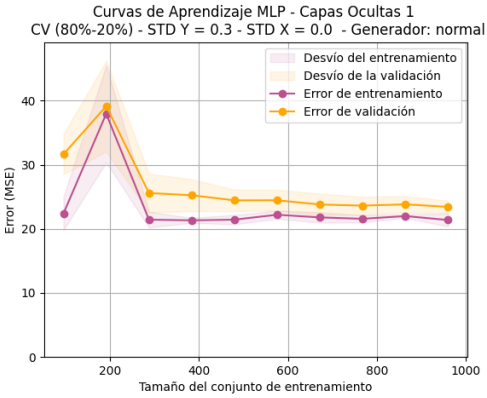


(b) Distribución uniforme

Figura C.12: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

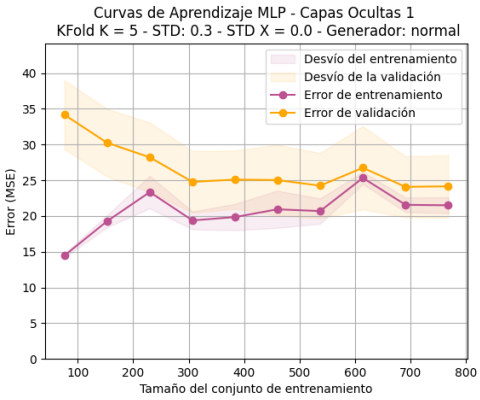


(a) Distribución normal

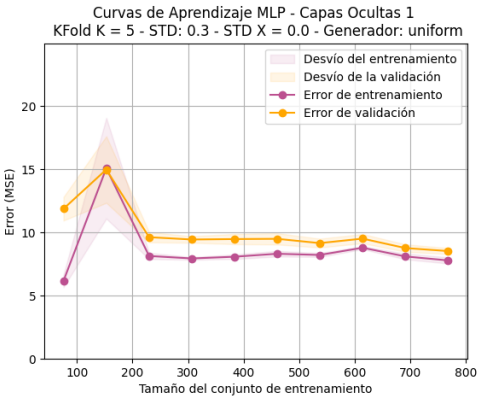


(b) Distribución uniforme

Figura C.13: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

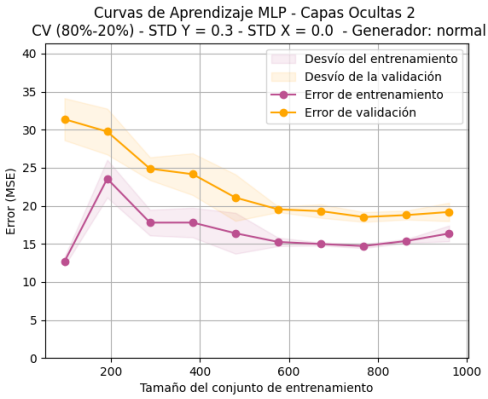


(a) Distribución normal

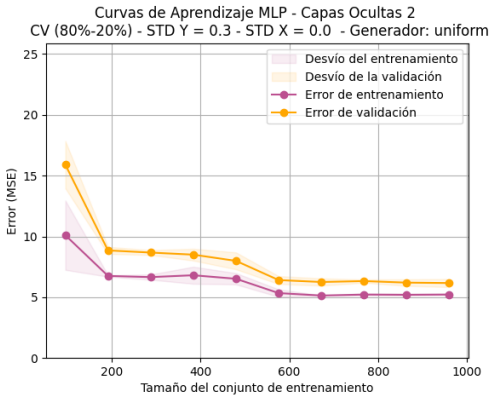


(b) Distribución uniforme

Figura C.14: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

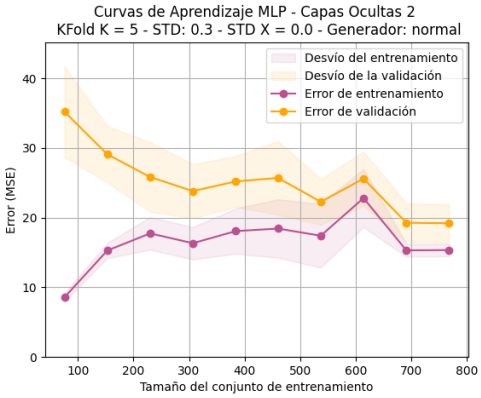


(a) Distribución normal

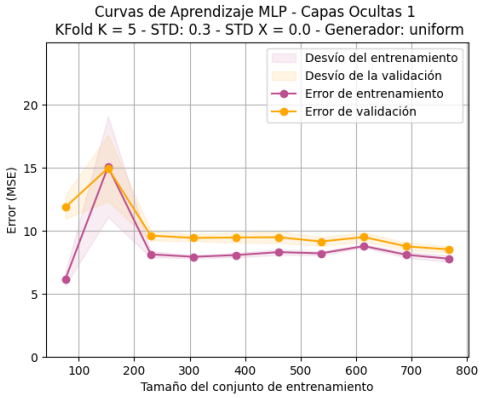


(b) Distribución uniforme

Figura C.15: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.



(a) Distribución normal



(b) Distribución uniforme

Figura C.16: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

Friedman 2

Friedman 3

C.1.2. GPD con ruido en $\hat{f}$

Friedman 1

Friedman 2

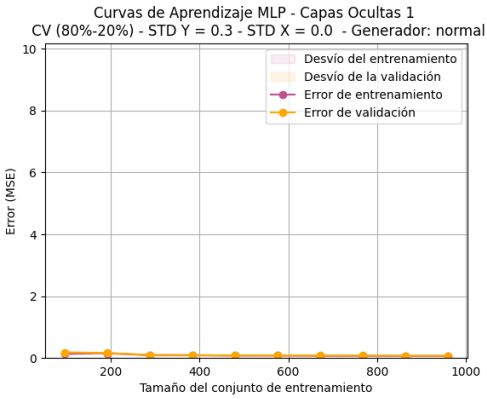
Friedman 3

C.1.3. GPD con ruido en X y en $\hat{f}$

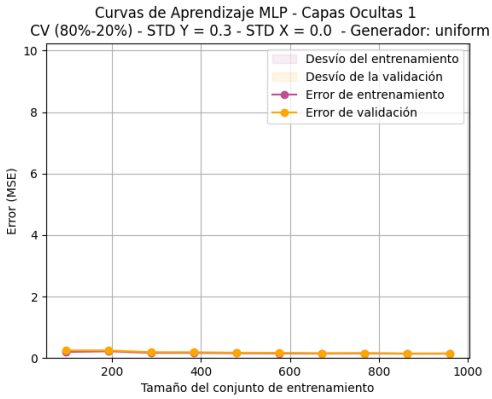
Friedman 1

Friedman 2

Friedman 3

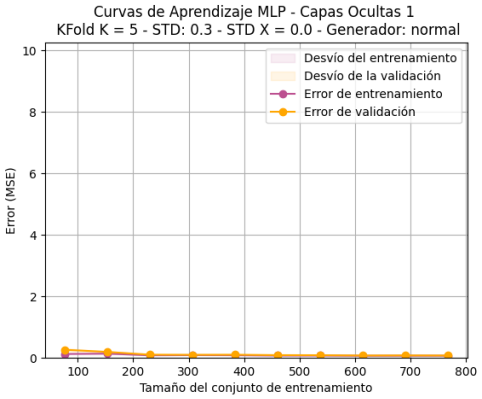


(a) Distribucion normal

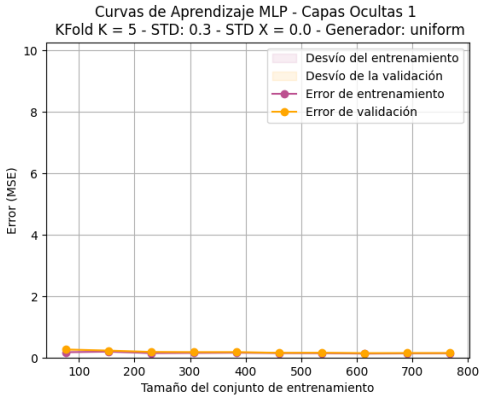


(b) Distribucion uniforme

Figura C.17: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

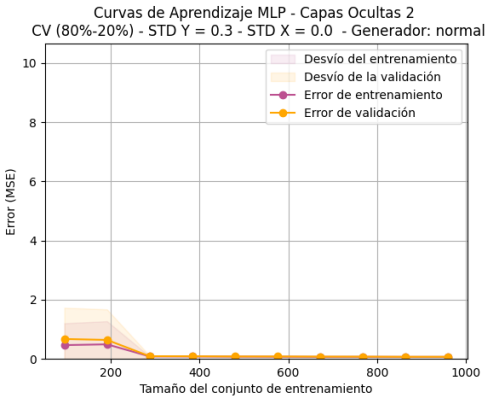


(a) Distribucion normal

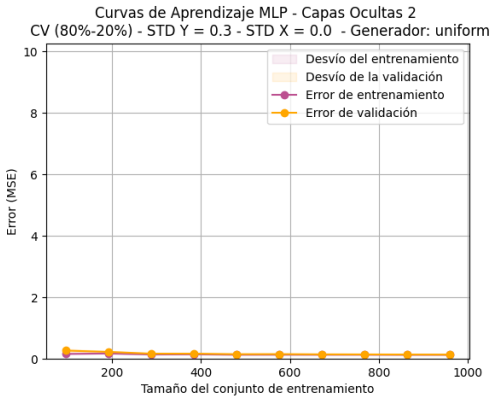


(b) Distribucion uniforme

Figura C.18: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta.

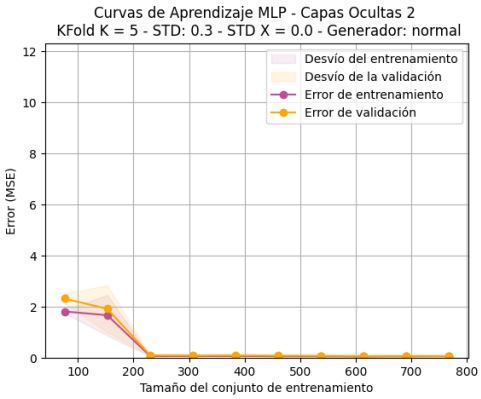


(a) Distribucion normal

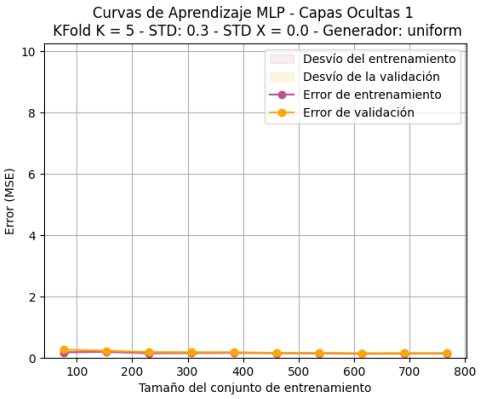


(b) Distribucion uniforme

Figura C.19: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

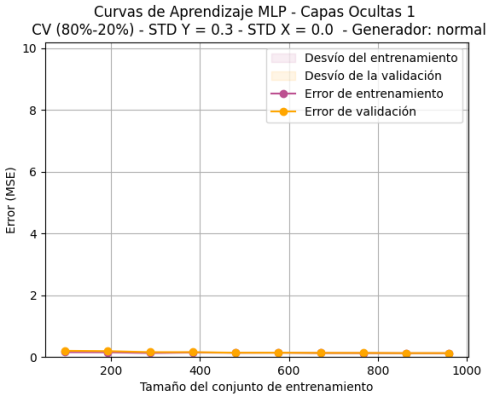


(a) Distribucion normal

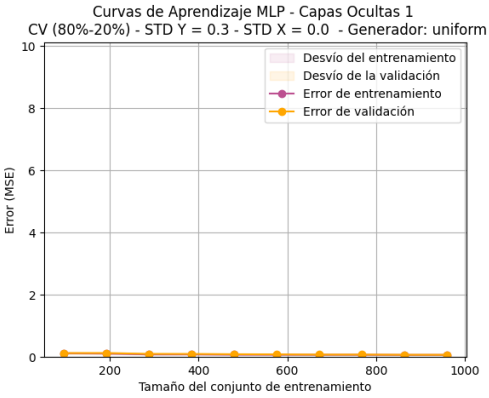


(b) Distribucion uniforme

Figura C.20: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta.

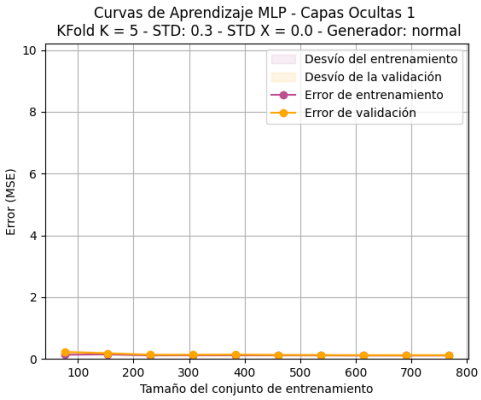


(a) Distribucion normal

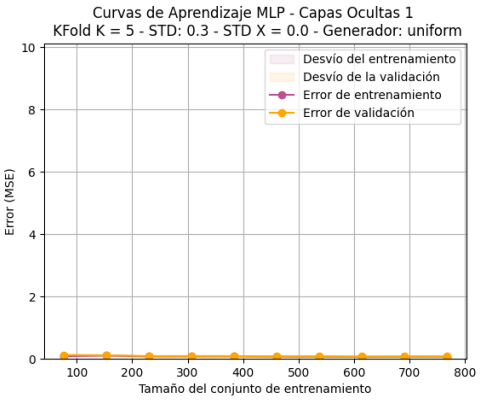


(b) Distribucion uniforme

Figura C.21: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

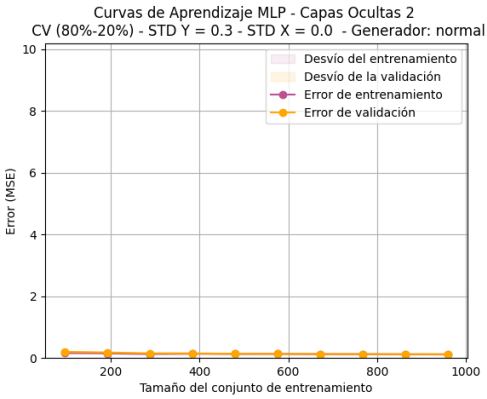


(a) Distribucion normal

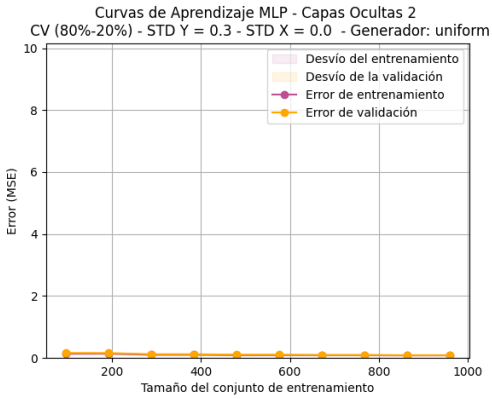


(b) Distribucion uniforme

Figura C.22: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta.

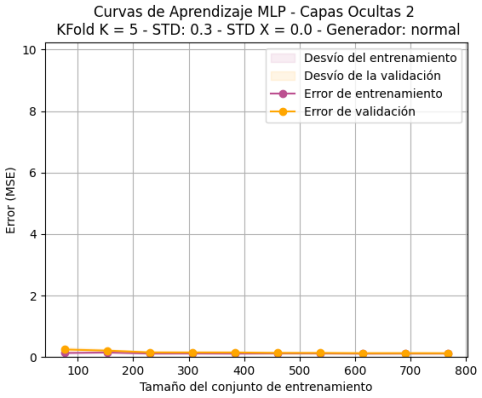


(a) Distribucion normal

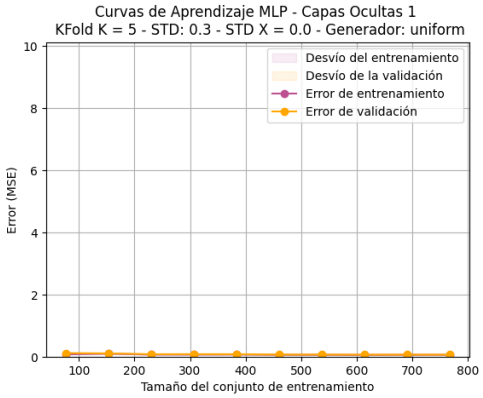


(b) Distribucion uniforme

Figura C.23: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

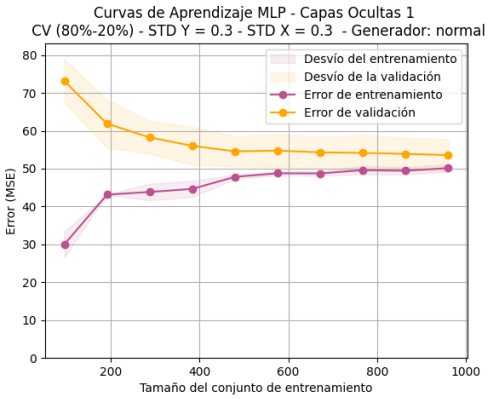


(a) Distribucion normal

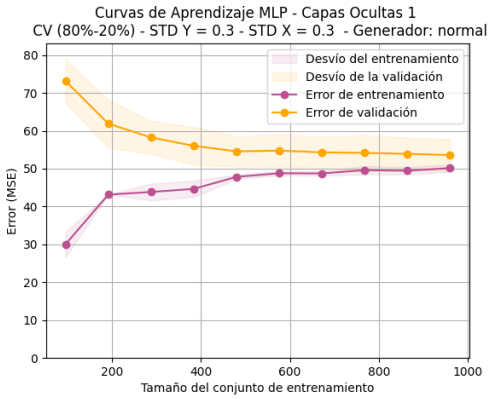


(b) Distribucion uniforme

Figura C.24: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta.

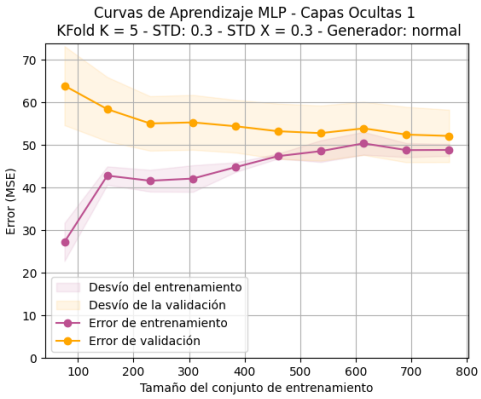


(a) Distribución normal

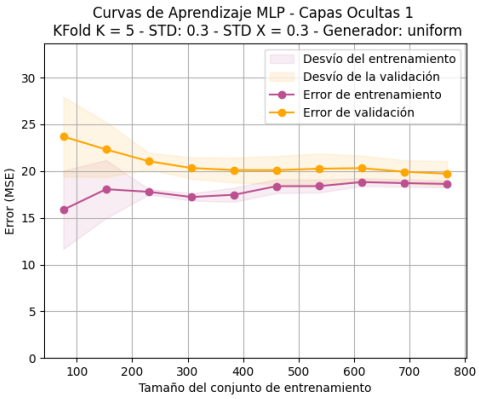


(b) Distribución uniforme

Figura C.25: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

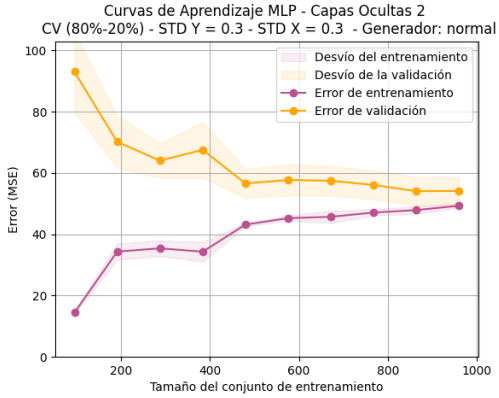


(a) Distribución normal

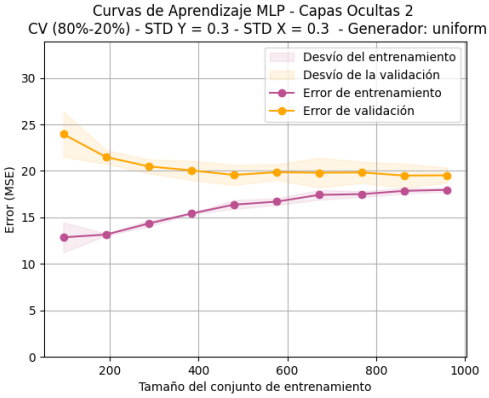


(b) Distribución uniforme

Figura C.26: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con una capa oculta.

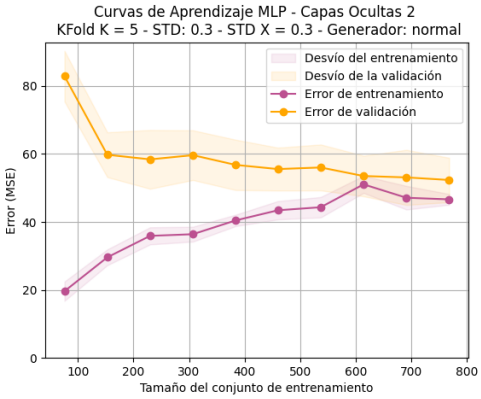


(a) Distribución normal

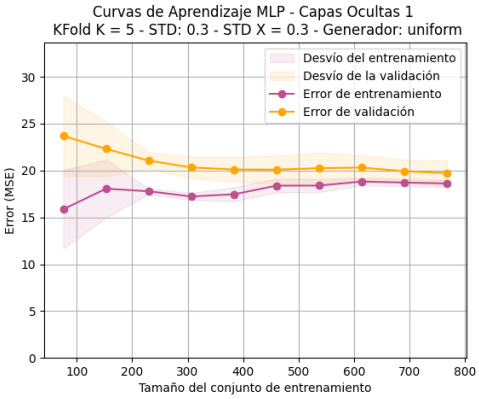


(b) Distribución uniforme

Figura C.27: CV - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

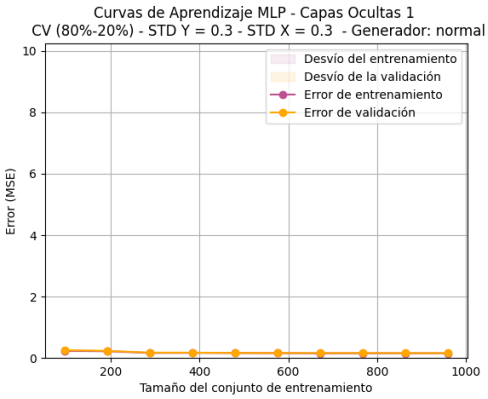


(a) Distribución normal

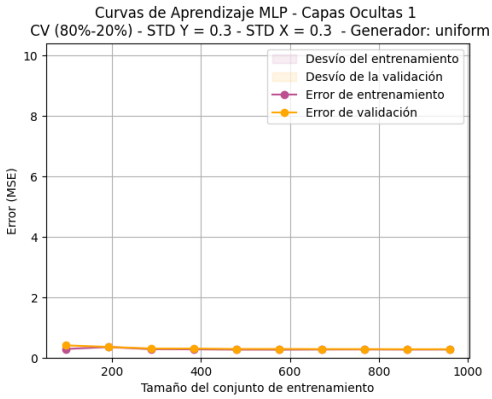


(b) Distribución uniforme

Figura C.28: KF - Comparación del desempeño de una MLP con distintas Distribuciónes de los datos en el PGD F1, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

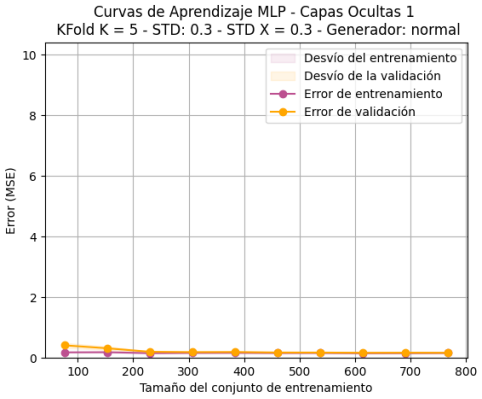


(a) Distribucion normal

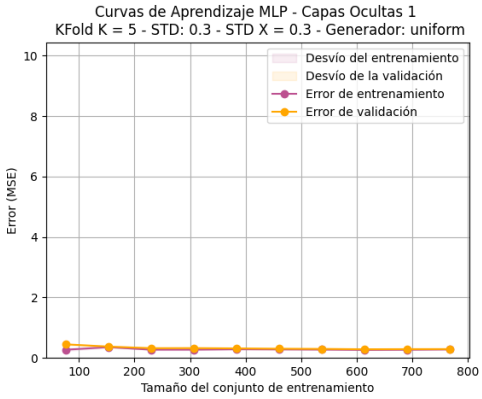


(b) Distribucion uniforme

Figura C.29: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

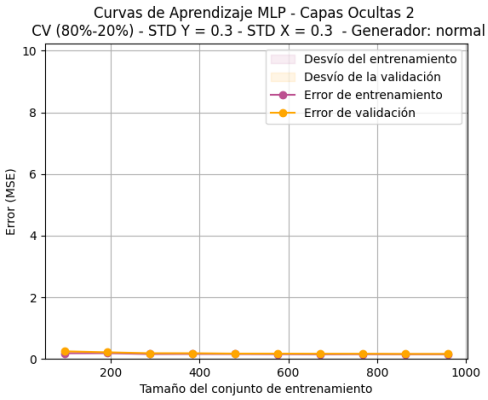


(a) Distribucion normal

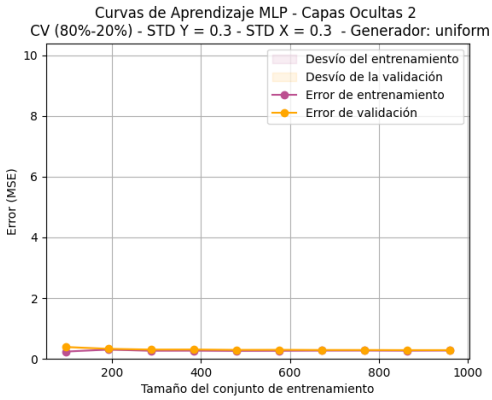


(b) Distribucion uniforme

Figura C.30: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta.

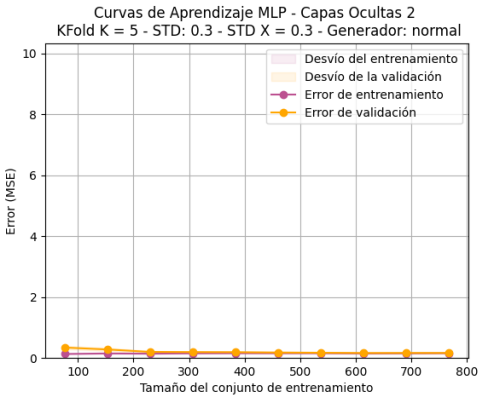


(a) Distribucion normal

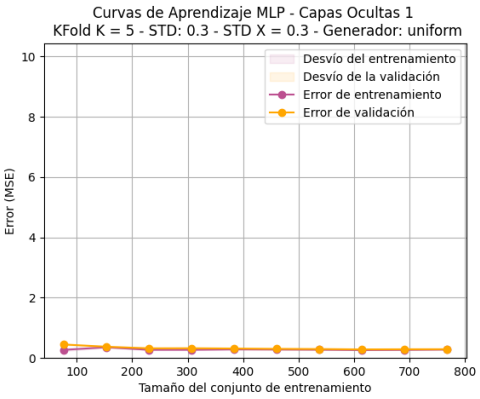


(b) Distribucion uniforme

Figura C.31: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.

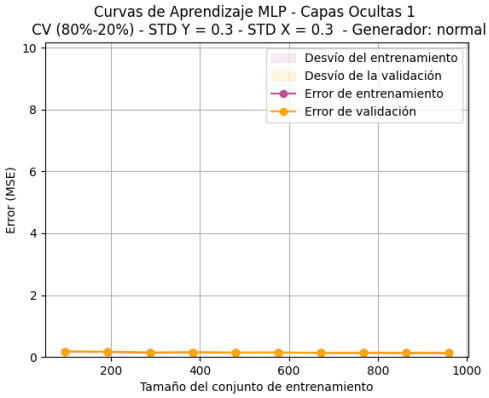


(a) Distribucion normal

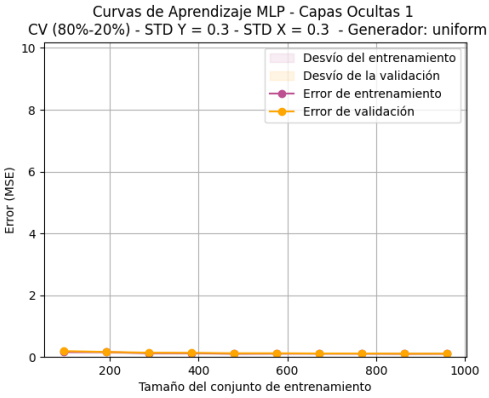


(b) Distribucion uniforme

Figura C.32: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F2, en este caso se trata de un modelo con una capa oculta.

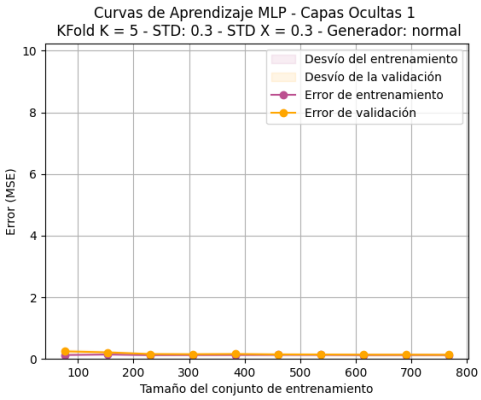


(a) Distribucion normal

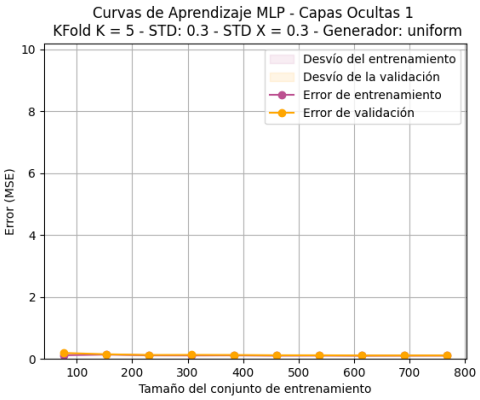


(b) Distribucion uniforme

Figura C.33: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta. Modelo poco profundo.

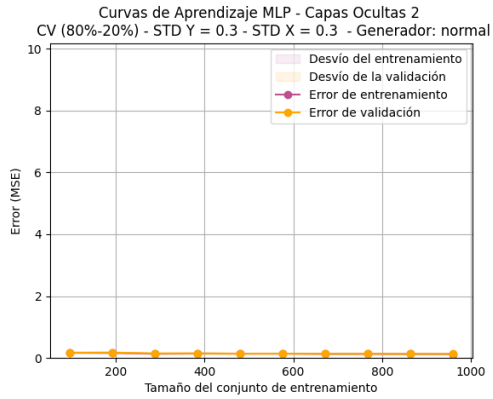


(a) Distribucion normal

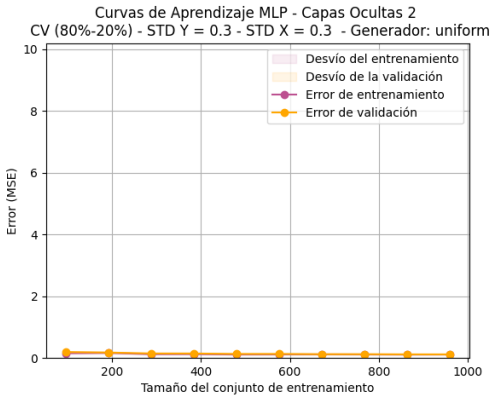


(b) Distribucion uniforme

Figura C.34: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta.

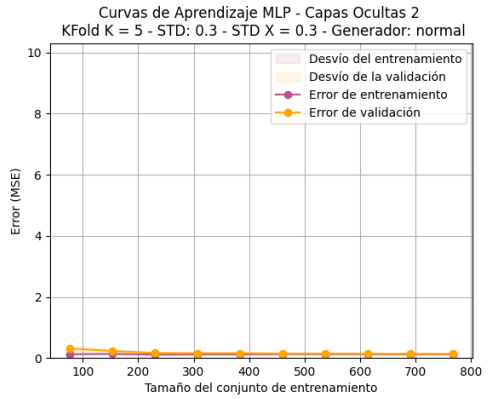


(a) Distribucion normal

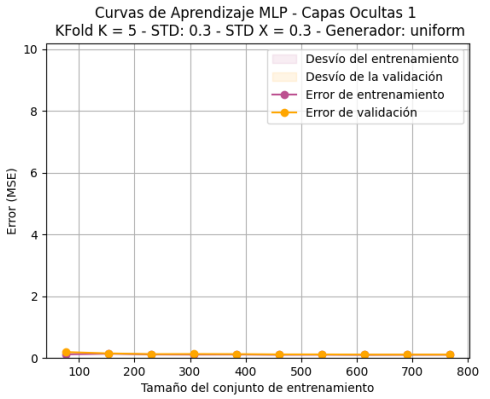


(b) Distribucion uniforme

Figura C.35: CV - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con dos capas oculta. Modelo profundo.



(a) Distribucion normal



(b) Distribucion uniforme

Figura C.36: KF - Comparación del desempeño de una MLP con distintas distribuciones de los datos en el PGD F3, en este caso se trata de un modelo con una capa oculta.

```
import sys

sys.path.insert(0, '../librerias')

import warnings

warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
import math
```

```
15
16 from sklearn.model_selection import learning_curve, train_test_split
17
18 import GPD as gpd
19
20 # Semilla para reproducibilidad
21 seed = 8834678
22 n_features = 5
23 dist_generators = ['normal', 'uniform']  # Valores posibles para dist_generator
24
25
26 def agregar_ruido_escalado_a_X(X, std=0.1):
27     min_val = np.min(X, axis=0)
28     max_val = np.max(X, axis=0)
29     range_val = max_val - min_val
30
31     ruido = std * range_val * np.random.normal(0, 1, X.shape)
32
33     X_noisy = X + ruido
34     return X_noisy
35
36
37 def plot_and_save_learning_curve(estimator, title, X, y, filename, generator_type,
       validation_method='kf', cv=5,
38                                  n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
39     # Siempre dividimos los datos
40     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
           random_state=42)
41
42     if validation_method == 'kf':
43         train_sizes, train_scores, test_scores = learning_curve(
44             estimator, X_train, y_train, cv=cv, n_jobs=n_jobs, train_sizes=
                   train_sizes,
45             scoring='neg_mean_squared_error',
46         )
47     else:  # 'cv'
48         train_sizes, train_scores, test_scores = learning_curve(
49             estimator, X, y, cv=None, n_jobs=n_jobs, train_sizes=train_sizes,
                   scoring='neg_mean_squared_error',
50         )
51
52
```

```
    train_scores_mean = -np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = -np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    #print(test_scores_mean[~np.isnan(test_scores_mean)])

    # Si algunos (pero no todos) son NaN o Inf
    if np.isnan(test_scores_mean).any() or np.isinf(test_scores_mean).any():
        print(title, ': no pudo ser generado')
        return None

    ylim = (0,max(test_scores_mean[~np.isnan(test_scores_mean)]) + 10)

    plt.figure()
    plt.title(f"{title} - Generador: {generator_type}")
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Error (MSE)")
    plt.ylim(ylim)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="#BC5090", label="Desvío del entrenamiento")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="#FFA600",
                          label="Desvío de la validación")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="#BC5090",
             label="Error de entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="#FFA600",
             label="Error de validación")

    plt.legend(loc="best")
    plt.savefig(filename)


# Lista con valores de desvío estándar
var = [0.0, 0.30, 0.50]
var_x = [0.0, 0.30, 0.50]


```

```
# Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
# en Kfold con k=5
for generator_type in dist_generators:

    # Cargar el conjunto de datos
    generator = gpd.make_friedman_1(generator_type)

    for std_value in var:  # Bucle para recorrer los valores de desvío estándar

        for std_value_x in var_x:
            # creación del conjunto de entrenamiento
            generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                )
            X, y = generator.get_data()

            # Corresponde agregar ruido en las X?
            if (std_value_x != 0.0):
                X = agregar_ruido_escalado_a_X(X, std=std_value_x)

            # Normalizar los datos para la red neuronal
            scaler = StandardScaler()
            X = scaler.fit_transform(X)

            # Definir diferentes configuraciones de capas ocultas
            hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]

            # Generar y guardar las curvas de aprendizaje para diferentes
                configuraciones de capas ocultas
            for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):
                mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
                                   max_iter=500,
                                   random_state=seed,
                                   solver='sgd',
                                   )

                title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                    hidden_layer_sizes)} \n KFold K = 5 - STD: {std_value} - STD X =
                     {std_value_x}"
                filename = f"LC_Capas/KF_learning_curve_hidden_layers_{len(
                    hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                    _std_value_x{std_value_x}.png"
```

```
129                 plot_and_save_learning_curve(mlp, title, X, y, filename,
                        generator_type, validation_method='kf', cv=5)
130
131 # Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
132 # en CV
133 for generator_type in dist_generators:
134
135     # Cargar el conjunto de datos
136     generator = gpd.make_friedman_1(generator_type)
137
138     for std_value in var:  # Bucle para recorrer los valores de desvío estándar de y
139
140         for std_value_x in var_x:  # Bucle para recorrer los valores de desvío está
                ndar de y
141             # creación del conjunto de entrenamiento
142             generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                    )
143             X, y = generator.get_data()
144
145             # Normalizar los datos para la red neuronal
146             scaler = StandardScaler()
147             X = scaler.fit_transform(X)
148
149             # Corresponde agregar ruido en las X?
150             if (std_value_x != 0.0):
151                 X = agregar_ruido_escalado_a_X(X, std=std_value_x)
152
153             # Definir diferentes configuraciones de capas ocultas
154             hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]
155
156             # Generar y guardar las curvas de aprendizaje para diferentes
                    configuraciones de capas ocultas
157             for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):
158
159                 mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
160                                    max_iter=500,
161                                    random_state=seed,
162                                    solver='sgd',
163                                    )
164
165                 title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                        hidden_layer_sizes)} \n CV (80%-20%) - STD Y = {std_value} - STD
```

```
                    X = {std_value_x} "
166                 filename = f"LC_Capas/CV_learning_curve_hidden_layers_{len(
                        hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                        _std_value_x_{std_value_x}.png"
167                 plot_and_save_learning_curve(mlp, title, X, y, filename,
                        generator_type, validation_method='cv')
168 # Mostrar las curvas de aprendizaje
169 # plt.show()
```

Friedman 2

```
1  import sys
2
3  sys.path.insert(0, '../librerias')
4
5  import warnings
6
7  warnings.filterwarnings("ignore")
8
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from sklearn.model_selection import learning_curve
12 from sklearn.neural_network import MLPRegressor
13 from sklearn.preprocessing import StandardScaler
14 import math
15
16 from sklearn.model_selection import learning_curve, train_test_split
17
18 import GPD as gpd
19
20 # Semilla para reproducibilidad
21 seed = 8834678
22 n_features = 5
23 dist_generators = ['normal', 'uniform']  # Valores posibles para dist_generator
24
25
26 def agregar_ruido_escalado_a_X(X, std=0.1):
27     min_val = np.min(X, axis=0)
28     max_val = np.max(X, axis=0)
29     range_val = max_val - min_val
30
31     ruido = std * range_val * np.random.normal(0, 1, X.shape)
```

```
32
33      X_noisy = X + ruido
34      return X_noisy
35
36
37  def plot_and_save_learning_curve(estimator, title, X, y, filename, generator_type,
        validation_method='kf', cv=5,
38                                   n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
39      # Siempre dividimos los datos
40      X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
            random_state=42)
41
42      if validation_method == 'kf':
43          train_sizes, train_scores, test_scores = learning_curve(
44              estimator, X_train, y_train, cv=cv, n_jobs=n_jobs, train_sizes=
                    train_sizes,
45              scoring='neg_mean_squared_error',
46          )
47      else:  # 'cv'
48          train_sizes, train_scores, test_scores = learning_curve(
49              estimator, X, y, cv=None, n_jobs=n_jobs, train_sizes=train_sizes,
                    scoring='neg_mean_squared_error',
50          )
51
52      train_scores_mean = -np.mean(train_scores, axis=1)
53      train_scores_std = np.std(train_scores, axis=1)
54      test_scores_mean = -np.mean(test_scores, axis=1)
55      test_scores_std = np.std(test_scores, axis=1)
56
57      # print(test_scores_mean[~np.isnan(test_scores_mean)])
58
59      # Si algunos (pero no todos) son NaN o Inf
60      if np.isnan(test_scores_mean).any() or np.isinf(test_scores_mean).any():
61          print(title, ': no pudo ser generado')
62          return None
63
64      ylim = (0, max(test_scores_mean[~np.isnan(test_scores_mean)]) + 10)
65
66      plt.figure()
67      plt.title(f"{title} - Generador: {generator_type}")
68      plt.xlabel("Tamaño del conjunto de entrenamiento")
69      plt.ylabel("Error (MSE)")
```

```
70      plt.ylim(ylim)
71
72      plt.grid()
73
74      plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
75                       train_scores_mean + train_scores_std, alpha=0.1,
76                       color="#BC5090", label="Desvío del entrenamiento")
77      plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
78                       test_scores_mean + test_scores_std, alpha=0.1, color="#FFA600",
                             label="Desvío de la validación")
79      plt.plot(train_sizes, train_scores_mean, 'o-', color="#BC5090",
80               label="Error de entrenamiento")
81      plt.plot(train_sizes, test_scores_mean, 'o-', color="#FFA600",
82               label="Error de validación")
83
84      plt.legend(loc="best")
85      plt.savefig(filename)
86
87
88  # Lista con valores de desvío estándar
89  var = [0.0, 0.30, 0.50]
90  var_x = [0.0, 0.30, 0.50]
91
92  # Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
93  # en Kfold con k=5
94  for generator_type in dist_generators:
95
96      # Cargar el conjunto de datos
97      generator = gpd.make_friedman_2(generator_type)
98
99      for std_value in var:  # Bucle para recorrer los valores de desvío estándar
100
101         for std_value_x in var_x:
102
103             # creación del conjunto de entrenamiento
104             generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                    )
105             X, y = generator.get_data()
106
107             # Corresponde agregar ruido en las X?
108             if (std_value_x != 0.0):
109                 X = agregar_ruido_escalado_a_X(X, std=std_value_x)
```

```
110
111             # Normalizar los datos para la red neuronal
112             scaler = StandardScaler()
113             X = scaler.fit_transform(X)
114
115             # El logaritmo natural, denotado como log, es la función inversa de la
116             # función exponencial, de manera que log(exp(x)) = x.
117             y = np.log(y)
118
119             # Definir diferentes configuraciones de capas ocultas
120             hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]
121
122
123             # Generar y guardar las curvas de aprendizaje para diferentes
                    configuraciones de capas ocultas
124             for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):
125
126                 mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
127                                    max_iter=500,
128                                    random_state=seed,
129                                    solver='sgd',
130                                    )
131
132                 title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                        hidden_layer_sizes)} \n KFold K = 5 - STD: {std_value} - STD X =
                         {std_value_x}"
133                 filename = f"LC_Capas/KF_learning_curve_hidden_layers_{len(
                        hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                        _std_value_x{std_value_x}.png"
134                 plot_and_save_learning_curve(mlp, title, X, y, filename,
                        generator_type, validation_method='kf', cv=5)
135
136 # Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
137 # en CV
138
139 for generator_type in dist_generators:
140
141     # Cargar el conjunto de datos
142     generator = gpd.make_friedman_2(generator_type)
143
144     for std_value in var:  # Bucle para recorrer los valores de desvío estándar
145
```

```
        for std_value_x in var_x:
            # creación del conjunto de entrenamiento
            generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                )
            X, y = generator.get_data()

            # Corresponde agregar ruido en las X?
            if (std_value_x != 0.0):
                X = agregar_ruido_escalado_a_X(X, std=std_value_x)

            # Normalizar los datos para la red neuronal
            scaler = StandardScaler()
            X = scaler.fit_transform(X)

            # El logaritmo natural, denotado como log, es la función inversa de la
            # función exponencial, de manera que log(exp(x)) = x.
            y = np.log(y)

            # Definir diferentes configuraciones de capas ocultas
            hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]

            # Generar y guardar las curvas de aprendizaje para diferentes
                configuraciones de capas ocultas
            for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):

                mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
                                   max_iter=500,
                                   random_state=seed,
                                   solver='sgd',
                                   )

                title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                    hidden_layer_sizes)} \n CV (80%-20%) - STD Y = {std_value} - STD
                     X = {std_value_x} "
                filename = f"LC_Capas/CV_learning_curve_hidden_layers_{len(
                    hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                    _std_value_x_{std_value_x}.png"
                plot_and_save_learning_curve(mlp, title, X, y, filename,
                    generator_type, validation_method='cv')

# Mostrar las curvas de aprendizaje
#plt.show()
```

Friedman 3

```
import sys

sys.path.insert(0, '../librerias')

import warnings

warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
import math

from sklearn.model_selection import learning_curve, train_test_split

import GPD as gpd

# Semilla para reproducibilidad
seed = 8834678
n_features = 5
dist_generators = ['normal', 'uniform']  # Valores posibles para dist_generator


def agregar_ruido_escalado_a_X(X, std=0.1):
    min_val = np.min(X, axis=0)
    max_val = np.max(X, axis=0)
    range_val = max_val - min_val
    ruido = std * range_val * np.random.normal(0, 1, X.shape)
    X_noisy = X + ruido
    return X_noisy


def plot_and_save_learning_curve(estimator, title, X, y, filename, generator_type,
    validation_method='kf', cv=5,
                                 n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
    # Siempre dividimos los datos
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
        random_state=42)

```

```
    if validation_method == 'kf':
        train_sizes, train_scores, test_scores = learning_curve(
            estimator, X_train, y_train, cv=cv, n_jobs=n_jobs, train_sizes=
                train_sizes,
            scoring='neg_mean_squared_error',
        )
    else:  # 'cv'
        train_sizes, train_scores, test_scores = learning_curve(
            estimator, X, y, cv=None, n_jobs=n_jobs, train_sizes=train_sizes,
                scoring='neg_mean_squared_error',
        )

    train_scores_mean = -np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = -np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # print(test_scores_mean[~np.isnan(test_scores_mean)])
    # Si algunos (pero no todos) son NaN o Inf
    if np.isnan(test_scores_mean).any() or np.isinf(test_scores_mean).any():
        print(title, ': no pudo ser generado')
        return None
    ylim = (0, max(test_scores_mean[~np.isnan(test_scores_mean)]) + 10)

    plt.figure()
    plt.title(f"{title} - Generador: {generator_type}")
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Error (MSE)")
    plt.ylim(ylim)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="#BC5090", label="Desvío del entrenamiento")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="#FFA600",
                          label="Desvío de la validación")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="#BC5090",
             label="Error de entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="#FFA600",
             label="Error de validación")
```

```
79
80      plt.legend(loc="best")
81      plt.savefig(filename)
82
83
84  # Lista con valores de desvío estándar
85  var = [0.0, 0.30, 0.50]
86  var_x = [0.0, 0.30, 0.50]
87
88  # Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
89  # en Kfold con k=5
90  for generator_type in dist_generators:
91
92      # Cargar el conjunto de datos
93      generator = gpd.make_friedman_3(generator_type)
94
95      for std_value in var:  # Bucle para recorrer los valores de desvío estándar
96
97          for std_value_x in var_x:
98
99              # creación del conjunto de entrenamiento
100             generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                    )
101             X, y = generator.get_data()
102
103             # Corresponde agregar ruido en las X?
104             if (std_value_x != 0.0):
105                 X = agregar_ruido_escalado_a_X(X, std=std_value_x)
106
107             # Normalizar los datos para la red neuronal
108             scaler = StandardScaler()
109             X = scaler.fit_transform(X)
110
111             # El logaritmo natural, denotado como log, es la función inversa de la
112             # función exponencial, de manera que log(exp(x)) = x.
113             # y = np.log(y)
114
115             # Definir diferentes configuraciones de capas ocultas
116             hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]
117
118
```

```
119             # Generar y guardar las curvas de aprendizaje para diferentes
                    configuraciones de capas ocultas
120             for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):
121                 mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
122                                    max_iter=500,
123                                    random_state=seed,
124                                    solver='sgd',
125                                    )
126
127
128                 title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                        hidden_layer_sizes)} \n KFold K = 5 - STD: {std_value} - STD X =
                         {std_value_x}"
129                 filename = f"LC_Capas/KF_learning_curve_hidden_layers_{len(
                        hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                        _std_value_x{std_value_x}.png"
130                 plot_and_save_learning_curve(mlp, title, X, y, filename,
                        generator_type, validation_method='kf', cv=5)
131
132 # Generar y guardar las curvas de aprendizaje para diferentes generadores de datos
133 # en CV
134 for generator_type in dist_generators:
135
136     # Cargar el conjunto de datos
137     generator = gpd.make_friedman_3(generator_type)
138
139     for std_value in var:  # Bucle para recorrer los valores de desvío estándar
140
141         for std_value_x in var_x:
142
143             # creación del conjunto de entrenamiento
144             generator.create(std=std_value, size=1200, columns=n_features, seed=seed
                    )
145             X, y = generator.get_data()
146
147             # Corresponde agregar ruido en las X?
148             if (std_value_x != 0.0):
149                 X = agregar_ruido_escalado_a_X(X, std=std_value_x)
150
151             # Normalizar los datos para la red neuronal
152             scaler = StandardScaler()
153             X = scaler.fit_transform(X)
```

```
154
155             # Definir diferentes configuraciones de capas ocultas
156             hidden_layer_sizes_list = [(10,), (10, 10,), (10, 10, 10)]
157
158
159             # Generar y guardar las curvas de aprendizaje para diferentes
                    configuraciones de capas ocultas
160             for idx, hidden_layer_sizes in enumerate(hidden_layer_sizes_list):
161                 mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
162                                    max_iter=500,
163                                    random_state=seed,
164                                    solver='sgd',
165                                    )
166
167                 title = f"Curvas de Aprendizaje MLP - Capas Ocultas {len(
                        hidden_layer_sizes)} \n CV (80%-20%) - STD Y = {std_value} - STD
                         X = {std_value_x} "
168                 filename = f"LC_Capas/CV_learning_curve_hidden_layers_{len(
                        hidden_layer_sizes)}_generator_{generator_type}_std_{std_value}
                        _std_value_x_{std_value_x}.png"
169                 plot_and_save_learning_curve(mlp, title, X, y, filename,
                        generator_type, validation_method='cv')
170
171 # Mostrar las curvas de aprendizaje
172 #plt.show()
```

GPD

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Sun Mar 28 13:21:59 2021
5
6  @author: lucila
7  """
8
9  #from sklearn.datasets import make_friedman1, make_friedman2, make_friedman3
10 import numpy as np
11 class gpd:
12
13     def __init__(self, generator):
14         self.generator = generator
```

```
15          self.data = []
16
17      def create(self, std, size, seed):
18          pass
19
20      def get_X(self):
21          return self.X
22
23      def get_y(self):
24          return self.y
25
26      def get_data(self):
27          return self.X,self.y
28
29
30
31  class make_friedman_1(gpd):
32
33      # ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
            make_friedman1.html
34
35      def __init__(self, generator):
36          super().__init__(generator)  # Llama al constructor de ClaseBase
37          self.Xi_MEAN = 0.5
38          self.Xi_STD = 0.3330
39          self.Xi_MIN = 0
40          self.Xi_MAX = 1
41
42      def create(self, std, size, columns, seed):
43
44          self.columns = columns
45          self.seed= seed
46          self.std = std
47          self.size = size
48
49          np.random.seed(self.seed)
50
51          if(self.generator=='normal'):
52              X0 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
53              X1 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
54              X2 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
55              X3 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
```

```
            X4 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
        else:
            X0 = np.random.uniform(self.Xi_MIN, self.Xi_MAX,size)
            X1 = np.random.uniform(self.Xi_MIN, self.Xi_MAX,size)
            X2 = np.random.uniform(self.Xi_MIN, self.Xi_MAX,size)
            X3 = np.random.uniform(self.Xi_MIN, self.Xi_MAX,size)
            X4 = np.random.uniform(self.Xi_MIN, self.Xi_MAX,size)

        #aca tengo que acumular las Xi
        self.X = np.column_stack((X0, X1, X2, X3,X4))

        self.y = (10 * np.sin(np.pi * self.X[:, 0] * self.X[:, 1]) +
              20 * (self.X[:, 2] - 0.5) ** 2 +
              10 * self.X[:, 3] + 5 * self.X[:, 4])

        #aca genero el ruido acorde
        if(std != 0):
            #falta la generacion del ruido
            noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                1, size)
            self.y = self.y + noise


# !/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 28 13:21:59 2021

@author: lucila
"""

# from sklearn.datasets import make_friedman1, make_friedman2, make_friedman3
import numpy as np


class gpd:

    def __init__(self, generator):
        self.generator = generator
        self.data = []

    def create(self, std, size, seed):
```

```
97          pass
98
99      def get_X(self):
100         return self.X
101
102     def get_y(self):
103         return self.y
104
105     def get_data(self):
106         return self.X, self.y
107
108
109 class make_friedman_1(gpd):
110
111     # ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
            make_friedman1.html
112
113     def __init__(self, generator):
114         super().__init__(generator)  # Llama al constructor de ClaseBase
115         self.Xi_MEAN = 0.5
116         self.Xi_STD = 0.3330
117         self.Xi_MIN = 0
118         self.Xi_MAX = 1
119
120     def create(self, std, size, columns, seed):
121
122         self.columns = columns
123         self.seed = seed
124         self.std = std
125         self.size = size
126
127         np.random.seed(self.seed)
128
129         if (self.generator == 'normal'):
130             X0 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
131             X1 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
132             X2 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
133             X3 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
134             X4 = np.random.normal(self.Xi_MEAN, self.Xi_STD, size)
135         else:
136             X0 = np.random.uniform(self.Xi_MIN, self.Xi_MAX, size)
137             X1 = np.random.uniform(self.Xi_MIN, self.Xi_MAX, size)
```

```
138             X2 = np.random.uniform(self.Xi_MIN, self.Xi_MAX, size)
139             X3 = np.random.uniform(self.Xi_MIN, self.Xi_MAX, size)
140             X4 = np.random.uniform(self.Xi_MIN, self.Xi_MAX, size)
141
142         # aca tengo que acumular las Xi
143         self.X = np.column_stack((X0, X1, X2, X3, X4))
144
145         self.y = (10 * np.sin(np.pi * self.X[:, 0] * self.X[:, 1]) +
146                   20 * (self.X[:, 2] - 0.5) ** 2 +
147                   10 * self.X[:, 3] + 5 * self.X[:, 4])
148
149         # aca genero el ruido acorde
150         if (std != 0):
151             # falta la generacion del ruido
152             noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                        1, size)
153             self.y = self.y + noise
154
155
156 class make_friedman_2(gpd):
157
158     # ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
            make_friedman2.html
159
160     def __init__(self, generator):
161         # Llama al constructor de ClaseBase
162         super().__init__(generator)
163         self.Xi_MEAN = [50, 260 * np.pi, 0.5, 6]
164         self.Xi_STD = [50 / 3, (260 * np.pi) / 3, 0.5 / 3, 2]
165         self.Xi_MIN = [0, 40 * np.pi, 0, 1]
166         self.Xi_MAX = [100, 560 * np.pi, 1, 11]
167
168     def create(self, std, size, columns, seed):
169
170         self.columns = columns
171         self.seed = seed
172         self.std = std
173         self.size = size
174
175         np.random.seed(self.seed)
176
177         if (self.generator == 'normal'):
```

```
            X0 = np.random.normal(self.Xi_MEAN[0], self.Xi_STD[0], size)
            X1 = np.random.normal(self.Xi_MEAN[1], self.Xi_STD[1], size)
            X2 = np.random.normal(self.Xi_MEAN[2], self.Xi_STD[2], size)
            X3 = np.random.normal(self.Xi_MEAN[3], self.Xi_STD[3], size)
        else:
            X0 = np.random.uniform(self.Xi_MIN[0], self.Xi_MAX[0], size)
            X1 = np.random.uniform(self.Xi_MIN[1], self.Xi_MAX[1], size)
            X2 = np.random.uniform(self.Xi_MIN[2], self.Xi_MAX[2], size)
            X3 = np.random.uniform(self.Xi_MIN[3], self.Xi_MAX[3], size)

        # aca tengo que acumular las Xi
        self.X = np.column_stack((X0, X1, X2, X3))

        self.y = (self.X[:, 0] ** 2 + (self.X[:, 1] * self.X[:, 2] - 1 / (self.X[:,
            1] * self.X[:, 3])) ** 2) ** 0.5
        # aca genero el ruido acorde
        if (std != 0):
            # falta la generacion del ruido
            noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                1, size)
            self.y = self.y + noise


class make_friedman_3(gpd):

    # ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
        make_friedman3.html

    def __init__(self, generator):
        # Llama al constructor de ClaseBase
        super().__init__(generator)
        self.Xi_MEAN = [50, 260 * np.pi, 0.5, 6]
        self.Xi_STD = [50 / 3, (260 * np.pi) / 3, 0.5 / 3, 2]
        self.Xi_MIN = [0, 40 * np.pi, 0, 1]
        self.Xi_MAX = [100, 560 * np.pi, 1, 11]

    def create(self, std, size, columns, seed):

        self.columns = columns
        self.seed = seed
        self.std = std
        self.size = size
```

```
217
218         np.random.seed(self.seed)
219
220         if (self.generator == 'normal'):
221             X0 = np.random.normal(self.Xi_MEAN[0], self.Xi_STD[0], size)
222             X1 = np.random.normal(self.Xi_MEAN[1], self.Xi_STD[1], size)
223             X2 = np.random.normal(self.Xi_MEAN[2], self.Xi_STD[2], size)
224             X3 = np.random.normal(self.Xi_MEAN[3], self.Xi_STD[3], size)
225         else:
226             X0 = np.random.uniform(self.Xi_MIN[0], self.Xi_MAX[0], size)
227             X1 = np.random.uniform(self.Xi_MIN[1], self.Xi_MAX[1], size)
228             X2 = np.random.uniform(self.Xi_MIN[2], self.Xi_MAX[2], size)
229             X3 = np.random.uniform(self.Xi_MIN[3], self.Xi_MAX[3], size)
230
231         # aca tengo que acumular las Xi
232         self.X = np.column_stack((X0, X1, X2, X3))
233
234         self.y = np.arctan((self.X[:, 1] * self.X[:, 2] - 1 / (self.X[:, 1] * self.X
                [:, 3])) / self.X[:, 0])
235
236         # Se agrega el ruido acorde
237         if (std != 0):
238             # falta la generacion del ruido
239             noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                    1, size)
240             self.y = self.y + noise
241
242
243 class make_friedman_2(gpd):
244
245     #ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
            make_friedman2.html
246
247     def __init__(self, generator):
248         # Llama al constructor de ClaseBase
249         super().__init__(generator)
250         self.Xi_MEAN = [50, 260 * np.pi, 0.5, 6]
251         self.Xi_STD = [50/3, (260 * np.pi)/3, 0.5/3, 2]
252         self.Xi_MIN = [0, 40 * np.pi, 0, 1 ]
253         self.Xi_MAX = [100, 560 * np.pi, 1, 11]
254
255
```

```
256     def create(self, std, size, columns, seed):
257 
258         self.columns = columns
259         self.seed= seed
260         self.std = std
261         self.size = size
262 
263         np.random.seed(self.seed)
264 
265         if(self.generator=='normal'):
266             X0 = np.random.normal(self.Xi_MEAN[0], self.Xi_STD[0], size)
267             X1 = np.random.normal(self.Xi_MEAN[1], self.Xi_STD[1], size)
268             X2 = np.random.normal(self.Xi_MEAN[2], self.Xi_STD[2], size)
269             X3 = np.random.normal(self.Xi_MEAN[3], self.Xi_STD[3], size)
270         else:
271             X0 = np.random.uniform(self.Xi_MIN[0], self.Xi_MAX[0],size)
272             X1 = np.random.uniform(self.Xi_MIN[1], self.Xi_MAX[1],size)
273             X2 = np.random.uniform(self.Xi_MIN[2], self.Xi_MAX[2],size)
274             X3 = np.random.uniform(self.Xi_MIN[3], self.Xi_MAX[3],size)
275 
276 
277         #aca tengo que acumular las Xi
278         self.X = np.column_stack((X0, X1, X2, X3))
279 
280         self.y = (self.X[:, 0] ** 2 + (self.X[:, 1] * self.X[:, 2]  - 1 / (self.X[:,
                1] * self.X[:, 3])) ** 2) ** 0.5
281         #aca genero el ruido acorde
282         if(std != 0):
283             #falta la generacion del ruido
284             noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                    1,size)
285             self.y = self.y + noise
286 
287 
288 class make_friedman_3(gpd):
289 
290     #ref https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
            make_friedman3.html
291 
292     def __init__(self, generator):
293         # Llama al constructor de ClaseBase
294         super().__init__(generator)
```

```
295         self.Xi_MEAN = [50, 260 * np.pi, 0.5, 6]
296         self.Xi_STD = [50/3, (260 * np.pi)/3, 0.5/3, 2]
297         self.Xi_MIN = [0, 40 * np.pi, 0, 1 ]
298         self.Xi_MAX = [100, 560 * np.pi, 1, 11]
299 
300     def create(self, std, size, columns, seed):
301 
302         self.columns = columns
303         self.seed= seed
304         self.std = std
305         self.size = size
306 
307         np.random.seed(self.seed)
308 
309         if(self.generator=='normal'):
310             X0 = np.random.normal(self.Xi_MEAN[0], self.Xi_STD[0], size)
311             X1 = np.random.normal(self.Xi_MEAN[1], self.Xi_STD[1], size)
312             X2 = np.random.normal(self.Xi_MEAN[2], self.Xi_STD[2], size)
313             X3 = np.random.normal(self.Xi_MEAN[3], self.Xi_STD[3], size)
314         else:
315             X0 = np.random.uniform(self.Xi_MIN[0], self.Xi_MAX[0],size)
316             X1 = np.random.uniform(self.Xi_MIN[1], self.Xi_MAX[1],size)
317             X2 = np.random.uniform(self.Xi_MIN[2], self.Xi_MAX[2],size)
318             X3 = np.random.uniform(self.Xi_MIN[3], self.Xi_MAX[3],size)
319 
320 
321         #aca tengo que acumular las Xi
322         self.X = np.column_stack((X0, X1, X2, X3))
323 
324         self.y = np.arctan((self.X[:, 1] * self.X[:, 2] - 1 / (self.X[:, 1] * self.X
                [:, 3])) / self.X[:, 0])
325 
326         #Se agrega el ruido acorde
327         if(std != 0):
328             #falta la generacion del ruido
329             noise = abs(max(self.y) - min(self.y)) * self.std * np.random.uniform(0,
                    1,size)
330             self.y = self.y + noise
```

C.2. Matrices de varianzas-covarianzas

C.2.1. Eficiencia energética

El conjunto de datos disponible en la URL https://archive.ics.uci.edu/ml/datasets/energy+efficiency se llama .Energy efficiencyz se refiere a la predicción de la eficiencia energética de edificios. En la fig. C.37 se muestra la matriz de varianza-covarianza correspondiente al conjunto de datos.

El conjunto de datos consta de 768 observaciones de edificios simulados. Hay ocho variables independientes, todas continuas: área total de la superficie del techo, área total de la superficie de la pared, área total de la superficie del piso, orientación, área acristalada, distribución de la superficie acristalada, altura del techo y edad del edificio. Las dos variables dependientes son la calefacción y la carga de refrigeración, que representan la energía necesaria para calentar y enfriar el edificio, respectivamente.

C.2.2. Yacht

En la fig. C.38 se muestra la matriz de varianza-covarianza correspondiente al conjunto de datos disponible en la URL https://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics se llama "Yacht Hydrodynamicsz se refiere a la resistencia de un modelo de yate en función de diferentes características.

El conjunto de datos consta de 308 observaciones de yates experimentales. Cada observación tiene seis variables independientes y una variable dependiente, todas con valores numéricos. La variable dependiente es la resistencia en la dirección longitudinal del yate (Y1), expresada en unidades de Newtons.

No hay valores faltantes en este conjunto de datos. Los datos están disponibles en formato de archivo CSV, donde cada columna representa una variable y cada fila representa una observación.

C.2.3. Incendios forestales

El conjunto de datos disponible en la URL https://archive.ics.uci.edu/ml/datasets/forest+fires es un conjunto de datos que contiene información sobre incendios forestales en el Parque Nacional Montesinho en Portugal.

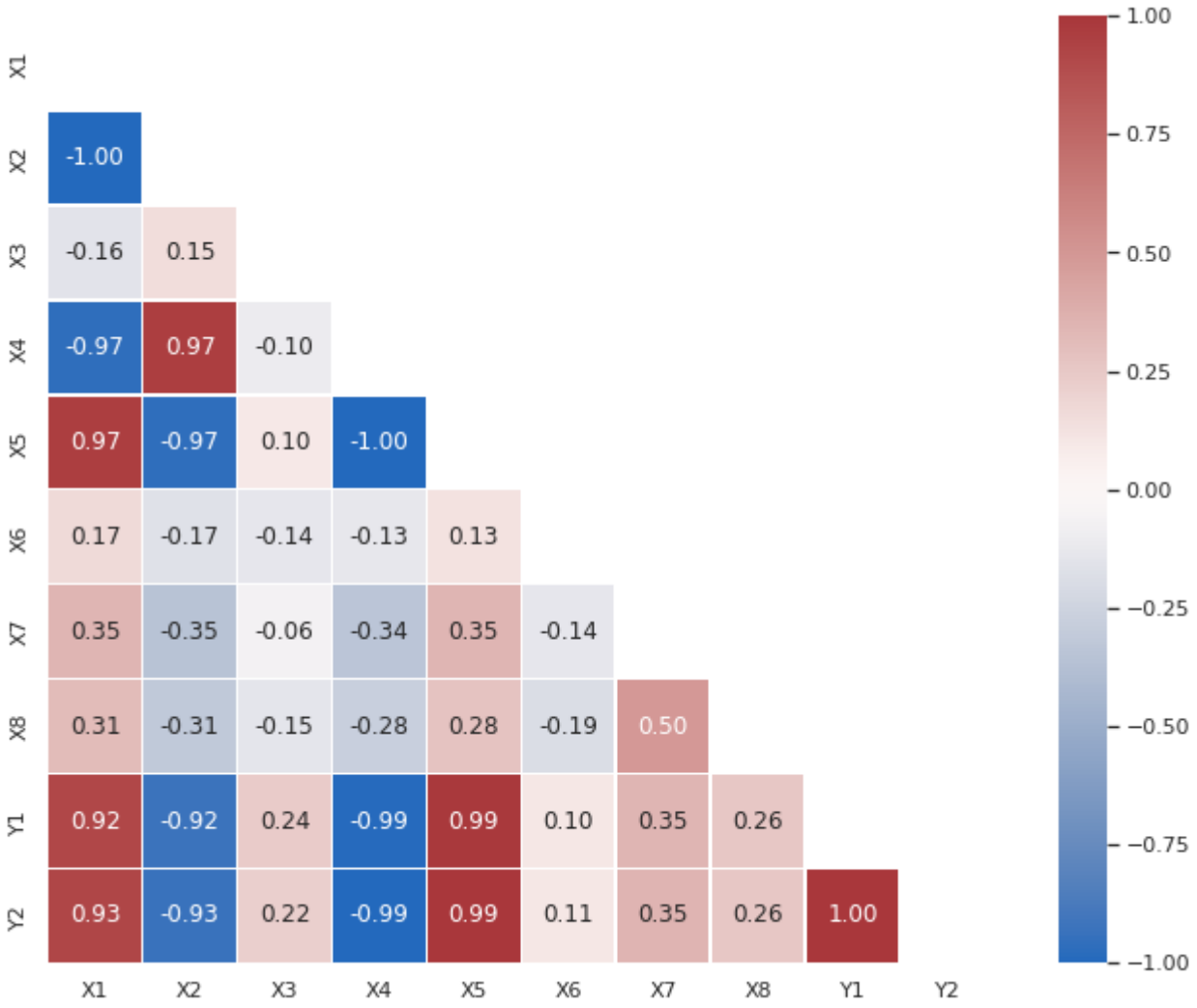


Figura C.37: Matriz de varianzas covarianzas del conjunto de datos ENB

El conjunto de datos contiene 517 registros y 13 atributos. Los atributos incluyen información sobre la ubicación geográfica del incendio, el mes y día del incendio, la temperatura, la humedad relativa, la velocidad del viento y otros factores meteorológicos que pueden afectar la propagación del fuego.

El atributo objetivo de este conjunto de datos es la variable "área", que indica el área total de bosque quemado en cada incendio. Los otros atributos se utilizan para predecir el valor de "área".

En la fig. C.39 se muestra la matriz de varianza-covarianza correspondiente a este conjunto de datos. Este conjunto de datos ha sido utilizado en la investigación para explorar cómo los factores meteorológicos y geográficos afectan la propagación de los incendios forestales y para desarrollar modelos predictivos que puedan ayudar a predecir el tamaño y la propagación de los

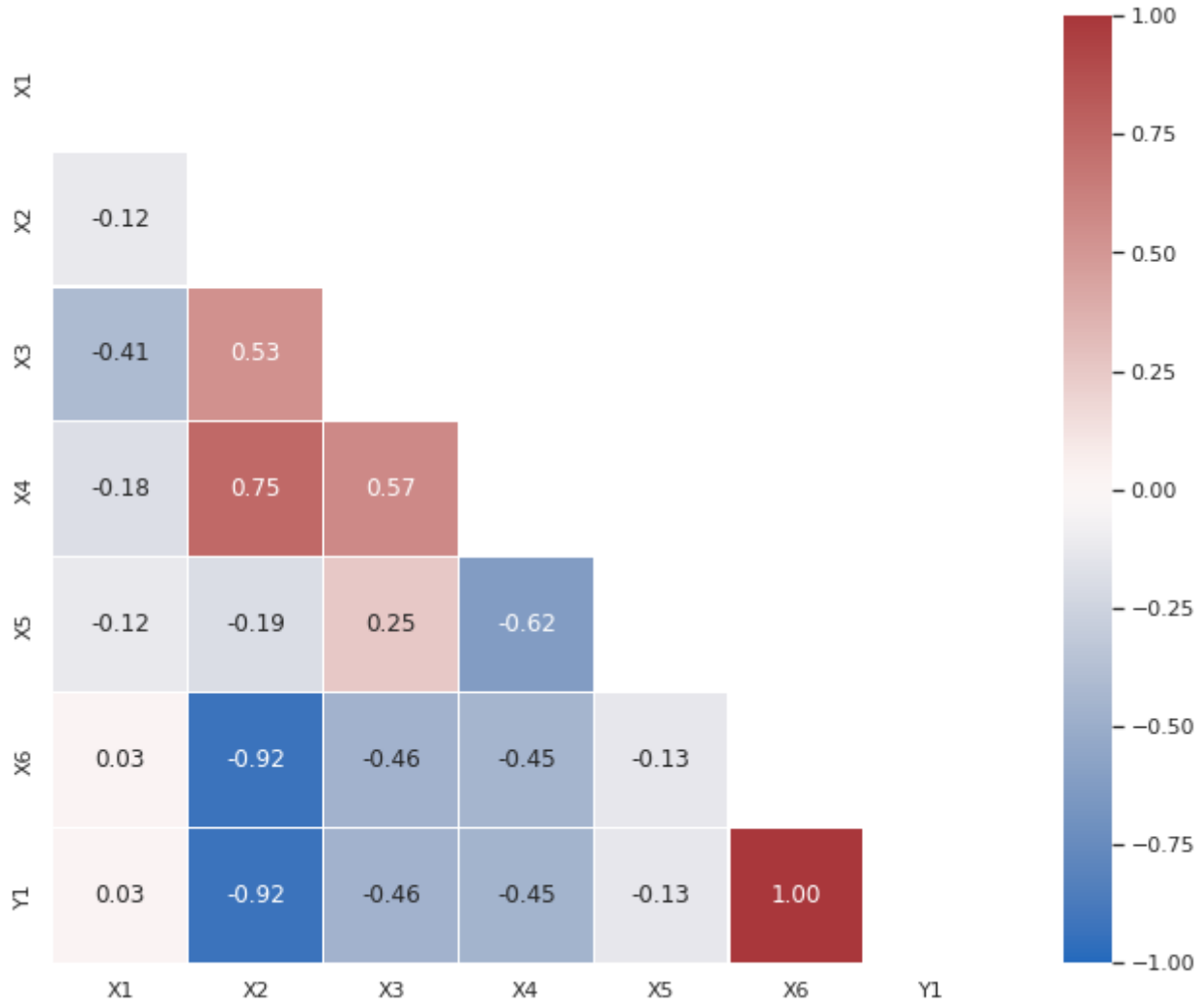


Figura C.38: Matriz de varianzas covarianzas del conjunto de datos Yacht

incendios forestales.

C.2.4. Concreto

El conjunto de datos disponible en la URL http://archive.ics.uci.edu/ml/datasets/concrete+slump+test se refiere a los resultados de pruebas de resistencia del concreto. El objetivo de estas pruebas es determinar la capacidad del concreto para mantener su forma y consistencia durante el proceso de vertido y fraguado.

El conjunto de datos contiene 103 registros y 9 atributos, incluyendo la cantidad de agua, el cemento y los agregados utilizados en la mezcla, así como información sobre la cantidad de superplastificante y otros aditivos que se agregaron. El atributo objetivo del conjunto de datos es

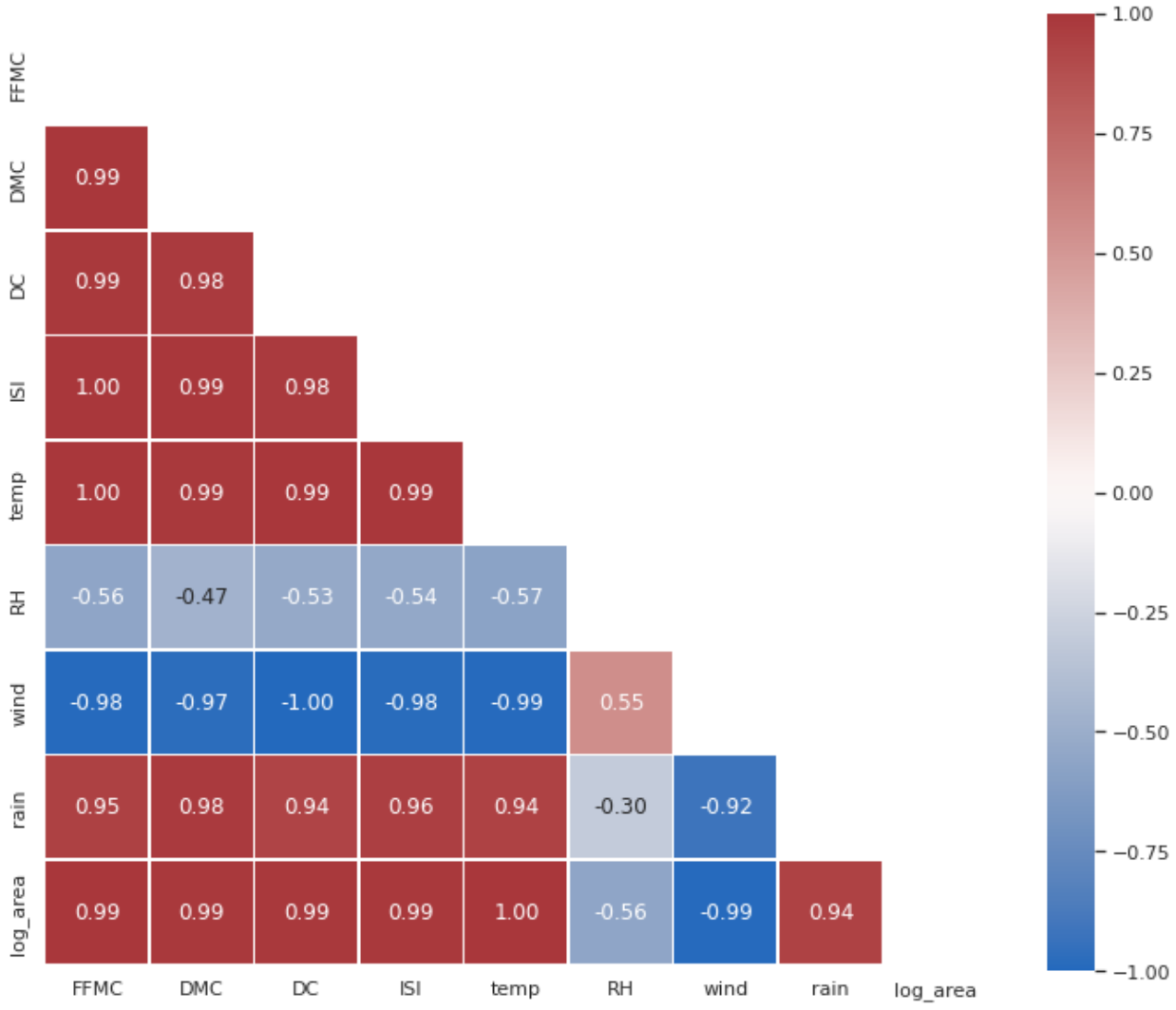


Figura C.39: Matriz de varianzas covarianzas del conjunto de datos Forest Fires

la "slump", que se refiere al asentamiento del concreto después de haber sido moldeado en un cono. Los otros atributos se utilizan para predecir el valor de slump.

Este conjunto de datos se ha utilizado en la investigación para explorar cómo los diferentes ingredientes y aditivos del concreto afectan su resistencia y capacidad de fraguado, lo que puede ayudar a los ingenieros y constructores a crear mezclas de concreto más efectivas. En la fig. C.40 se muestra la matriz de varianza-covarianza correspondiente a este conjunto de datos.

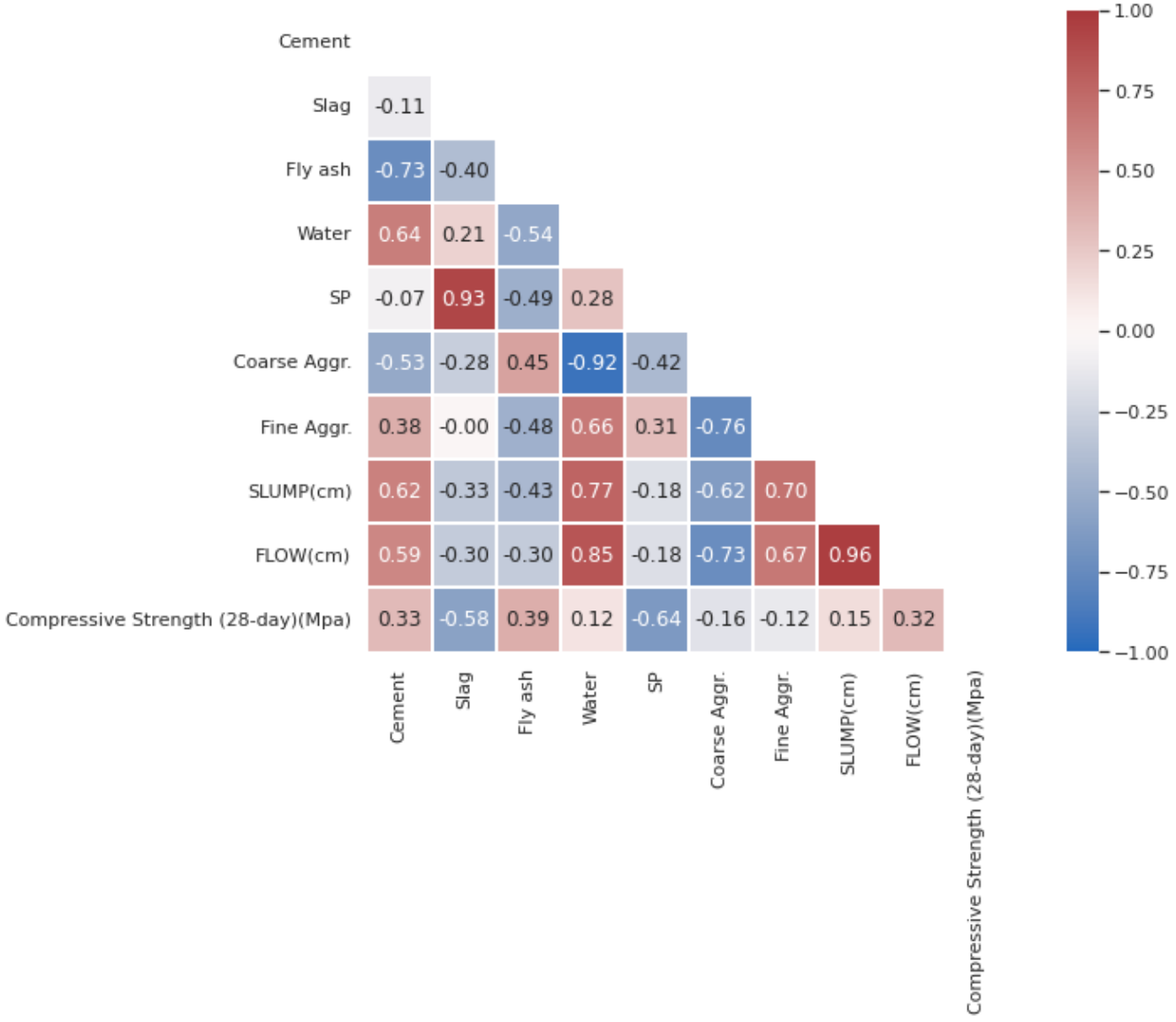


Figura C.40: Matriz de varianzas covarianzas del conjunto de datos Slump

D. Anexos del Capítulo 5

D.1. Esquema de algoritmo de optimización de hiperparametros basado en recocido simulado

Algoritmo de optimización de hiperparametros

Datos de entrada:

espacio_de_búsqueda

lista de parámetros

pesos_parametros

intentos

constructor_modelo

X_entrenamiento, X_testeo

y_entrenamiento, y_testeo

Datos de salida:

Comienzo

#inicialización de variables y estructura de datos

#auxiliares necesarios para el algoritmo

#Inicializa la temperatura inicial

temp_actual <- 1000

MM resume MejorModelo

MM_mae, MM_mape, MM_mse, MM_perdida <- +Infinito

MM_hp, MM <- Ninguno

```
#MA resume ModeloActual
MA_mae, MA_mape, MA_mse, MA_perdida <- +Infinito
MA_hp, MA <- Ninguno

#Otras inicializaciones
calendario_parametros <- generar_lista(lista de parámetros,
                                       pesos_parametros)

espacio_busqueda <- clonar(espacio_de_búsqueda_original espacio_busqueda)

lista_tabu_log_param <- crear_lista_vacia()

#Mientras haya presupuesto para continuar la busqueda
Mientras(intentos > 0):

    #Algunas inicializaciones

    #Se necesita comenzar o reiniciar la búsqueda ?
    Si(HP_candidato = Ninguno) o MA
        entonces
            HP_candidato,
            parametro <- nuevos_parametros(
                                espacio_de_búsqueda_original,
                                temp_actual,
                                EXPLORACION,
                                lista_de_parametros)
        sino
            #Selecciona el prox. parametro
            param_actual <- proximo(calendario_parametros)
```

```
                intentos_nuevo_param <- 3
                mientras(intentos_nuevo_param > 3):
                    HP_candidato,
                    parametro <- nuevos_parametros(
                                        espacio_de_búsqueda,
                                        temp_actual,
                                        EXPLORACION,
                                        lista_de_parametros)
                    intentos_nuevo_param <- intentos_nuevo_param -1

                    Si no pertenece(HP_candidato, lista_tabu_log_param)
                    entonces retornar mientras
                fin mientras

                espacio_busqueda <- HP_candidato

        fin si
fin mientras

rna_candidato <-constructor_modelo(HP_candidato)


MA_mae, MA_mape, MA_mse, MA_perdida <-
                entrenar(rna_candidato,
                            X_entrenamiento, X_testeo
                           y_entrenamiento, y_testeo)

Si(MA_perdida < MM_perdida)
    entonces #he encontrado un psuedo optimo
        MM_mae, MM_mape, MM_mse, MM_perdida <-
                MA_mae, MA_mape, MA_mse, MA_perdida
```

```
            MM_hp, MM <- MA_hp, MA
            cambio_a_EXPLOTACION <- Verdadero

    #Guardan datos
    Si(cambio_a_EXPLOTACION)
        entonces
            MAnt_mae, MAnt_mape, MAnt_mse, MAnt_perdida <-
                                MA_mae, MA_mape, MA_mse, MA_perdida

            MAnt, Mant_HP <- MA_hp, MA

            intentos_explotacion <- zona_explotacion

            mientras(intentos_explotacion >0):

                intentos_explotacion_nuevo_param <- 3
                #se genera una nueva combinación de hiperparametros
                #esta combinacion se genera dentro del entorno
                #reducido del pseudooptimo hallado en el paso anterior
                mientras(intentos_explotacion_nuevo_param > 3):
                    HP_candidato,
                    parametro <- nuevos_parametros(
                                        espacio_de_búsqueda,
                                        temp_actual,
                                        EXPLORACION,
                                        lista_de_parametros)
                    intentos_explotacion_nuevo_param <- intentos_explotacion_nuevo_param -1

                    Si no pertenece(HP_candidato, lista_tabu_log_param)
                    entonces retornar mientras
                fin mientras
```

```
                espacio_busqueda <- HP_candidato

                rna_explotacion <-constructor_modelo(HP_candidato)

                MExp_mae, MExp_mape, MExp_mse, MExp_perdida <-
                                        entrenar(rna_explotacion,
                                                X_entrenamiento, X_testeo
                                                y_entrenamiento, y_testeo)

                #pudimos mejorar la solución encontrada?
                Si MExp_perdida < MAnt_perdida
                    entonces
                        MM_mae, MM_mape, MM_mse, MM_perdida <-
                                                MExp_mae, MExp_mape, MExp_mse, MExp_perdida

                        #creo NM deberia ser MExp
                        MM_hp, MM <- HP_candidato, rna_explotacion
                fin si
                Si MExp_perdida < MAnt_perdida
                    entonces
                        MAnt_mae, MAnt_mape, MAnt_mse, MAnt_perdida <-
                            MExp_mae, MExp_mape, MExp_mse, MExp_perdida

                        MAnt, Mant_HP <- HP_candidato, rna_explotacion
                fin si

                intentos_explotacion <- intentos_explotacion - 1

        fin mientras
sino
```

```
        Si (MA_perdida < MAnt_perdida)
            entonces
                MAnt_mae, MAnt_mape, MAnt_mse, MAnt_perdida <-
                                    MA_mae, MA_mape, MA_mse, MA_perdida
                MAnt, Mant_HP <- MA_hp, MA
            sino
                si no es un no numero(MA_perdida)
                    entonces
                        #esta solución es peor que la anterior por lo que
                        #debe ser aceptada con un criterio probabilistico
                       delta_exp <-  math.exp((MA_perdida - MAnt_perdida) /
                                                temp_actual)

                       Si delta_exp > generacion_numero_alaeatorio_entre_0_y_1()
                            entonces
                                #acepto la solución aunque sea peor que la actual
                                    MAnt_mae, MAnt_mape, MAnt_mse, MAnt_perdida <-
                                            MA_mae, MA_mape, MA_mse, MA_perdida
                                        MAnt, Mant_HP <- MA_hp, MA
                            sino
                                    #rechazo la solución peor
                                    No hacer nada

        #identación de esta linea
        temperatura <- actualizar_temperatura()

    retornar MM_mae, MM_mape, MM_mse, MM_perdida, MM_hp, MM
```

Fin del algoritmo

Bibliografía

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. URL: https://books.google.com.ar/books?id=8Wjqvsoo48MC.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. URL: https://proceedings.mlr.press/v70/arjovsky17a.html.

[AJO+18] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. URL: https://www.sciencedirect.com/science/article/pii/S2405844018332067, doi:https://doi.org/10.1016/j.heliyon.2018.e00938.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24:2546–2554, 2011.

[BD20] B. Barz and J. Denzler. Deep learning on small datasets without pre-training using cosine loss. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1360–1369, Los Alamitos, CA, USA, mar 2020. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/WACV45572.2020.9093286, doi:10.1109/WACV45572.2020.9093286.

[Bel61] Richard Ernest Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, Princeton, NJ, 1961.

[BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BK88] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988. doi:10.1007/BF00332918.

[BK19] Steven L. Brunton and J. Nathan Kutz. *Data-driven science and engineering. Machine learning, Dynamical systems and control*. Cambridge University Press, New York, 2019.

[BL07] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. *Large-scale Kernel Machines*, 34(5):1–41, 2007.

[BLPL06] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL: https://proceedings.neurips.cc/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf.

[BLS+22] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022. doi:10.1109/TNNLS.2022.3229161.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda

Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[CA21] Clément Chadebec and Stéphanie Allassonnière. Data augmentation with variational autoencoders and manifold sampling. In Sandy Engelhardt, Ilkay Oksuz, Dajiang Zhu, Yixuan Yuan, Anirban Mukhopadhyay, Nicholas Heller, Sharon Xiaolei Huang, Hien Nguyen, Raphael Sznitman, and Yuan Xue, editors, *Deep Generative Models, and Data Augmentation, Labelling, and Imperfections*, pages 184–192, Cham, 2021. Springer International Publishing.

[Cao17] Longbing Cao. Data science: A comprehensive overview. *ACM Comput. Surv.*, 50(3), jun 2017. doi:10.1145/3076253.

[Cau47] Augustin Cauchy. Méthode générale pour la résolution de systèmes d'Équations simultanées. *Comptes Rendus des Séances de l'Académie des Sciences*, pages 536–538, 1847.

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Deep Learning and Representation Learning*

Workshop, 2014. https://arxiv.org/abs/1412.3555. URL: https://nips.cc//Conferences//2014//Schedule/?showEvent/=4294, doi:10.48550/ARXIV.1412.3555.

[CM07] Paulo Cortez and Aníbal de Jesus Raimundo Morais. A data mining approach to predict forest fires using meteorological data. *Associação Portuguesa para a Inteligência Artificial (APPIA)*, December 2007. URL: https://core.ac.uk/works/9801673.

[Col85] Jack Colin. Torrent of print strains the fabric of libraries. *The New York Times*, 1985. URL: https://www.nytimes.com/1985/02/25/us/torrent-of-print-strains-the-fabric-of-libraries.html.

[Cua19] Carlos María Cuadras. *Nuevos métodos de análisis multivariante*. CMC Edicions Barcelona, Spain, 2019.

[DB11] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems 24*, pages 666–674. Curran Associates, Inc., 2011.

[Dev07] J. Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2007. URL: https://books.google.com.ar/books?id=hDjQwAEACAAJ.

[DKLM05] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how*, volume 488. Springer, 2005.

[DKNU+20] Will Dabney, Zeb Kurth-Nelson, Naoshige Uchida, Clara Kwon Starkweather, Demis Hassabis, Rémi Munos, and Matthew Botvinick. A distributional code for value in dopamine-based reinforcement learning. *Nature*, 577(7792):671–675, 2020. doi:10.1038/s41586-019-1924-6.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[Doz16] Timothi Dozat. Incorporating nesterov momentum into adam. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, pages 1480–1489, 2016.

[EFH+13] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H. Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization*, 2013.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[ES15] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*, volume 53 of *Natural Computing Series*. Springer Publishing Company, Incorporated, Berlin, 2nd edition, 2015.

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL: `http://arxiv.org/abs/1803.03635`, `arXiv:1803.03635`.

[Fey81] Richard P. Feynman. *The Pleasure of Finding Things Out*. Basic Books, 1981.

[FH19] Matthias Feurer and Frank Hutter. *Hyperparameter optimization*, pages 3–39. Springer, 2019.

[Fri91] Jerome Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991.

[FSM20] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training. In *International Conference on Learning Representations*, 2020. URL: `https://openreview.net/forum?id=Hkl1iRNFwS`.

[FTD+16] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2016. doi:10.1109/ICRA.2016.7487173.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. http://www.deeplearningbook.org. URL: https://books.google.com.ar/books?id=Np9SDQAAQBAJ.

[Gol91] Arthur S. Goldberger. *A Course in Econometrics*. Harvard University Press, 1991.

[GOV22] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022. URL: https://arxiv.org/abs/2207.08815, doi:10.48550/ARXIV.2207.08815.

[Gow71] J.C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27:857–871, 1971. doi:10.2307/2528823.

[GP18] M. Gendreau and J.Y. Potvin. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer International Publishing, 2018. URL: https://books.google.com.ar/books?id=J3hvDwAAQBAJ.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors,

Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[GRB22] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24991–25004. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9e9f0ffc3d836836ca96cbf8fe14b105-Paper-Conference.pdf.

[GS11] Enzo Grossi and K Suzuki. Artificial neural networks and predictive medicine: a revolutionary paradigm shift. *Artificial Neural Networks-Methodological Advances and Biomedical Applications*, pages 139–150, 2011.

[GSM+17] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1487–1495, 2017.

[HAMS21] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[Han18] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/13f9896df61279c928f19721878fac41-Paper.pdf.

[HHLB11] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, volume 6683, 2011.

[Hin86] Geoffrey E Hinton. Learning representations by back-propagating errors. *Proceedings of the eighth annual conference of the cognitive science society*, pages 131–144, 1986.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. doi:10.1162/neco.2006.18.7.1527.

[HP96] Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, volume 18, pages 177–186, Seattle, 1996. Erlbaum, Mahwah, NJ.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL: http://arxiv.org/abs/1207.0580, arXiv:1207.0580.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL: https://www.sciencedirect.

com/science/article/pii/0893608089900208, doi:https://doi.org/10.1016/0893-6080(89)90020-8.

[HTFF17] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning*. Springer, 2da. edition, 2017.

[Ing11] Lester Ingber. Asa options. 2011. URL: https://www.ingber.com/asa11_options.pdf.

[JKRL09] Kevin Jarrett, Koray Kavukcuoglu, Marc Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL: http://arxiv.org/abs/1412.6980.

[KGJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LJ18] Lisha Li and Kevin Jamieson. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[LJD+16] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2016. URL: https://arxiv.org/abs/1603.06560, doi:10.48550/ARXIV.1603.06560.

[Mas66] Abraham H. Maslow. *The Psychology of Science a Reconnaissance*. Harper and Row, 1966.

[McC95] James L. McClelland. *Toward a Pragmatic Connectionism*, pages 131–144. Princeton University Press, 1995. URL: http://www.jstor.org/stable/j.ctt7ztg81.12.

[MDB18] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=ByJHuTgA-.

[MHS+13] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230, 2013.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[Moi02] Stephane Moins. *Implementation of a Simulated Annealing algorithm for Matlab*. dissertation, Institutionen för systemteknik, 2002. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-1344.

[MP43] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[MP69] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479:480, 1969.

[MPCB14] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates,

Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/109d2dd3608f669ca17920c511c2a41e-Paper.pdf.

[MS69] Minsky Marvin and A Papert Seymour. Perceptrons. *Cambridge, MA: MIT Press*, 6:318–362, 1969.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[MZZS18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[Nak19] Preetum Nakkiran. More data can hurt for linear regression: Sample-wise double descent, 2019. URL: https://arxiv.org/abs/1912.07242, doi:10.48550/ARXIV.1912.07242.

[Ng04] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1015330.1015435.

[NH09] Vinod Nair and Geoffrey E Hinton. 3d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems 22*, pages 1339–1347. Curran Associates, Inc, 2009.

[OLG07] Ivan Ortigosa, Raimundo Lopez, and Jose Garcia. A neural networks approach to residuary resistance of sailing yachts prediction. In *Proceedings of the International Conference on Marine Engineering MARINE 2007*, 2007.

[OW02] Jason Osborne and Elaine Waters. Four assumptions of multiple regression that researchers should always test. *Practical Assessment, Research & Evaluation*, 8(2), 2002. URL: http://PAREonline.net/getvn.asp?v=8&n=2.

[Pen20] Huimin Peng. A comprehensive overview and survey of recent advances in meta-learning. 2020. URL: https://arxiv.org/abs/2004.11149, doi:10.48550/ARXIV.2004.11149.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL: https://proceedings.mlr.press/v28/pascanu13.html.

[PTDU16] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, 2016.

[RDM22] Filip Radenovic, Abhimanyu Dubey, and Dhruv Mahajan. Neural basis models for interpretability. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 8414–8426. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/37da88965c016dca016514df0e420c72-Paper-Conference.pdf.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua

Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL: http://arxiv.org/abs/1511.06434.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2010.

[Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[RPCC06] Marc' aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Cun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL: https://proceedings.neurips.cc/paper/2006/file/87f4d79e36d68c3031ccf6c55e9bbd39-Paper.pdf.

[RSL+21] Hubert Ramsauer, Benjamin Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Marko Pavlovic, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *9th International Conference on Learning Representations (ICLR)*, 2021. URL: https://openreview.net/forum?id=tL89RnzIiCd.

[RTC17] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2), 2017. URL: https://www.mdpi.com/1099-4300/19/2/47, doi:10.3390/e19020047.

[Sch87] Jürgen Schmidhuber. Evolutionary principles in self-referential learning. In *Proceedings of the International Conference on Machine Learning*, pages 14–18. Morgan Kaufmann, 1987.

[SGBIGG18] Román Salmerón Gómez, Víctor Blanco Izquierdo, and Catalina García García. Micronumerosidad aproximada y regresión lineal múltiple. *Revista de Métodos Cuantitativos para la Economía y la Empresa*, 25:174–189, 2018.

[SGBNH22] Bernhard Schäfl, Lukas Gruber, Angela Bitto-Nemling, and Sepp Hochreiter. Hopular: Modern hopfield networks for tabular data, 2022. URL: https://arxiv.org/abs/2206.00664, doi:10.48550/ARXIV.2206.00664.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[SK17] Torgyn Shaikhina and Natasha Khovanova. Handling limited datasets with neural networks in medical applications: A small-data approach. *Artificial Intelligence in Medicine*, 75, 01 2017. doi:10.1016/j.artmed.2016.12.003.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25:2951–2959, 2012.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL: https://proceedings.mlr.press/v28/sutskever13.html.

[SSW+16] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. doi:10.1109/JPROC.2015.2494218.

[TA12] Bruce E. Bayers Teresa Audersik, Gerald Audersik. *Biologia, La vida en la tierra con fisiología*. Pearson Educación de México, 2012.

[TL19] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL: https://proceedings.mlr.press/v97/tan19a.html.

[TSK+18] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[TX12] Athanasios Tsanas and Anastasia Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[WGGK13] Matt N. Williams, Carlos Alberto Gómez Grajales, and Dason Kurkiewicz. Assumptions of multiple regression: Correcting two misconceptions. *Practical Assessment, Research & Evaluation*, 18(11):1–14, 2013.

[WH60] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[Whi89] Halbert White. Learning in artificial neural networks: A statistical perspective. *Neural computation*, 1(4):425–464, 1989. doi:10.1162/neco.1989.1.4.425.

[WM03] D.Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003. URL: https://www.sciencedirect.com/science/article/pii/S0893608003001382, doi:https://doi.org/10.1016/S0893-6080(03)00138-2.

[WM05] Cort J. Willmott and Kenji Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005. doi:10.3354/cr030079.

[WMMY17] Ronald E Walpole, Raymond H Myers, Sharon L Myers, and Keying Ye. *Probability and Statistics for Engineers and Scientists*. Pearson, 9th edition, 2017.

[WSP+20] Michael Widrich, Benjamin Schäfl, Marko Pavlovic, Hubert Ramsauer, Lukas Gruber, Markus Holzleitner, Johannes Brandstetter, Geir Kjetil Sandve, Victor Greiff, Sepp Hochreiter, and Günter Klambauer. Modern hopfield networks and attention for immune repertoire classification. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[Yeh07] I-Cheng Yeh. Modeling slump flow of concrete using second-order regressions and artificial neural networks. *Cement and Concrete Composites*, 29(6):474–480, 2007.

[YZ20] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *ArXiv e-prints*, 2020. URL: https://arxiv.org/abs/2003.05689, arXiv:2003.05689, doi:10.48550/arXiv.2003.05689.