



Esta obra está bajo una
Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

CURSO MODULAR BIG DATA Y DATA SCIENCE APLICADOS A LA ECONOMÍA Y A LA ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

MODULO 8. MINERÍA DE DATOS II

Mauricio Beltrán
Alfonso Carlos Carabantes
Pablo Sánchez Cabrera
Juan Antonio Vicente

Organizadores

Colaborador



aeca
Asociación Española de Contabilidad
y Administración de Empresas



2023

Índice

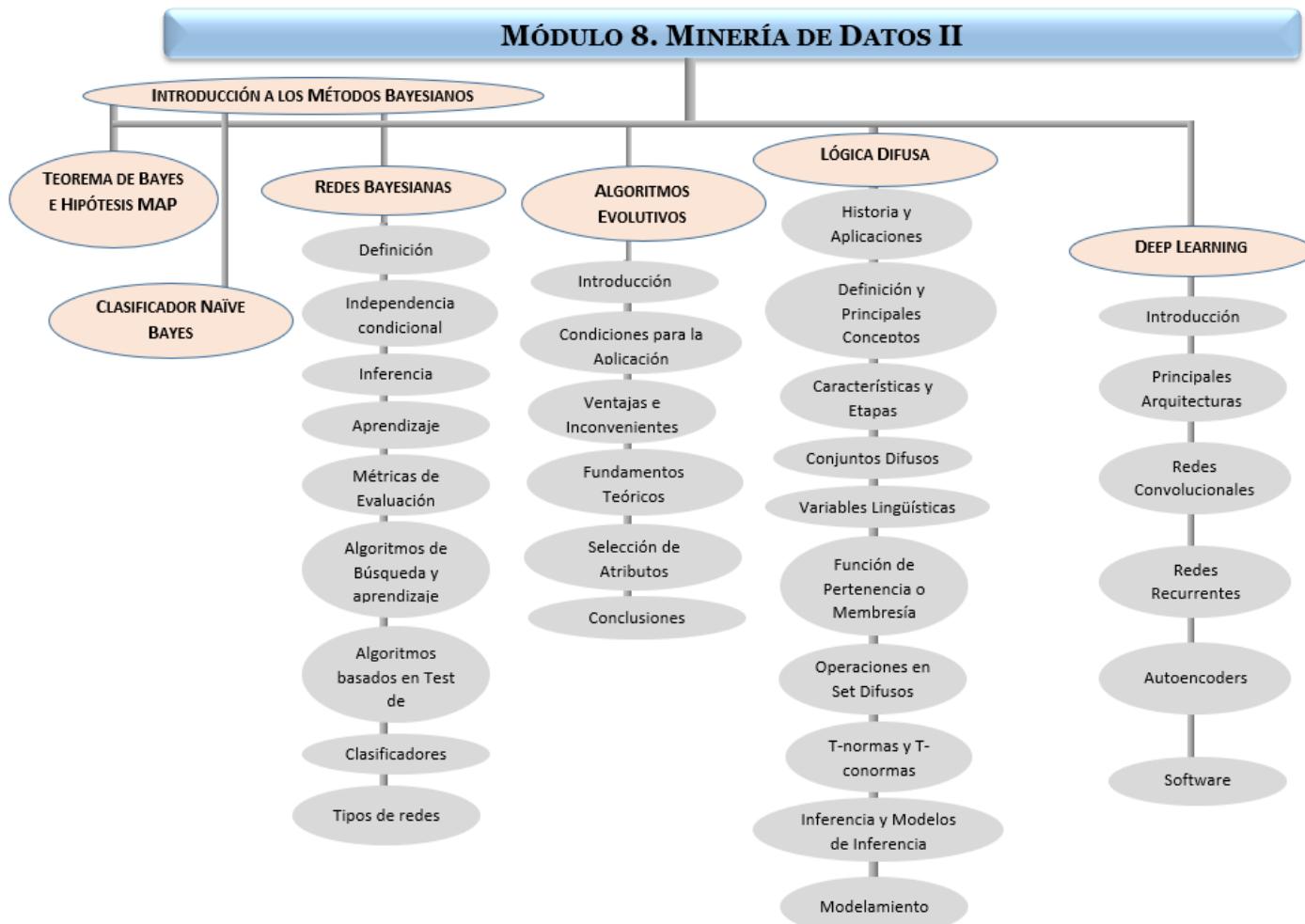
1. Introducción a los Métodos Bayesianos.....	6
2. Teorema de Bayes e hipótesis MAP	7
3. Clasificador Naïve Bayes.....	9
4. Redes bayesianas	12
4.1. Definición formal	12
4.2. Independencia condicional.....	14
4.3. Inferencia o propagación del conocimiento en la Red Bayesiana.....	16
4.4. Aprendizaje en las Redes Bayesianas	17
4.5. Métricas de evaluación.....	18
4.5.1. Métricas bayesianas	19
4.5.2. Métricas basadas en la teoría de la información.....	22
4.6. Algoritmos de búsqueda y aprendizaje	23
4.6.1. Algoritmo K2	24
4.6.2. Algoritmo B	25
4.6.3. Algoritmo Hill Climbing	25
4.6.4. Simulated Annealing	26
4.6.5. Tabu Search	27
4.7. Algoritmos basados en test de independencia	28
4.8. Clasificadores basados en Redes Bayesianas	30
4.8.1. Algoritmo TAN	30
4.8.2. Clasificadores bayesianos K-dependientes.....	32
4.8.3. Naïve Bayes aumentado (BAN).....	33
4.8.4. Average One-Dependence Estimators (AODE).....	34
4.8.5. Enfoques semi naïve Bayes y clasificadores extendidos	35
4.8.6. Multiredes bayesianas	39
4.8.7. Naive Bayes extendido a través de un árbol de clasificación (NBtree)	40
4.9. Tipos de Redes Bayesianas	41

4.9.1. Redes bayesianas Gausianas	41
4.9.2. Redes bayesianas Multinomiales.....	41
4.9.3. Redes bayesianas Mixtas	42
4.10. Ejemplo de Redes Bayesianas con WEKA.....	42
4.11. Redes bayesianas con R y Python.....	53
5. Algoritmos evolutivos.....	73
6. Lógica Difusa.....	100
6.1. Historia	100
6.2. Aplicaciones.....	101
6.3. Definición y principales conceptos	102
6.4. Características	103
6.5. Etapas	104
6.6. Conjuntos difusos	104
6.6.1. Definiciones sobre conjuntos difusos.....	106
6.7. Variables lingüísticas	107
6.8. Función de pertenencia o membresía.....	108
6.9. Operaciones en sets difusos	112
6.10. Complementos difusos, t-normas y t-conormas.....	114
6.10.1. Complementos difusos	114
6.10.2. T-normas.....	115
6.10.3. T-conormas	115
6.10.4. Principio de extensión	117
6.10.5. Operaciones de agregación	117
6.11. Inferencia usando lógica difusa	118
6.11.1. Relaciones difusas.....	118
6.11.2. Reglas IF-THEN difusas.....	119
6.11.3. Razonamiento difuso	120
6.11.4. Métodos de difusificación	123
6.12. Modelos (mecanismos) de inferencia difusa.....	123
6.12.1. El modelo Mamdani.....	124
6.12.2. El modelo Sugeno	126

6.12.3. Modelo Tsukamoto:.....	127
6.13. Modelamiento difuso	129
6.13.1. Grid partition	129
6.13.2. Tree partition	130
6.13.3. Scatter partition.....	130
6.14. Modelamiento difuso con R y Python	131
6.14.1. Sistema de inferencia difusa para controlar inundaciones	131
6.14.2. Algoritmo Fuzzy C-Means para agrupamiento	139
6.14.3 Lógica difusa en Python	140
7. Deep Learning.....	147
7.1. Introducción.....	147
7.2. Revisión de las Redes Neuronales	148
7.3. Principales arquitecturas de Deep Learning.....	162
7.4. Redes Neuronales Convolucionales	167
7.4.1. Introducción.....	167
7.4.2. Convolución	170
7.4.3. Pooling	173
7.4.4. Padding	175
7.4.5. Stride.....	177
7.4.6. Redes convolucionales con nombre propio	178
7.4.7. Ejemplos con Python y R.....	182
7.4.7.1. Código de Python	184
7.4.7.2. Código de R.....	217
7.4.8. Ejemplo de clasificación de textos con Conv1D en Python	222
7.5. Redes Neuronales Recurrentes	230
7.5.1. Introducción.....	230
7.5.2. Redes clásicas: Elman y Jordan	232
7.5.3. Red. LSTM (Long Short-Term Memory)	232
7.5.4. Red GRU	236

7.6. Autoencoders	246
7.7. Redes Generativas Adversarias	270
7.8. Actualidad y algunos conceptos relacionados con el Deep Learning.....	276
7.9. Software para aplicar Deep Learning.	280
7.9.1. Introducción.....	280
7.9.2. TensorFlow	281
7.9.3. Pytorch.....	283
7.9.4. Keras	284
7.9.5. MXNET	285
7.9.6. CNTK.....	286
7.9.7. DeepLearning4j.....	287
7.9.8. H2O	291
7.9.9. Google Colaboratoy para Python y R.....	292
7.9.10. Conclusiones	300
8. Bibliografía.....	301

Diagrama Conceptual



1. Introducción a los Métodos Bayesianos

Como seres humanos, nos enfrentamos muchas veces a la incertidumbre. Los métodos y técnicas bayesianas incorporan y cuantifican esta incertidumbre añadiendo la teoría de la probabilidad. De forma coloquial, podemos definir a estos procedimientos como una representación gráfica para manejar la incertidumbre en sistemas expertos.

Actualmente, se pueden considerar los métodos bayesianos como construcciones sencillas, con una semántica clara, y con un enfoque sólido y elegante. El problema que encuentran algunos autores es su elevado coste computacional.

Los modelos bayesianos sirven tanto para resolver problemas desde una perspectiva descriptiva como predictiva. Como método descriptivo se centra en descubrir las relaciones de dependencia/independencia. Desde esta óptica se puede afirmar que a veces complementan y/o incluso superan a las reglas de asociación. En cuanto a la función predictiva se circumscribe a las técnicas bayesianas como métodos de clasificación.

Michell (1997) nos sugiere dos razones por las que los métodos bayesianos son algunas de las técnicas que más se han utilizado en los problemas de inteligencia artificial, el aprendizaje automático y la minería de datos:

1. Constituyen un método muy válido y práctico para realizar inferencias con los datos que disponemos, lo que implica inducir modelos probabilísticos que, una vez calculados, se pueden utilizar con otras técnicas de minería de datos.
2. Son extremadamente útiles en la comprensión de otras técnicas de inteligencia artificial y minería de datos que no trabajan con las probabilidades de las que nos dotan las técnicas bayesianas. Esta combinación de métodos es muy provechosa para optimizar las soluciones de algunos problemas planteados en la minería de datos.

2. Teorema de Bayes e hipótesis MAP

Para comprender estas técnicas bayesianas vamos a empezar con el teorema de Bayes. Definamos las siguientes expresiones:

- $P(h)$ es la probabilidad a priori de que se cumpla la hipótesis h . Esta probabilidad contiene el conocimiento que tenemos de que la hipótesis h es correcta.
- $P(h/D)$ es la probabilidad a posteriori de que se cumpla la hipótesis h una vez conocidos los datos D . Esta expresión refleja la influencia que tienen los datos observados sobre la hipótesis h .
- $P(D/h)$ es la probabilidad de que los datos D sean observados en un escenario en el que la hipótesis h sea correcta.

Sabemos que:

$$P(h \cap D) = P(h) * P(D/h) \quad [1]$$

$$P(h \cap D) = P(D) * P(h/D) \quad [2]$$

Es decir:

$$P(h) * P(D/h) = P(D) * P(h/D) \quad [3]$$

Por lo tanto:

$$\frac{P(h/D)}{P(h)} = \frac{P(D/h)}{P(D)}$$

A posteriori A priori Factor de corrección

[4]

Observando la expresión del teorema de Bayes sabemos que $P(h/D)$ aumenta si se incrementa $P(h)$ y $P(D/h)$ o disminuye $P(D)$.

Como ya disponemos de la fórmula adecuada que nos da la probabilidad a posteriori, estamos interesados ahora en obtener aquella hipótesis más probable o hipótesis MAP (maximum a posteriori), observados los datos.

La expresión anterior la podemos escribir ahora como:

$$h_{MAP} = \arg \max_h P(h/D) = \arg \max_h [P(h) * P(D/h) / P(D)] \quad [5]$$

Y al ser $P(D)$ la misma en todas las hipótesis, la obtención del máximo se calcula prescindiendo de este término:

$$h_{MAP} = \arg \max_h P(h) * P(D/h) \quad [6]$$

h_{MAP} es la hipótesis más probable, dados los datos observados, $P(h/D)$.

En los problemas de clasificación disponemos de una variable clase (C) y un conjunto de variables predictoras o atributos que denominaremos A_1, A_2, \dots, A_n . Con estas especificaciones el teorema de Bayes tiene la siguiente expresión:

$$P(C/A_1, A_2, \dots, A_n) = \frac{P(C)P(A_1, A_2, \dots, A_n / C)}{P(A_1, A_2, \dots, A_n)} \quad [7]$$

En los procedimientos bayesianos la hipótesis más plausible es aquella que tiene la máxima probabilidad a posteriori dados los atributos (hipótesis MAP), cuya expresión es la siguiente:

$$\begin{aligned} c_{MAP} &= \arg \max_{C \in \Omega_C} P(c/A_1, A_2, \dots, A_n) = \arg \max_{C \in \Omega_C} \frac{P(c)P(A_1, A_2, \dots, A_n / c)}{P(A_1, A_2, \dots, A_n)} = \\ &= \arg \max_{C \in \Omega_C} P(c)P(A_1, A_2, \dots, A_n / c) \end{aligned} \quad [8]$$

Donde Ω_C representa el conjunto de valores que puede tomar la variable C .

En el último paso se ha eliminado el denominador debido a que sería el mismo para todas las categorías de la variable C .

Este método sencillo y claro posee un problema que es la complejidad computacional debido a que necesitamos trabajar con distribuciones de probabilidad que involucran muchas variables, lo que resulta, en la mayoría de los casos inmanejable.

3. Clasificador Naïve Bayes

El desarrollo de este famoso clasificador, incluido en la gran mayoría de paquetes informáticos, se encuentra en Duda y Hart (1973). Este método parte de la suposición de que todos los atributos son independientes conocido el valor de la variable clase:

$$I(X_i X_j | C), \forall i, j \quad [9]$$

La factorización de la función de probabilidad conjunta de este modelo es de la siguiente forma:

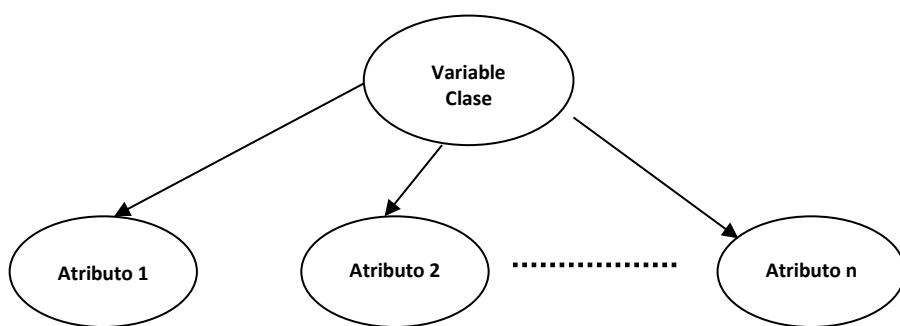
$$p(X_1, X_2, \dots, X_n, c) = P(C) \prod_{i=1}^n P(X_i | C) \quad [10]$$

Este supuesto es poco realista en la mayoría de los casos, pero, aun así, es uno de los más competitivos comparado con otras técnicas como las redes neuronales o los árboles de clasificación.

La estimación de los parámetros en este método, es decir, la clase o valor a devolver, será la resultante de aplicar la siguiente fórmula:

$$c_{MAP} = \arg \max_{C \in \Omega_C} P(c) P(A_1, A_2, \dots, A_n / c) = \arg \max_{C \in \Omega_C} P(c) \prod_{i=1}^n P(A_i / c) \quad [11]$$

FIGURA 1. ESQUEMA DE REPRESENTACIÓN DE NAÏVE-BAYES



Dados los datos de entrenamiento, se recorren todos esos datos y se computa la clasificación de cada uno de ellos, obteniendo $P(C_j)$ para cada clasificación posible.

Cuando los atributos son discretos, la estimación de la probabilidad condicional se extrae de la base de datos ya que son las frecuencias de aparición. Si $n(x_i, Pa(x_i))$ representa al número de registros de nuestra base de datos en el que la variable X_i toma el valor x_i y a los padres de X_i lo

denotamos por $Pa(x_i)$; entonces la fórmula de la probabilidad condicional viene determinada por el cociente entre el número de casos favorables y el de casos posibles:

$$P(x_i / Pa(x_i)) = \frac{n(x_i, Pa(x_i))}{n(Pa(x_i))} \quad [12]$$

Cuando las muestras son pequeñas o si se realizan muestreos en el que los cruces de dimensiones son frecuentes es muy probable que los resultados obtenidos sean muy dudosos. Para atenuar este problema existen procedimientos de estimadores basados en suavizados. Uno de los más conocidos es el estimador basado en la sucesión de Laplace, que viene definido por la siguiente fórmula:

$$P(x_i / Pa(x_i)) = \frac{n(x_i, Pa(x_i)) + 1}{n(Pa(x_i)) + |alt|} \quad [13]$$

Ahora la estimación de la probabilidad viene expresada por el número de casos favorables + 1 dividida por el de casos totales más el número de posibilidades o alternativas.

Esta estimación asume una distribución a priori uniforme y no puede ajustarse a nuestras necesidades si es que queremos suavizar más o menos la probabilidad. Existe otra forma de resolver el cálculo de la probabilidad que es a través del m-estimador, que no es más que una generalización de la corrección de Laplace. Su expresión matemática viene dada por:

$$P(x_i / Pa(x_i)) = \frac{n(x_i, Pa(x_i)) + mf_{Priori}(C)}{n(Pa(x_i)) + m} \quad [14]$$

Ahora el numerador son los casos favorables más una constante m multiplicada por la frecuencia de aparición a priori del evento y, el denominador es el número de casos totales más la constante m.

Cuando los datos son continuos el estimador Naïve-Bayes supone que la distribución de esta variable continua sigue una distribución normal. La media aritmética y la desviación típica que caracterizan a esta distribución gaussiana se estiman a través de los datos muestrales.

$$P(A_i / c) \propto N(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right] \quad [15]$$

Cuando las variables continuas no siguen una distribución de probabilidad normal las estimaciones a través de este método pueden ser muy deficientes, pero, en estos supuestos, se pueden aproximar a través de métodos kernel o, también, realizar la transformación de las variables cuantitativas en otras de intervalos con lo que se pueden obtener mejores resultados.

Existen diferentes contribuciones publicadas en la literatura para mejorar este método, las cuales pueden agruparse en dos tipos: clasificadores ingenuos extendidos y clasificadores ingenuos jerárquicos, publicados más recientemente.

Según diversos autores, Pearl (1988), Castillo et al. (1997), Jensen (2001) y Cowel (2001), este método de clasificación, que en muchas aplicaciones prácticas obtiene excelentes resultados, no alcanza a considerar de forma adecuada la semántica intrínseca de las redes bayesianas.

4. Redes bayesianas

Las redes bayesianas se conocen en la literatura existente con otros nombres como redes causales o redes causales probabilísticas, redes de creencia, sistemas probabilísticos, sistemas expertos bayesianos o, también, como diagramas de influencia. Las redes bayesianas son métodos estadísticos que representan la incertidumbre a través de las relaciones de independencia condicional que se establecen entre ellas (Edwards, 1998). Este tipo de redes codifica la incertidumbre asociada a cada variable por medio de probabilidades. Siguiendo a Kadie, Hovel y Horvitz (2001), una red bayesiana es un conjunto de variables, una estructura gráfica conectada a estas variables y un conjunto de distribuciones de probabilidad.

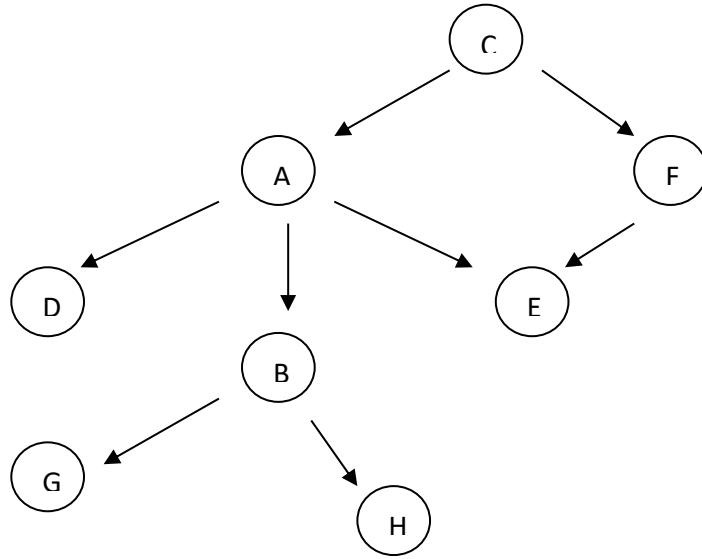
4.1. Definición formal

Las redes bayesianas probabilísticas automatizan el proceso de modelización utilizando toda la expresividad de los grafos para representar las dependencias e independencias, cuantificando esas relaciones a través de la teoría de la probabilidad. En esta unión se realiza de forma eficiente tanto el aprendizaje automático como la inferencia con los datos y la información disponible.

Una red bayesiana queda especificada formalmente por una dupla $B=(G,\Theta)$ donde G es un grafo dirigido acíclico (GDA) y Θ es el conjunto de distribuciones de probabilidad. Definimos un grafo como un par $G = (V, E)$, donde V es un conjunto finito de vértices, nodos o variables, y E es un subconjunto del producto cartesiano $V \times V$ de pares ordenados de nodos que llamamos enlaces o aristas.

El grafo es dirigido y acíclico. Dirigido porque los enlaces entre los vértices de la estructura están orientados, por ejemplo si $(A,B) \in E$ pero $(B,A) \notin E$ diremos que hay un enlace o un arco entre los nodos y lo representamos como $A \rightarrow B$. Cuando se dice que es acíclico es porque no pueden existir ciclos o bucles en el grafo, lo que significa que si empezamos a recorrer un camino desde un nodo no se puede regresar al punto de partida.

FIGURA 2. TOPOLOGÍA DE UNA RED BAYESIANA



Las conexiones del tipo $A \rightarrow B$ indican dependencia o relevancia directa entre las variables, en este caso se indica que B depende de A o que A es la causa de B y B el efecto de A. También se dice que A es el padre y B el hijo. La ausencia de arcos entre los nodos nos está aportando una valiosa información ya que en este caso el grafo nos informa de independencia condicional.

Las redes bayesianas tienen la habilidad de codificar la causalidad entre las variables por lo que han sido muy utilizadas en el modelado o en la búsqueda automática de estructuras causales (López, García y De la fuente 2006). La potencia de las redes bayesianas está en su capacidad de codificar las dependencias/independencias relevantes considerando no sólo las dependencias marginales sino también las dependencias condicionales entre conjuntos de variables

Los grafos definen un modelo probabilístico con las mismas dependencias utilizando una factorización mediante el producto de varias funciones de probabilidad condicionada:

$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | padres(x_i)) \quad [16]$$

$padres(x_i)$ son las variables predecesoras inmediatas de la variable x_i en la red, precisamente

$p(x_i | padres(x_i))$ son los valores que se almacenan en el nodo que precede a la variable x_i

A través de la factorización, las independencias del grafo son traducidas al modelo probabilístico de forma muy práctica.

Las redes bayesianas representan el conocimiento cualitativo del modelo mediante el grafo dirigido acíclico. Esta representación del conocimiento está articulada en la definición de las relaciones de dependencia/independencia. Al utilizar la representación gráfica a través del grafo hace que las redes bayesianas sean una herramienta muy poderosa y atractiva como representación del conocimiento.

4.2. Independencia condicional

La estructura o topología de la red bayesiana no sólo representa las dependencias entre las variables, sino que describe además las independencias condicionales existentes entre ellas.

Se dice que una variable X es condicionalmente independiente de otra variable Y dada una tercera Z, si el hecho de conocer Z hace que X e Y sean independientes. Es decir, que si conozco Z, Y no tiene influencia en X.

$$P(X|Y,Z)=P(X|Z) \quad [17]$$

Esta condición en una red bayesiana se traduce en que cada variable es independiente de todos aquellos nodos que no son sus descendientes.

Si enumeramos los nodos de la red bayesiana X_1, X_2, \dots, X_i de tal forma que cualquier nodo aparezca antes que cualquiera de sus descendientes, podemos afirmar que cada variable X_i es condicionalmente independiente de las variables del conjunto $\{X_1, X_2, \dots, X_i\}$ conocidos los valores de sus padres. Dicho de otro modo, conociendo los padres de una variable, ésta se vuelve independiente del resto de sus predecesores.

Pearl (1988) especifica que en una red bayesiana, la probabilidad conjunta de todas las variables, definida como:

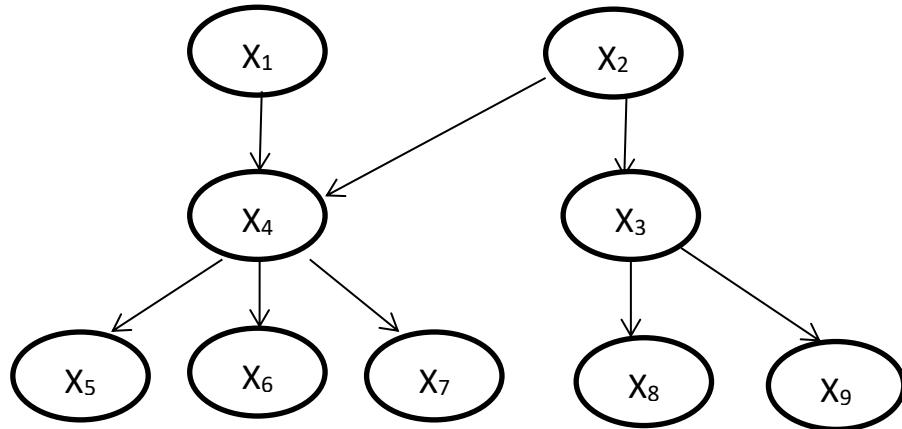
$$P(X_1, X_2, \dots, X_n) = \prod_1^n P(X_1 | X_2, \dots, X_{t-1}) \quad [18]$$

podemos calcularla a partir de las tablas de probabilidad condicional de cada variable en función de sus padres:

$$P(X_1, X_2, \dots, X_n) = \prod_1^n P(X_1 | X_2, \dots, X_{t-1}) = \prod_1^n P(X_1 | Pa(X_1)) \quad [19]$$

Para ilustrar este concepto tan importante, vamos a calcular la probabilidad conjunta de todos los nodos que componen la siguiente red bayesiana de nueve variables, definida la estructura por el siguiente diagrama:

FIGURA 3. TOPOLOGÍA DE UNA RED CON NUEVE PARÁMETROS



Aplicando la regla de la cadena obtenemos la siguiente expresión:

$$\begin{aligned}
 P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9) = & P(X_1) * P(X_2 | X_1) * P(X_3 | X_2, X_1) * P(X_4 | X_3, X_2, X_1) * P(X_5 | X_4, X_3, X_2, X_1) * \\
 & P(X_5 | X_4, X_3, X_2, X_1) * P(X_6 | X_5, X_4, X_3, X_2, X_1) * P(X_7 | X_6, X_5, X_4, X_3, X_2, X_1) * P(X_8 | X_7, X_6, X_5, X_4, X_3, X_2, X_1) * \\
 & P(X_9 | X_8, X_7, X_6, X_5, X_4, X_3, X_2, X_1)
 \end{aligned}$$

Debido a que las probabilidades condicionales sólo están influenciadas por sus padres, la expresión de la probabilidad conjunta de la red se reduce a la siguiente expresión:

$$\begin{aligned}
 P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9) = & P(X_1) * P(X_2) * P(X_3 | X_2) * P(X_4 | X_2, X_1) * P(X_5 | X_4) * P(X_6 | X_4) * \\
 & P(X_7 | X_4) * P(X_8 | X_3) * P(X_9 | X_3)
 \end{aligned}$$

Como se puede observar, el efecto de las probabilidades condicionales una vez dada la estructura de las relaciones entre las variables según la topología de la red se ha reducido considerablemente. La representación del conocimiento requiere ahora la estimación de muchos menos parámetros.

Un importante concepto para la construcción de una red bayesiana es el criterio *d-separación*, Jensen y Nielsen (2007).

Se dice que dos variables distintas *A* y *B* en una red causal están *d-separadas* (*d* para grafos dirigidos) si para todos los caminos entre *A* y *B*, hay una variable intermedia *V* (distinta de *A* y *B*) tal que se cumple una de las dos proposiciones siguientes:

- la conexión es serial o divergente y V está instanciada, o
- la conexión es convergente y ni V ni ninguno de sus descendientes ha recibido evidencia.

Si A y B no están d-separadas se llaman d-conectadas.

Cuando Z d-separa X e Y en G , se escribe $I(X, Y | Z)_G$ para indicar que la relación de independencia viene dada por el grafo G ; en caso contrario, se escribe $D(X, Y | Z)_G$ para indicar que X e Y son condicionalmente dependientes dado Z en el grafo.

4.3. Inferencia o propagación del conocimiento en la Red Bayesiana

Una vez obtenida la red bayesiana cuando se disponen de nuevos datos, nueva información o *evidencia*, necesitamos obtener nuevas conclusiones. A este proceso se le denomina inferencia probabilística o propagación del conocimiento a través de la red bayesiana.

La inferencia en redes bayesianas también conocida como actualización de creencias (belief updating) es el proceso de actualización de las probabilidades a posteriori en toda la estructura de la red, dado el conjunto de evidencias. Según definen Zhang y Poole (1996), en consonancia con diversos autorés más, se trata de un mecanismo para el cálculo de las distribuciones a posteriori de probabilidades para un conjunto de variables dado el conjunto de evidencias.

Si llamamos E a la evidencia presentada (lista de valores observados), y X es el conjunto de datos, el cálculo de la probabilidad de los datos X dada la evidencia E se realiza a través del teorema de Bayes:

$$P(X | E) = \frac{P(X)P(E | X)}{P(E)} \quad [20]$$

El cálculo de estas probabilidades es computacionalmente intratable para problemas con muchas variables debido al elevado número de combinaciones de valores que están involucrados. Sin embargo, existen métodos eficientes de propagación de la evidencia. Cooper (1990) demostró que la imputación de la probabilidad a posteriori es un problema NP-difícil incluso para una variable, lo que ha originado una búsqueda de soluciones que han ido encaminadas en dos direcciones: **métodos exactos** y **métodos aproximados**. Un buen resumen de métodos eficientes que utilizan la estructura de dependencias del grafo se puede consultar en Castillo et al. (1997).

- Los **métodos exactos** calculan las probabilidades a posteriori de forma exacta, pero al ser un problema de complejidad exponencial sólo se utiliza este procedimiento cuando el tiempo de proceso es asumible, o en algunos tipos especiales de redes bayesianas.

Algoritmos de propagación exacta han sido desarrollados mediante árboles de unión (expansión) en Pearl (1988), Castillo et al. (1997), Jensen (2001), Schachter et al. (1994), Baldi y Soren (2001), El-Hay (2001), Jensen y Nielsen (2007), Kjærulff y Madsen (2008) y a través de propagación perezosa (lazy propagation) en Shenoy (1992) y en Madsen y Jensen (1999).

- Los **métodos aproximados**, Salmerón (1998), se utilizan cuando la propagación exacta de probabilidades no puede llevarse a cabo en un tiempo razonable. Estos métodos pierden precisión en la solución que aportan, a cambio de obtener resultados en tiempos más cortos. Generalmente emplean distintas técnicas de simulación.

4.4. Aprendizaje en las Redes Bayesianas

Ya se ha señalado que para obtener una red bayesiana se ha de especificar una estructura gráfica y una función de probabilidad conjunta, que viene especificada por el producto de las probabilidades de cada nodo dados sus padres, lo que implica que en la mayoría de las ocasiones no se conocen ni la estructura ni las probabilidades. Esta es la razón por la que se han desarrollado diferentes métodos de aprendizaje para obtener la red bayesiana dados los datos.

Las tareas de aprendizaje a las que se enfrentan los diferentes métodos se pueden dividir en un aprendizaje estructural y un aprendizaje paramétrico. En este sentido la mayoría de los autores afirman que las redes bayesianas tienen dos dimensiones: una cuantitativa y otra cualitativa: Cowell, et al., (1999); Garbolino y Taroni, (2002); Nadkarni y Shenoy, (2001, 2004); Martínez y Rodríguez, (2003)

Dimensión cuantitativa. Aprendizaje paramétrico.

No sólo estas redes bayesianas modelan cualitativamente las relaciones, sino que también cuantifican y expresan de forma numérica la fuerza de esas relaciones entre las variables. Existen tres elementos que caracterizan la dimensión cuantitativa de la red bayesiana: el concepto de probabilidad, como medida del grado de creencia subjetiva relativa a un evento, un conjunto de funciones de probabilidad condicionada que definen a cada variable en el modelo y el teorema de Bayes que se utiliza para actualizar las probabilidades con base a la experiencia.

La fuerza de las relaciones entre las variables está especificada en las distribuciones de probabilidad como una medida de la creencia que tenemos sobre esas relaciones en el modelo.

El aprendizaje paramétrico consiste en hallar los parámetros asociados a la estructura de la red. Estos parámetros están constituidos por las probabilidades de los nodos raíz y las probabilidades condicionales de las demás variables dados sus padres.

Las probabilidades previas se corresponden con las marginales de los nodos raíz y las condicionales se obtienen de las distribuciones de cada nodo con sus padres.

Dimensión cualitativa. Aprendizaje estructural.

En el aprendizaje estructural es donde se establecen las relaciones de dependencia que existen entre las variables del conjunto de datos para obtener el mejor grafo que represente estas relaciones. Este problema, como ya se ha visto, es bastante complejo, dado que la búsqueda de la estructura que nos represente mejor a los datos es un problema NP-completo, lo que lo hace computacionalmente intratable cuando el número de variables es grande. Muchas veces se buscan algoritmos eficientes que, si bien no es lo óptimo, sí que se aproximan a la solución buscada con costes computacionales acotados, Neapolitan (2003).

Básicamente, se pueden englobar en dos tipos los métodos de aprendizaje de la estructura. Se encuentra por una parte aquellos métodos que utilizan métricas de complejidad-bondad de ajuste y los algoritmos de búsqueda.

4.5. Métricas de evaluación

La métrica define la calidad de la red bayesiana en función de los datos y el algoritmo de búsqueda tratará de encontrar la red que maximice esta métrica explorando todas las posibilidades. Téngase en cuenta que el número de posibles estructuras gráficas aumenta considerablemente con el número de variables, es un problema NP-duro. Por ejemplo, existen 12 grafos para tres variables y se eleva a 543 si las variables son cuatro.

Dependiendo de la métrica utilizada y la técnica de búsqueda, existen una amplia gama de procedimientos que pueden ir desde métodos voraces simples Cooper y Herskovitz, (1992) hasta métodos que utilizan algoritmos genéticos Larrañaga et al. (1996).

Otros métodos están basados en test estadísticos para detectar las posibles dependencias/independencias presentes en los datos, por lo que la red se ajustaría a estas

dependencias descubiertas. Estos métodos parecen más eficientes, pero pueden ser muy sensibles a los fallos en los test, especialmente cuando en el problema están involucradas muchas variables, Friedman et al. (1999).

También se pueden utilizar ambas estrategias para optimizar la búsqueda y construir el grafo, Campos (2006).

Dado un conjunto de datos $D = (X_1, X_2, \dots, X_n)$, se define una métrica de evaluación como el procedimiento de encontrar el GDA (G^*) tal que verifique la siguiente expresión:

$$\zeta^* = \arg \max_{\zeta \in \zeta^n} f(\zeta : D) \quad [21]$$

Donde

$$f(\zeta : D) = \sum_{i=1}^n f_D(X_i, Pa_\zeta(X_i))$$

$f(\zeta : D)$ representa a la función de evaluación que mide la calidad de un GDA candidato ζ en relación al conjunto de datos y ζ^n es el conjunto de todos los GDAs que pueden construirse con los n nodos de las variables.

Para poder utilizar estas fórmulas, es necesario que las métricas sean descomponibles ante datos completos. Se dice que una métrica es descomponible, Nielsen y Jensen (2009), si la evaluación de un GDA dado es igual a la suma de los valores obtenidos de cada uno de sus nodos y sus correspondientes familias (nodo y sus padres). Esta propiedad se puede expresar como:

$$f_D(X_{i,Pa_\zeta}(X_i)) = f_D(X_i | Pa_\zeta(X_i)) : N_{xi,pa_\zeta(X_i)} \quad [22]$$

$N_{xi,pa_\zeta(X_i)}$ son los estadísticos de las variables X_i y $Pa_\zeta(X_i)$ son calculadas sobre los datos D que se corresponden con cualquier configuración posible de $\{X_i\} \cup Pa_\zeta(X_i)$

4.5.1. Métricas bayesianas

Las métricas bayesianas buscan la estructura que maximiza la probabilidad de una red condicionada a la base de datos usando la fórmula de Bayes:

$$P(G | D) = \frac{P(D | G)P(G)}{P(D)} \quad [23]$$

$P(G)$ es la distribución a priori de cada estructura candidata y $P(D | G)$ se le conoce como evidencia ya que es la verosimilitud promedio que puede calcularse bajo ciertas suposiciones.

Se puede prescindir del denominador de la expresión de la fórmula de Bayes porque los datos son siempre los mismos para las distintas redes que podemos construir de un mismo problema. También se trabaja habitualmente con los logaritmos de las fórmulas, por ser más fácil a la hora de trabajar con las diferentes métricas. Por otra parte, el término $P(G)$ puede ignorarse si se utiliza una distribución uniforme.

En los párrafos siguientes se van a describir las fórmulas de los principales métodos, pero en primer lugar se concreta la notación común a todos ellas en la tabla 1.

En la métrica del procedimiento K2 de Cooper y Herskovitz, (1992) si se verifican un conjunto de condiciones tales como la independencia de los casos de la base de datos, que no existan casos perdidos o desconocidos y la uniformidad de las distribuciones de probabilidad de los parámetros de una red, podemos establecer cuál es la distribución de probabilidad conjunta de un GDA y una base de datos:

TABLA 1. NOTACIÓN DE LAS DIFERENTES MÉTRICAS DE REDES BAYESIANAS

Elemento	Descripción
N	Número de instancias en el conjunto de datos D
r_i	Número de estados de la variable aleatoria X_i
x_{ik}	k-ésimo valor de la variable X_i
$q_i = \prod_{x_j \in Pax_i} r_j$	Número de configuraciones posibles del conjunto de padres Pax_i de X_i
w_{ij}	j-ésima configuración de C siendo $1 \leq j \leq q_i$
N_{ijk}	Número de instancias del D en las que la variable X_i toma el k-ésimo valor x_{ik} y las variables en Pax_i toman su j-ésima configuración w_{ij}
$N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$	Número de instancias en D en las que las variables en Pax_i toman su j-ésima configuración w_{ij}
$N_{ik} = \sum_{j=1}^{l_i} N_{ijk}$	Número de instancias en D en las que la variable X_i toma el k-ésimo valor a x_{ik} .

$$f_{K2}(\zeta : D) = \log(p(\zeta)) + \sum_{i=1}^n \sum_{j=1}^{q_i} \left(\log \left(\frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \right) \right) + \sum_{k=1}^{r_i} \log(N_{ijk}!) \quad [24]$$

La métrica BD (Bayesian Dirichlet) es una generalización de la métrica que emplea el algoritmo de búsqueda K2 y fue propuesta por Heckerman et al (1994)

$$f_{BD}(\zeta : D) = \log(p(\zeta)) + \sum_{i=1}^n \sum_{j=1}^{q_i} \left(\log \left(\frac{\Gamma(\eta_{ij})}{\Gamma(N_{ij} + \eta_{ij})} \right) \right) + \sum_{k=1}^{r_i} \log \left(\frac{\Gamma(N_{ijk} + \eta_{ijk})}{\Gamma(\eta_{ijk})} \right) \quad [25]$$

Donde η_{ijk} representan los hiperparámetros de la distribución a priori de Dirichlet dada la estructura de la red y $\Gamma()$ es la distribución Gamma.

La especificación de los hiperparámetros η_{ijk} no es sencilla, pero si realizamos la asunción de equivalencia en verosimilitud, el procedimiento se simplifica y la métrica así calculada recibe el nombre de métrica BDe, Heckerman et al. (1994) y cuya expresión es igual a la anterior pero ahora los valores se calculan a través de la siguiente expresión:

$$\eta_{ijk} = \eta \times p(x_{ik} w_{ij} | \zeta_0) \quad [26]$$

Donde $p(x_{ik} w_{ij} | \zeta_0)$ es una distribución de probabilidad asociada a la red bayesiana a priori y η es un parámetro que representa el tamaño muestral equivalente.

Otra variación muy interesante la encontramos en Buntine (1991) donde ahora la asignación de la probabilidad a priori de cada variable y sus padres se realiza a través de una distribución uniforme,

lo que significa que $p(x_{ik} w_{ij} | \zeta_0) = \frac{1}{r_i q_i}$. A esta métrica se la conoce con el nombre de BDeu y

sólo depende del parámetro η . Su expresión matemática sustituyendo en la ecuación anterior es la siguiente:

$$f_{BDeu}(\zeta : D) = \log(p(\zeta)) + \sum_{i=1}^n \sum_{j=1}^{q_i} \left(\log \left(\frac{\Gamma\left(\frac{\eta}{q_i}\right)}{\Gamma\left(N_{ij} + \frac{\eta}{q_i}\right)} \right) \right) + \sum_{k=1}^{r_i} \log \left(\frac{\Gamma\left(N_{ijk} + \frac{\eta}{r_i q_i}\right)}{\Gamma\left(\frac{\eta}{r_i q_i}\right)} \right) \quad [27]$$

4.5.2. Métricas basadas en la teoría de la información

Las métricas basadas en la teoría de la información representan otra opción de evaluar la calidad de un GDA respecto del conjunto de datos en base a conceptos relativos al campo de la codificación y de la teoría de la información.

En la teoría de la información, la codificación tiene como objetivo representar los mensajes utilizando el menor número de elementos posibles. En concreto, el principio MDL del inglés Minimun Descriptions Length trata de minimizar la longitud de la descripción del modelo y su capacidad para representar el conjunto de los datos.

En las redes bayesianas modelos muy complejos serán aquellos donde los nodos estén densamente conectados y serán redes muy precisas, bastante ajustadas a los datos. El caso extremo sería un grafo completo. Pero hay que tener en cuenta que redes muy complejas pueden dificultar seriamente la comprensión del modelo, aumentar notablemente el tiempo de computación y producir un sobreajuste de los datos, por lo que en realidad se buscan redes más sencillas y menos precisas. El principio MDL tratará de obtener un equilibrio entre la complejidad del modelo y la precisión frente a los datos.

La longitud de descripción mínima en las redes bayesianas incluye la longitud requerida para representar la red, además de la longitud necesaria para representar los datos dado el modelo $L(D/G)$. Bouckaert (1995) presenta un cálculo de esta expresión a través de la negación del logaritmo de la función de verosimilitud de G dado D , definiendo esta métrica basada en el log_verosimilitud o LL (Log Likelihood). Si estimamos los parámetros con la frecuencia relativa, su formulación matemática se puede expresar de la siguiente manera:

$$f_{LL}(\zeta : D) = \sum_{i=1}^n \sum_{j=1}^{q_i} \sum_{k=1}^{r_j} N_{ijk} \log \left(\frac{N_{ijk}}{N_{ij}} \right) \quad [28]$$

La fórmula anterior se puede interpretar como la cantidad de bit que son necesarios para representar a D . Se puede afirmar que cuanto mayor sea la log-verosimilitud mejor modela el grafo G la distribución en los datos D .

La longitud de la descripción de la red depende del número de parámetros libres de la factorización de la probabilidad conjunta. Es un término que se denomina complejidad de la red

$C(\zeta)$: es este valor junto con una función de penalización $f(N)$ la que permite definir las métricas más utilizadas en las aplicaciones reales.

$$C(\zeta) = \sum_{i=1}^n (r_i - 1)q_i \quad [29]$$

Si la función de penalización es $f(N) = \frac{1}{2} \log(N)$ obtenemos la métrica MDL o métrica BIC (Bayesian Information Criterion) que propuso Gideon (1978).

$$f_{BIC}(\zeta : D) = f_{LL}(\zeta : D) - \frac{1}{2} \log(N) C(\zeta) \quad [30]$$

Si $f(N) = 1$ entonces la métrica es la propuesta por Akaike (1974) denominada AIC (Akaike Information Criterion)

$$f_{AIC}(\zeta : D) = f_{LL}(\zeta : D) - C(\zeta) \quad [31]$$

Otra propuesta de métrica más reciente que se puede utilizar es la MIT (Mutual Information Test) propuesta por De Campos (2006) y que tiene como expresión:

$$f_{MIT}(\zeta : D) = \sum_{\substack{i=1 \\ Pax_i \neq \emptyset}}^n 2NI(X_i; Pax_i) - \sum_{j=1}^{q_i} \chi_{\alpha, l_{i\sigma_i^*(j)}} \quad [32]$$

Donde la primera parte de la ecuación incorpora la información mutua entre una variable y sus padres y lo que trata es de medir el grado de interacción entre las variables y, la segunda parte de la expresión es un término para penalizar la red en base a una serie de test de independencia basados en la χ^2 .

4.6. Algoritmos de búsqueda y aprendizaje

El considerable número de combinaciones que se producen a la hora de buscar la mejor estructura del modelo de red bayesiana ha originado que, antes de verse imposibilitado a dar una solución óptima, se considere ofrecer soluciones que satisfagan al usuario que las demanda. Esta imposibilidad de cubrir todo el espacio de búsqueda ha motivado que los algoritmos de tipo heurístico sean cada vez más empleados como valiosas herramientas donde los algoritmos exactos no son capaces de encontrarlas.

Existen multitud de algoritmos que se han desarrollado en las últimas décadas. Algunos ejemplos interesantes que se utilizan en la gran mayoría de programas son los algoritmos de Búsqueda Tabú, el Enfriamiento Simulado y los Algoritmos Genéticos. Existen otros enfoques que también se muestran muy eficaces: GRASP (Greedy Randomized Adaptive Search Procedure), Algoritmos Meméticos, VNS (Búsqueda por Entornos Variables), Colonias de Hormigas, Estimación de Distribuciones (EDA), Programación por restricciones o la Búsqueda Dispersa, entre otras propuestas que se han realizado en este campo del conocimiento tan fecundo.

A continuación, se describen someramente los principales algoritmos utilizados y que se encuentran disponibles en la mayor parte de paquetes estadísticos.

4.6.1. Algoritmo K2

Este algoritmo basado en búsqueda y optimización de una métrica bayesiana es considerado como el predecesor y fuente de inspiración para las generaciones posteriores. El algoritmo K2 realiza una búsqueda voraz muy eficaz para encontrar una red de calidad en un tiempo razonable (Cooper y Herskovitz, 1992). Para llevar a cabo la búsqueda, previamente el algoritmo ordena los nodos (variables de entrada) de forma que los posibles padres de una variable aparecen en el orden antes que ella misma, lo que evita la generación de ciclos. Esta restricción es bastante fuerte, pero al proporcionar un orden hace que el algoritmo sólo tenga que buscar los padres posibles entre las variables predecesoras.

Partiendo de que el conjunto vacío es el conjunto de padres para cada variable y siguiendo un orden establecido pasa a procesar cada variable y calcula la ganancia que se produce en la medida al introducir una variable como parente. El proceso se repite para cada nodo mientras el incremento de calidad supere un cierto umbral preestablecido. En el algoritmo también se puede limitar el número de padres de cada variable. Considerando la descomponibilidad de la red, la contribución que realiza cada variable a la calidad de la red viene dada por la siguiente expresión:

$$\sum_{j=i}^{r_i} \sum_{k=1}^{s_i} N_{ijk} \log \frac{N_{ijk}}{N_{ik}} \quad [33]$$

El pseudocódigo del algoritmo K2 para aprendizaje estructural de redes bayesianas requiere una muestra de datos de un conjunto previamente ordenado de nodos.

El pseudocódigo es el siguiente:

$D = \{X_1, \dots, X_n\}$ y un grafo desconexo B .

```
1: for i = 1 : n - 1 do
2: S = score ( D , B );
3: for k = i + 1 : n do
4: M = score ( D , B ∪ {i → k} );
5: if M > S then
   S = M; B ( i , k ) = 1;
6: end if
7: end for
8: end for
```

Nota: Un grafo dirigido acíclico B y su medida de calidad S .

4.6.2. Algoritmo B

En este algoritmo se elimina el problema de la dependencia de la ordenación previa de los nodos lo que añade mayor complejidad, Buntine, (1991). Este algoritmo apareció muy temprano y al igual que su predecesor utiliza un esquema voraz. El orden de complejidad computacional es mayor. Al igual que el K2 se inicia con padres vacíos y en cada etapa se añade aquel enlace que maximiza el incremento de calidad eliminando aquellos que producen ciclos. El proceso se detiene cuando la inclusión de un arco no representa ninguna ganancia o bien se obtiene una red completa.

El pseudocódigo del algoritmo B para el aprendizaje estructural de redes bayesianas es el siguiente. Se requiere de una muestra de datos de un conjunto de nodos $D = \{X_1, \dots, X_n\}$ y un grafo desconexo B .

```
1: for i = 1 : n - 1 do
2: S = score ( D , B );
3: for k = i : n do
4: M = score ( D , B ∪ {i → k} );
5: if M > S ∧ acyclic ( B ∪ {i → k} ) then
   S = M; B ( i , k ) = 1;
6: end if
7: end for
8: end for
```

Nota: Un grafo dirigido acíclico B y su medida de calidad S .

4.6.3. Algoritmo Hill Climbing

El algoritmo Hill Climbing (HC), que se ha traducido como Ascensión de Colinas, en su forma básica realiza la selección del siguiente nodo a expandir de acuerdo con alguna medición heurística que

permite estimar la distancia que queda por recorrer hasta la meta. Se trata de un procedimiento de búsqueda que parte de la solución s y realiza movimientos desde esa solución a otras soluciones vecinas según una definición de entorno. Se le denomina también mejora iterativa porque cada nuevo movimiento requiere que la solución sea mejor que la anterior. Una rama no tiene por qué ser explorada hasta agotarse, sino que el proceso de expansión terminará en el momento en que se encuentra un nodo sucesor que no mejora el estado actual.

Es un algoritmo local que utiliza el concepto de vecindad clásica y que parte de una solución inicial y, a partir de ésta, se calcula el nuevo valor utilizando todas las soluciones vecinas a la solución actual, seleccionando el vecino que mejor solución presenta. Es decir, este algoritmo finaliza cuando no existe ningún vecino que pueda mejorar la solución vecina.

Una variante muy útil y muy empleada consiste en considerar todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado. A este método se le denomina ascensión por la máxima pendiente o búsqueda del gradiente.

Este algoritmo, como todos los utilizados en la búsqueda de la estructura de la red bayesiana, es un método que aprovecha la descomponibilidad de las métricas cuando tienen que recalcular las modificaciones que se realizan en los nodos vecinos.

4.6.4. Simulated Annealing

Este algoritmo, Simulated Annealing (Recocido simulado o enfriamiento simulado), según Kirkpatrick et al. (1983) y Cerny (1985), es una de las más antiguas metaheurísticas que ya contiene, como idea fundamental, una estrategia implícita para escapar de los mínimos locales. La idea de este algoritmo es permitir movimientos que conduzcan a soluciones de peor calidad, pero que permitan escaparse de los mínimos locales. La probabilidad de aceptación de los movimientos va disminuyendo durante la búsqueda y va a depender de la solución inicial, $f(s)$ y de un parámetro de control que modula que proporción de malas soluciones que se aceptan.

El algoritmo comienza generando una solución inicial. Esta primera solución puede ser aleatoria o creada a través de procedimientos heurísticos. También se inicializa un parámetro de temperatura T . En cada solución se escoge una solución s' del entorno $N(s)$ aceptándose la solución en función de su valor objetivo, $f(s')$, el valor objetivo de s , $f(s)$ y la temperatura. Si el valor objetivo de s' es mejor entonces se remplaza. La forma de elegir este enfriamiento es crucial en este algoritmo.

El pseudocódigo del algoritmo Simulated Annealing es el siguiente:

```

Procedure RecocidoSimulado()
    s ← GenerarSoluciónInicial()
    T ← T0
    While no se cumplan las condiciones de parada do
        s' ← EscogerAleatoriamente(N(s))
        if f(s') < f(s) then
            s ← s'
        else
            Aceptar s' como nueva solución con probabilidad p(T,s',s) end if
            Actualizar (T)
        End while

```

4.6.5. Tabu Search

La metaheurística Tabú Search (Búsqueda Tabú) está desarrollada en Glover (1986), Bouckaert (1995) y en Glover y Laguna (1997). Ese algoritmo se estructura en tres fases: preliminar, intensificación y diversificación.

En la fase de búsqueda preliminar, partiendo de la solución inicial s evalúa todas las soluciones del entorno de s , $N(s)$ encontrando un punto del espacio s' mejor que s o incluso peor que s . Esta posibilidad puede generar bucles ya que en los siguientes movimientos puede retroceder de s' a s generando ciclos. Para evitar estos ciclos el algoritmo crea una lista de movimientos prohibidos de longitud l . En esta lista el primer elemento en entrar es el primero en salir, lo que puede interpretar como una memoria a corto plazo. El movimiento de s' a s queda prohibido durante los siguientes l movimientos.

La fase de intensificación empieza con la mejor solución encontrada limpiando la lista tabú y procediendo como en la primera fase. En la fase de diversificación vuelve a limpiarse la lista tabú y se colocan en la lista los movimientos realizados hasta el momento con mayor frecuencia. La segunda fase actúa como una lupa en las regiones más prometedoras que se encontraron en la primera fase mientras que la fase de diversificación impulsa al algoritmo a explorar regiones novedosas.

El pseudocódigo del algoritmo Tabú Search es como sigue:

Procedure *TabuSearch()*

```
s ←— GenerarSoluciónInicial()
InicializarListasTabú(TL1,...,TLr)
k ←— 0
while no se cumplan las condiciones de parada do
    ConjuntoPermitido(s,k)←— { s' ∈ N(s) | s' no viola una condición tabú o
        satisface algún criterio de aspiración }
    s ←— ElegirMejor(ConjuntoPermitido(s,k))
    ActualizarListaTabúYCriteriaosAspiración()
    k ←— k+1
end while
```

Algunas extensiones de este algoritmo derivan de considerar la lista tabú de diversas maneras, para beneficiarse de la historia de la búsqueda. Por ejemplo, se han considerado memoria de soluciones élite donde sólo están las mejores, la novedad en los atributos, el número de veces que ha sido visitada una solución, la calidad o la influencia (aquellas decisiones que se muestran más importantes).

Otra solución importante desarrollada sobre este algoritmo con la idea de escapar de mínimos locales es el algoritmo denominado Búsqueda Tabú Reactive (Reactive Tabu Search), Battiti (1996). Este autor considera que la historia de la búsqueda se emplee como una guía para ajustar los parámetros de la heurística para poder detectar mínimos locales y poder actuar reajustando los parámetros de la búsqueda para poder solventarlos.

4.7. Algoritmos basados en test de independencia

Los test de independencia comprueban de forma eficiente las distintas dependencias combinando las variables, aunque se suelen considerar conjuntos reducidos de las variables si el número de variables es considerable. La idea de estos métodos es satisfacer el mayor número de independencias presentes en los datos, Neapolitan (2004).

Cuando estas dependencias se han detectado se expresan en el grafo. Al referirnos a las dependencias e independencias de variables nos referimos a conjuntos de variables. Sean, por ejemplo, tres conjuntos disjuntos de variables X, Y y Z. Se dice que X es condicionalmente independiente de Y dado Z y lo expresamos como $I(X,Y/Z)$, si y sólo si, se cumple la siguiente igualdad:

$$p(x/z, y) = p(x/z) \Leftrightarrow p(x, y/z) = p(x/z)p(y/z) \quad [34]$$

para todos los valores posibles x, y, z . En cualquier otro caso, X e Y se dicen condicionalmente dependientes dado Z , y lo expresamos como $D(X, Y/Z)$.

La independencia condicional lleva implícita la idea de que una vez conocida Z el conocimiento de Y no altera la probabilidad de X . En otras palabras, se afirma que el conocimiento de Y no añade información alguna sobre el de X .

Uno de los algoritmos más populares basados en test de independencia es el algoritmo PC. Este algoritmo utiliza una medida denominada información mutua. El concepto de información mutua se deriva de la entropía y proporciona un excelente criterio de medida de la independencia a partir de la información que una variable tiene sobre otra. La expresión matemática de la información mutua de dos variables es la siguiente:

$$MI(X, Y/Z) = \sum_z p(z) \sum_{x,y} p(x, y/z) \log \frac{p(x, y/z)}{p(x/z)p(y/z)} \quad [35]$$

Las probabilidades se estiman a partir de las frecuencias relativas que se observan en el conjunto de datos y son los estimadores de máxima verosimilitud de la probabilidad.

Una vez que se ha establecido esta medida podemos comprobar la hipótesis:

$I(X, Y/Z)$ a través del estadístico:

$$G = 2N MI(X, Y/Z) \quad [36]$$

En esta fórmula N representa el tamaño de la muestra y el estadístico G , bajo la hipótesis de independencia se distribuye como una distribución χ^2 con grados de libertad igual a $(r_x - 1)(r_y - 1)r_z$ donde r_x, r_y y r_z representan la cardinalidad de cada variable.

En el aprendizaje paramétrico se calculan los parámetros de acuerdo con la estructura del tipo de grafo aprendido en la fase estructural y la base de datos de la que disponemos: tablas de probabilidad, medias, varianzas, etcétera.

En la literatura estadística existen diferentes propuestas para detectar aquellas redes que mejor representan a los datos (concepto de bondad de ajuste) y que al mismo tiempo penalizan las estructuras más complejas (concepto de penalización). A cada red se le asigna una medida de

calidad que es función de la probabilidad a posteriori. Esta distribución de probabilidad a posteriori $p(B/D)$ se expresa por la siguiente fórmula:

$$p(B \mid D) = p(M, \theta \mid D) = \frac{p(M, \theta, D)}{p(D)} \propto p(M)p(\theta \mid M)p(D \mid M, \theta) \quad [37]$$

Donde $B = (M, \theta)$ es la red con el grafo dirigido M y θ representa a los parámetros.

Si las redes bayesianas son multinomiales y se asumen ciertas hipótesis de las distribuciones a priori de los parámetros y una probabilidad inicial uniforme para todos los modelos, se puede encontrar una medida de calidad bayesiana que se basa en el logaritmo de la verosimilitud y que incluye un término de penalización, Heckerman, (1996).

Otra medida de calidad para redes multinomiales descrita en Lam y Bacchus (1994) es la denominada MDL (Minimum Description Length) donde se evalúan a la vez la verosimilitud de los datos de entrenamiento D y la simplicidad, maximizando la expresión:

$$MDL(B \mid D) = \sum_{i=1}^N \log p_B(D) - r_B * \log(N/2) \quad [38]$$

Donde r_B representa el número de parámetros libres asociados a la función de distribución conjunta.

Otras medidas de calidad basadas de la teoría de la información se encuentran descritas en Cheng et al. (1997). Algunas otras medidas maximizan la verosimilitud condicional (aprendizaje discriminativo) pero por ahora sólo son aplicables a problemas de baja dimensionalidad dado que presentan graves problemas de eficiencia computacional, Grossman y Domingos (2004)

Cuando las redes son gausianas, si se considera que la distribución de parámetros es del tipo normal-Wishart se obtienen medidas de calidad similares a las anteriores. Para más detalles véase Geiger y Heckerman (1994).

4.8. Clasificadores basados en Redes Bayesianas

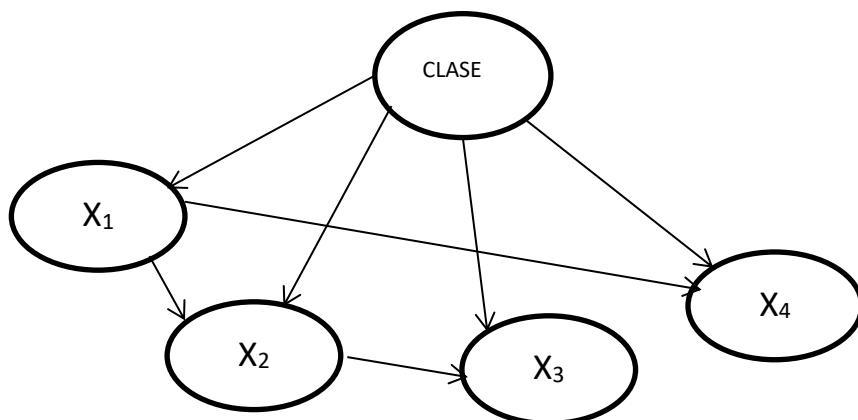
4.8.1. Algoritmo TAN

A la hora de enfrentarse con la construcción de un clasificador bayesiano teniendo en cuenta las dependencias entre las variables involucradas en el problema, existen alternativas más sencillas que enfrentarse a la construcción de una red bayesiana sin restricciones. En general, podemos

decir que estas alternativas representan una extensión del clasificador Naïve-Bayes con ciertas modificaciones estructurales. La forma de proceder es construir una estructura que refleje algunas relaciones de dependencia entre los atributos manteniendo al margen la variable clase.

El algoritmo TAN del inglés Tree Augmented Network fue propuesto por Friedman y colaboradores en 1997. Este algoritmo consistió en una adaptación del algoritmo que propuso Chow-Liu (1968) El TAN utiliza el concepto de cantidad de información mutua condicionada a la variable clase, en lugar de la cantidad de información mutua en la que se basa el algoritmo de Chow-Liu. En el modelo TAN todos los atributos tienen como padre a otro atributo como mucho, además de la clase en sí, de forma que cada atributo obtiene un arco aumentado apuntando a él.

FIGURA 4. MODELO GRÁFICO DEL CLASIFICADOR TAN



Dadas las variables discretas X e Y y la clase C , la cantidad de información que la variable Y nos proporciona sobre la variable X dada la variable clase, es calculada por la siguiente expresión:

$$I(X, Y / C) = \sum_{x,y,c} p(x, y / c) \log \frac{p(x, y, c)}{p(x / c) p(y / c)} \quad [39]$$

Cuando aprende la estructura del árbol entre todos los atributos, el algoritmo TAN añade la variable clase y la hace padre de todas las variables.

En Friedman y Goldszmidt (1996) se presenta un algoritmo que consta de cinco pasos:

1. En primer lugar se calcula $I(X_i; X_j / C)$ con $i < j$; $i, j = 1, 2, 3, \dots, n$. Estos valores se estiman a partir de la muestra.

2. Obtenidos los valores del paso 1 se construye un grafo no dirigido completo donde los nodos se corresponden con los atributos X_1, X_2, \dots, X_n . Se asigna a cada arco entre los nodos X_i, X_j , un peso dado por la $I(X_i; X_j | C)$.
3. Se aplica el algoritmo de Krustal al grafo construido en el paso anterior con el objetivo de construir un árbol expandido máximo. Este algoritmo parte de los $n(n-1)/2$ pesos del grafo completo para construir un árbol con todos los nodos de tal forma que la suma de los pesos de los nodos sea máxima. Para lograr esto opera en tres pasos en el que primeramente se asignan las dos aristas de mayor peso al árbol a construir. Seguidamente se examina la siguiente arista de mayor peso y se añade al grafo a no ser que forme un ciclo en cuyo caso se descarta y se busca la siguiente arista de mayor peso. Este paso se repite hasta que se hayan asignado $n - 1$ aristas.
4. El cuarto paso consiste en transformar el árbol no dirigido resultado del paso anterior en uno que esté dirigido. Para lograr esto, se escoge un nodo cualquiera como raíz, asignando direcciones a todas las aristas a partir de él.
5. El último paso para construir un modelo TAN es añadir la clase C y un arco desde C a cada nodo. Según se ha demostrado, si estamos en un contexto en el cual los datos de entrenamiento han sido generados por una estructura TAN, el algoritmo visto anteriormente es asintóticamente correcto, lo que significa que si la muestra es suficientemente grande el algoritmo recuperará la estructura que generó los datos.

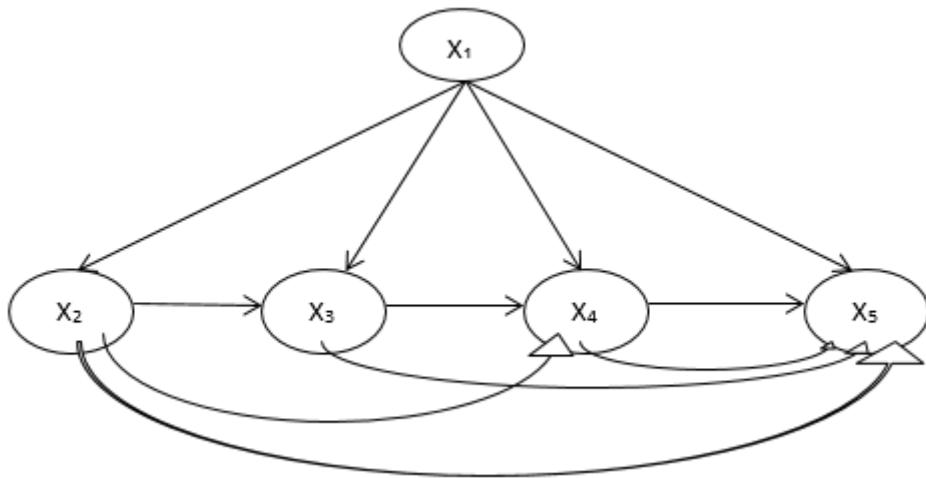
También se asegura que la estructura de red obtenida contiene la máxima verosimilitud del conjunto de todas las posibles estructuras TAN (Hernández, 2004). Por otra parte la complejidad de este algoritmo es $O(n^2 n)$, siendo n el número de atributos y N el tamaño del conjunto de entrenamiento.

Keogh y Pazzani (1999) proponen un algoritmo voraz que va añadiendo arcos a una estructura Naïve-Bayes. En cada uno de los pasos se añade un arco que mejore en mayor medida el porcentaje de instancias bien clasificadas, manteniendo la condición de que en la estructura final cada variable no tenga más de un parente.

4.8.2. Clasificadores bayesianos K-dependientes

Un paso más en la relajación del modelo Naïve-Bayes es también la generalización que realiza Sahami (1996) mediante lo que se denomina clasificadores k-dependientes (KDB)

FIGURA 5. MODELO GRÁFICO DEL CLASIFICADOR K DEPENDIENTE



En esta nueva estructura de red bayesiana se permite a cada variable X_i de la base de datos tener un máximo de k atributos. Su formalismo matemático sería el siguiente:

$$Pa(X_i) = \{C, X_{pa_i}\} \quad [40]$$

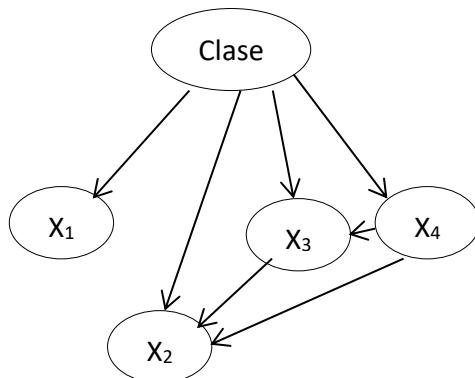
Donde X_{pa_i} es un conjunto máximo de k atributos y la clase carece de padres.

En este clasificador si variamos el valor de K podemos disponer de todas las combinaciones de dependencias de variables inductoras.

4.8.3. Naïve Bayes aumentado (BAN)

Este tipo de estructura recoge la organización de Naïve Bayes, pero ahora se ve aumentada con arcos entre todas las variables, con la única limitación de que no formen ciclos. Este tipo de estructura puede representar cualquier forma de red bayesiana.

FIGURA 6. MODELO GRÁFICO DEL CLASIFICADOR BAN.



4.8.4. Average One-Dependence Estimators (AODE)

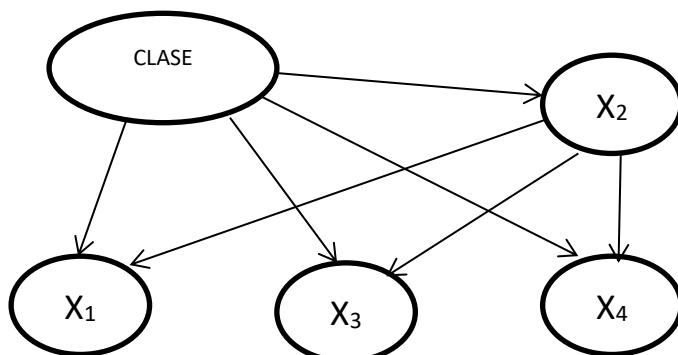
Este clasificador también está basado en el modelo Naïve Bayes, pero en esta propuesta se incrementa la estructura mediante arcos relajando la suposición de independencia de las variables dada la clase, que impone el método original.

Este clasificador diseñado por Webb et al. (2005) está siendo muy utilizado debido a su alta eficiencia y el bajo error en problemas de clasificación.

Este clasificador utiliza una estructura que está basada en el concepto de clasificador 1-dependiente de Sahami (1996). Al igual que el algoritmo TAN, cada variable tiene como padre a la variable clase y como máximo a otro atributo. Sin embargo, en la selección de modelos este clasificador utiliza un conjunto de modelos, Dietterich (2000), sobre los cuales calcula una media ponderada para obtener la predicción definitiva.

De entre todos los posibles modelos de clasificadores 1-dependientes para un conjunto de datos, el clasificador AODE emplea un subconjunto que está formado por aquellos modelos en los que existe un atributo padre, que en este caso recibe el nombre de súper padre dado que hace de padre del resto de atributos. Esta estructura fue desarrollada por Keogh y Pazzani (1999 y 2002) y se conoce por sus siglas en inglés SPODE (Superparent One-Dependence Estimators). El número total de estructuras que utiliza SPODE es n , una por cada variable presente en la base de datos, en el que cada uno de ellos hace una vez de súper padre.

FIGURA 7. GRAFO DE UN CLASIFICADOR AOED.1-DEPENDIENTE DE TIPO SPODE



Para clasificar una instancia nueva $x = (x_1, x_2, \dots, x_n)$ a través del conjunto de los n modelos SPODE (ensemble) utilizamos la siguiente expresión

$$\arg \max_{c \in \Omega_c} \left(\sum_{i=1}^n P(c, x_i) \prod_{j=1, j \neq i}^n P(x_j | c, x_i) \right) \quad [41]$$

Algunas contribuciones se han orientado a la mejora de este modelo explotando la versatilidad de los modelos ensemble. Por otra parte, algunos trabajos se han dirigido a plantear aproximaciones a la selección de modelos y a la ponderación de los mismos.

Las diferentes aportaciones respecto a las estrategias de selección de modelos a través de métodos wrapper se encuentran en Zheng y Webb. (2007) y Yang et al. (2007). Otra línea de investigación trata de clasificar nuevas observaciones realizando una selección de modelos perezosa (Lazy), Zheng y Webb. (2007).

Otras aproximaciones se centran en la ponderación o pesos que se da a cada modelo en el proceso de agregación en lugar de la ponderación uniforme que utiliza AODE. Yang et al. (2007) y Jiang y Zhang (2006) utilizan una media ponderada. Otras estrategias se han encaminado a utilizar ponderaciones más avanzadas a través de modelos bayesianos (Bayesian model averaging), Hoeting et al. (1999) o mediante mixturas lineales, Cerquides y López (2005).

Una aportación interesante la realizan Flores et al. (2009) que proponen un clasificador híbrido para incluir la posibilidad de utilizar todo tipo de bases de datos. En esta propuesta los autores deciden considerar cada super-padre como discreto en su modelo correspondiente, en principio, a través de cualquier método de discretización.

Yang et al. (2007) realizan un amplio estudio donde analizan el rendimiento del clasificador original AODE y los comparan con distintas aproximaciones basadas en selección de modelos y ponderación de modelos y concluyen que el clasificador AODE presenta una notable robustez y estabilidad.

4.8.5. Enfoques semi naïve Bayes y clasificadores extendidos

En este epígrafe lo que se trata es de recoger algunas de las múltiples propuestas sobre el clasificador Naïve-Bayes que consiguen de una forma u otra mejorar su exactitud y que son recogidas en la literatura como enfoque semi Naïve-Bayes.

Para organizar de alguna manera estas diferentes perspectivas se sigue y se extiende la ya conocida de Webb y Pazzani (1998), donde el conjunto de las diferentes propuestas se organiza en función de las actividades pre/post proceso que realizan. De esta manera, el autor encuentra tres grandes grupos:

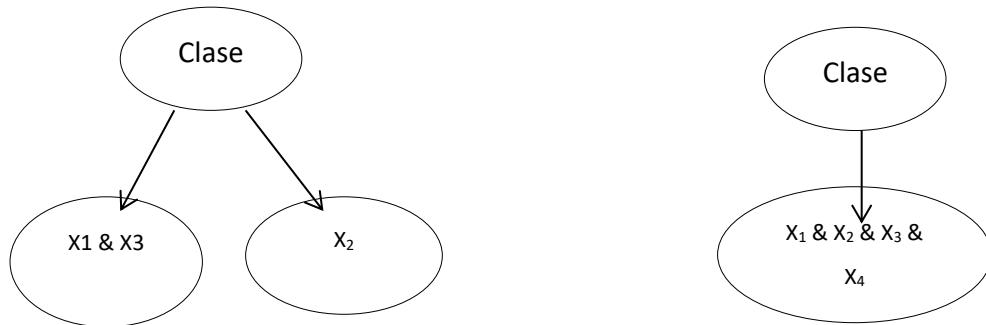
1. Aquellos procedimientos que procesan de alguna manera las variables antes de aplicar el Naïve-Bayes.
2. Enfoques que corrigen las probabilidades que arroja Naïve-Bayes.
3. Otros trabajos donde antes de procesar el algoritmo del Naïve-Bayes se seleccionan el conjunto de instancias.

En el primer grupo se encuentran todos los procedimientos relacionados con la selección de variables, que ya fueron expuestos en el documento del preprocesado. De forma resumida, recordemos que se pueden utilizar dos enfoques: filter y wrapper, Kohavi y John (1997).

En relación a la selección de variables existen algunas referencias bibliográficas interesantes con estrategias de tipo wrapper donde se utilizan diversos procedimientos como, por ejemplo, algoritmos genéticos en Liu et al. (2001) y en Inza et al. (2000). Este último autor, también emplea procedimientos de estimación denominados algoritmos de estimación de distribuciones (EDAs).

Otro enfoque con bastantes buenos resultados es el que sugiere Pazzani (1997). A través de un algoritmo voraz es capaz de detectar variables irrelevantes y variables dependientes antes de emplear el modelo Naïve-Bayes.

FIGURA 8. GRAFO DE UNA RED DE PAZZANI



Los algoritmos que se proponen para corregir las probabilidades producidas por el método Naïve-Bayes son muy variados, pero los principales se pueden sintetizar en los siguientes:

- Webb y Pazzani (1998) proponen un método que realiza un ajuste lineal del peso de las probabilidades de cada clase denominado APNBC (Adjusted Probability Naïve-Bayes). Este ajuste se puede expresar de la siguiente manera:

$$P(C = c_i | X_1 = x_1, \dots, X_n = x_n) \propto w_i P(C = c_i \prod_{k=1}^n P(X_k = x_k | C = c_i)) \quad [42]$$

- Ferreira et al. (2001) realizan una nueva aportación en el mismo sentido que los anteriores autores, pero en vez de ponderar la clase, permiten asociar pesos a particiones. En este procedimiento los ejemplos de cada variable son particionados de forma recursiva, con el objetivo de maximizar la entropía. El ajuste es el siguiente:

$$P(C = c_i | X_1 = x_1, \dots, X_n) \propto w_i P(C = c_i) \prod_{k=1}^n [P(X_k = x_k | C = c_i)]^{w(X_k, k, i)} \quad [43]$$

Donde $w(X_k, k, i)$ es la función de peso para la variable X_k .

- El algoritmo Iterative Bayes propuesto por Gama (2000) parte de la idea de mejorar la forma iterativa de calcular las probabilidades en el método original Naïve-Bayes. La forma que se propone es avanzar a través del algoritmo de ascenso de colinas HC (Hill Climbing). En cada iteración se utilizan las probabilidades actuales para todas las instancias del conjunto de entrenamiento a través de la siguiente función:

$$\frac{1}{N} \sum_{i=1}^N (1 - \max p(C = c_j | x^{(i)})) \quad [44]$$

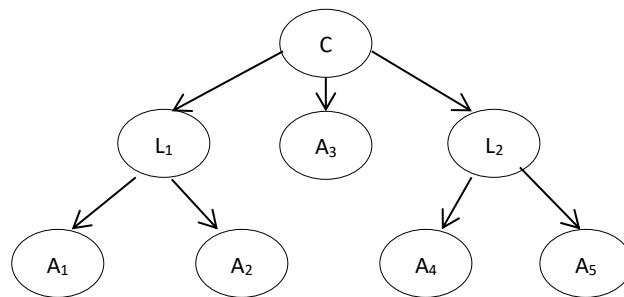
Donde N representa el conjunto de todos los ejemplos de la base de datos y j son los valores de la clase. Al contrastar los resultados de este algoritmo sobre 27 conjuntos de datos, los autores consiguen reducir el error medio en un 1,24%.

- El enfoque que utiliza Zaffalon (2002) se basa en conjuntos convexos de distribuciones de probabilidad (conjuntos credales) con lo que supera el requerimiento de los valores puntuales de Naïve Bayes. El Naïve Bayes Credal (NBC) representa las distribuciones de probabilidad como puntos que pertenecen a regiones geométricas cerradas y acotadas y que se encuentran descritas por restricciones lineales.
- El clasificador Robust Bayesian Classifier (RBC) que propusieron Ramoni y Sebastiani (2001) es un algoritmo desarrollado para aplicar cuando en la base de datos hay valores ausentes.
- Un trabajo interesante es el de Greiner y Zhou (2002) cuya propuesta consiste en encontrar los parámetros de la distribución a través de la maximización de la verosimilitud condicional en lugar de la verosimilitud de la muestra en una red bayesiana. Se tiene en cuenta la existencia de la variable a clasificar y se propone un algoritmo de descenso por el gradiente.

- En Robles (2003) se propone un nuevo algoritmo denominado Internal Estimation Naïve-Bayes (IENB), cuyo enfoque se centra en estimar las probabilidades necesarias para el clasificador Naïve-Bayes a través de una estimación por intervalo en lugar de la estimación puntual que se aplica en el método primitivo. La búsqueda se realiza con algoritmos heurísticos de optimización (EDAs) y está guiada por la exactitud de los clasificadores.

Podemos encontrar otros algoritmos basados en Naïve Bayes y que en alguna publicación se han denominado Clasificadores ingenuos extendidos, que también representan una buena opción de aprendizaje para resolver problemas de clasificación, como por ejemplo los Clasificadores Ingenuos Jerárquicos, Hierarchical Naïve Bayes (HBN) que están descritos en Langseth y Nielsen (2002). Una representación gráfica de este modelo se observa en la figura 9.

FIGURA 9. EJEMPLO DE HIERARCHICAL NAÏVE BAYES



En esta configuración se introducen variables latentes con el objetivo de relajar las hipótesis de independencia entre los atributos. Estos atributos son introducidos considerando la dependencia condicional entre ellos, seleccionando la pareja de atributos que consiga la mayor información mutua.

Otras aportaciones que se han desarrollado es introduciendo en el algoritmo restricciones estructurales derivadas del conocimiento previo o del orden de las variables. También Acid et al. (2005) proponen un procedimiento donde consideran clases equivalentes, tanto en términos de independencia condicional, como en términos de equivalencia, para reducir el espacio de búsqueda.

Existe una amplia literatura sobre mezclas o comités de expertos combinando las predicciones de diferentes clasificadores, Lanngseth y Nielsen (2006) y Webb et al. (2005).

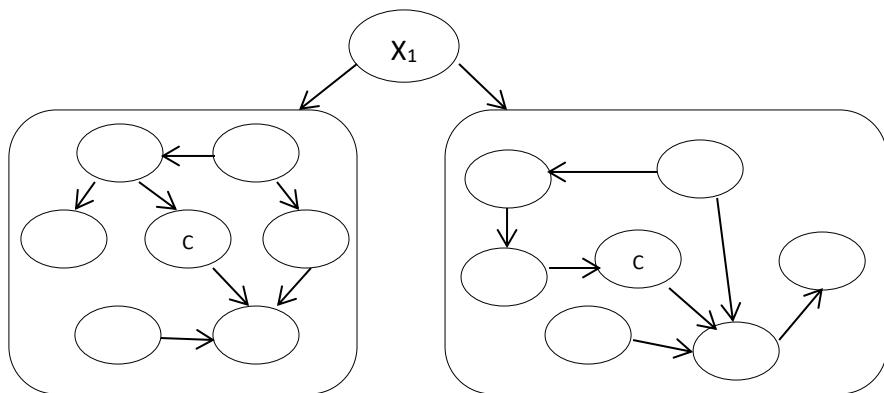
Para concluir este epígrafe reseñar que existe una amplia información sobre el aprendizaje de clasificadores bayesianos.

4.8.6. Multiredes bayesianas

Un avance en la modelización de datos a través de redes bayesianas es cuando consideramos un clasificador distinto para cada una de las clases tal y como se observa en la Figura 10, porque una conclusión que se extrae de la observación en la construcción de las estructuras de las propuestas anteriores, es que la relación es siempre la misma en cada una de las clases.

Esta consideración nos permite representar independencias asimétricas, Heckerman (1991). Este autor distingue dos tipos de asimetrías, una que podemos encontrar entre la variable a clasificar y los atributos, denominada asimetría de subconjunto y otra que el autor denominó hipótesis específica y que se origina cuando solo se considera la relación entre atributos. En la primera asimetría se construye una red bayesiana para cada valor de la clase a clasificar como se refleja en la Figura 10.

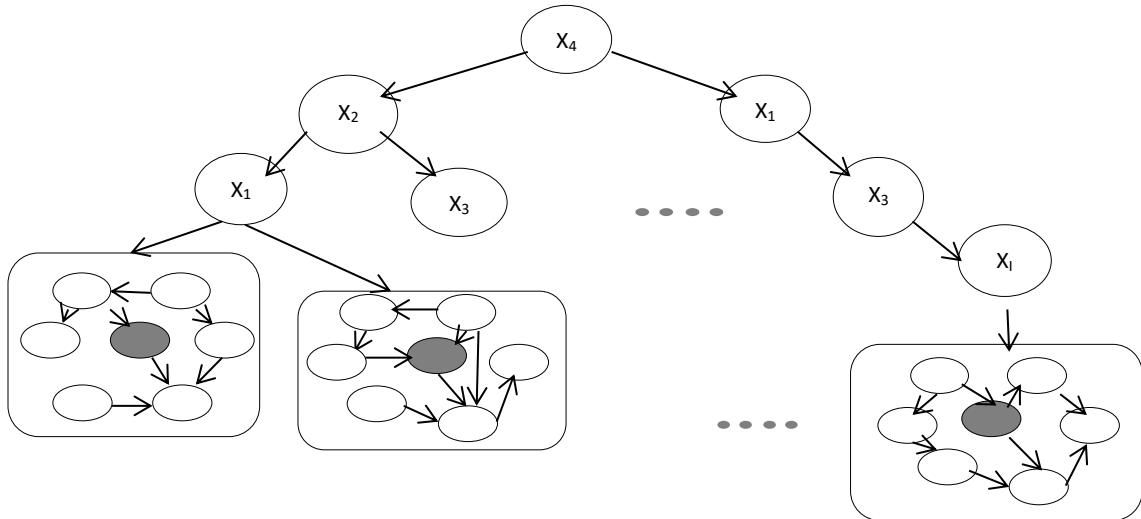
FIGURA 10. MULTIRRED BAYESIANA



Esta construcción es considerada como una extensión de las redes bayesianas, y el desarrollo y formalismo se puede encontrar en Geiger y Heckerman (1995).

Otro tipo de estructura multired que se encuentra en la literatura consultada es el que se conoce como redes recursivas y que se caracteriza por que se seleccionan diferentes atributos agrupados en forma de árbol donde en sus extremos tienen diferentes clasificadores bayesianos, como se puede apreciar en la Figura 11.

FIGURA 11. MULTIRED BAYESIANA RECURSIVA



4.8.7. Naïve Bayes extendido a través de un árbol de clasificación (NBtree)

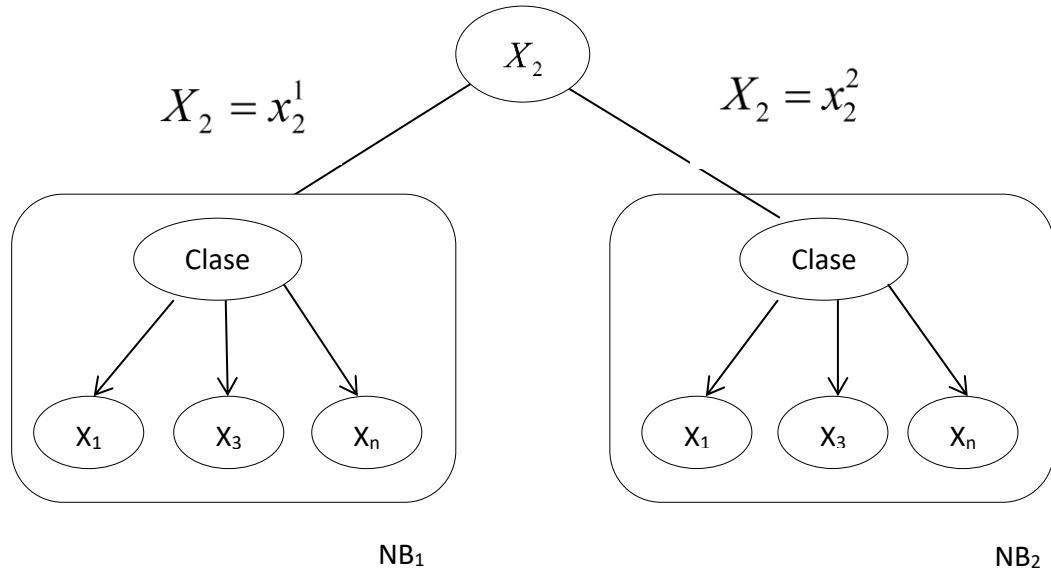
El Naïve Bayes Tree propuesto por Kohavi (1996) es un algoritmo híbrido entre un árbol de clasificación y el clasificador Naïve Bayes. También podemos considerarlo como un tipo especial de multired bayesiana recursiva donde las hojas son clasificadores Naïve Bayes. En realidad, lo que perseguía el autor al crear esta estructura era combinar las ventajas de los árboles de decisión, Quinlan (1993), y las virtudes del clasificador Naïve Bayes.

En esta estructura podemos observar las propiedades de la misma: (i) una primera observación es que, como en los algoritmos anteriores, cada nodo interno de la red representa una variable, (ii) estos nodos engendran tantos hijos o ramas salientes como estados tiene la variable, (iii) todas las hojas se encuentran en el mismo nivel, (iv) en cualquier camino que se transite desde la raíz hasta las hojas no existen variables repetidas.

Un algoritmo heurístico para estimación de esta estructura se encuentra en Peña et al. (2002).

Es evidente que en la construcción de la estructura NBtree se pueden generar nodos terminales que contengan un reducido número de instancias, por lo que algunas aportaciones intentan mejorar este problema. Una buena solución es el algoritmo LBR (Lazy Bayesian Rule) de Zheng y Webb (2000). Las pruebas que realizaron contrastando ambos enfoques sobre 29 conjuntos de datos del repositorio UCI demuestran la superioridad del algoritmo LBR sobre el Naïve BayesTree.

FIGURA 12. GRAFO DE UNA RED NBTREE



4.9. Tipos de Redes Bayesianas

Los diferentes tipos de redes bayesianas vienen determinadas por el carácter discreto o continuo de las variables involucradas en el modelo.

4.9.1. Redes bayesianas Gausianas

Cuando las variables siguen una distribución normal multivariante $N(\mu, \Sigma)$ decimos que la red es gausiana. La función de densidad conjunta viene determinada por la siguiente expresión:

$$f(x) = (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right\} \quad [45]$$

donde μ es el vector de medias n-dimensional, Σ es la matriz de covarianzas n x n, $|\Sigma|$ es el determinante de Σ , y μ^T denota la traspuesta de μ .

4.9.2. Redes bayesianas Multinomiales

En este tipo de red se considera que todas las variables son discretas, lo que implica que todas las variables tienen un número finito de posibles estados. También suponemos que las funciones de probabilidad de cada variable condicionada a sus predecesores (padres) es también multinomial y, por lo tanto, están especificadas en las diferentes combinaciones de estado de las variables involucradas. La reducción de parámetros a estimar es considerable.

4.9.3. Redes bayesianas Mixtas

Las redes bayesianas mixtas tienen un alto grado de complejidad a la hora de definirlas, aunque casos particulares han sido estudiados. En Jordan (1998) se describe un caso en el que se permite que una variable continua tenga padres con valores discretos. Otro ejemplo lo encontramos en Castillo et al, (1998) donde abordan un problema utilizando variables discretas y funciones Beta.

4.10. Ejemplo de Redes Bayesianas con WEKA

En este epígrafe, vamos a aplicar el programa WEKA, dado que contiene una gran cantidad de redes bayesianas.

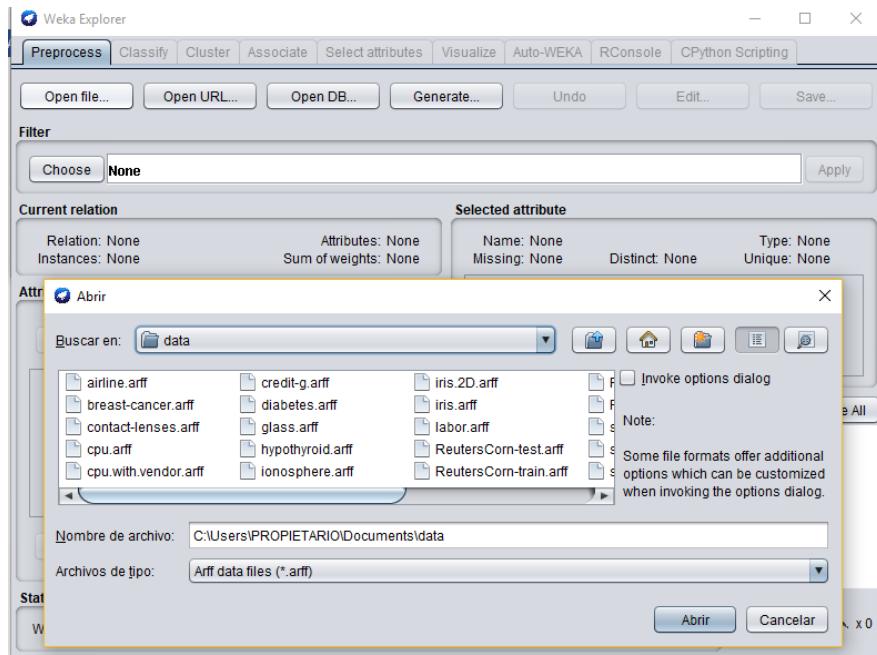
En primer lugar, una vez que se haya inicializado el programa WEKA, activamos la opción Explorer.

FIGURA Nº13: APPLICACIONES DE WEKA



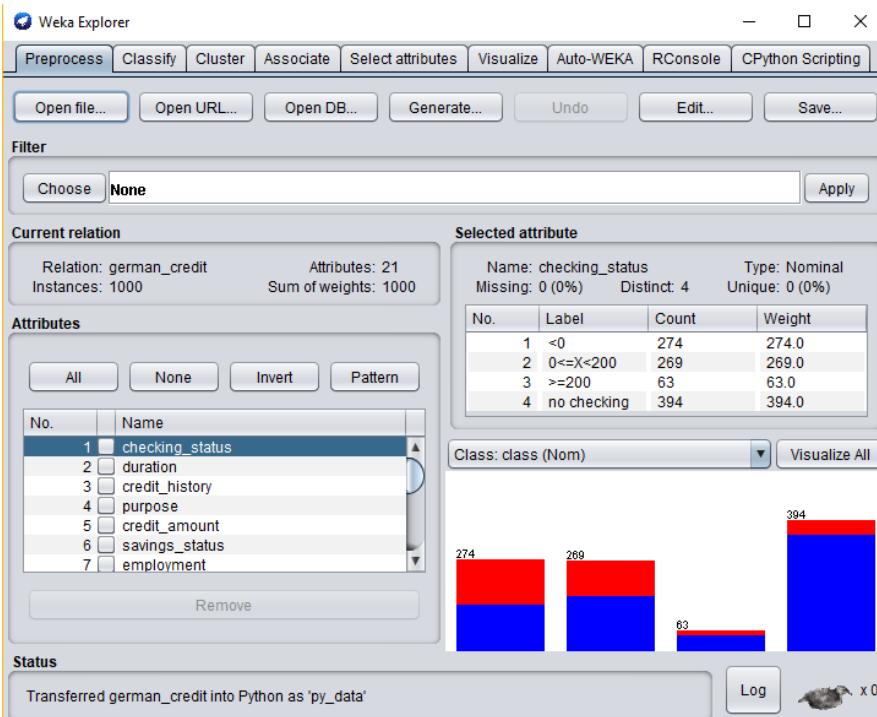
En primer lugar, abrimos el fichero con el que queremos trabajar que, en este caso, va a ser la base de datos de créditos bancarios denominada German Credit, a través del Menú/Open File donde podemos elegir el formato en el que tengamos los datos (arff, csv, xlsx...)

FIGURA Nº 14: MÉTODOS DE CLASIFICACIÓN Y REGRESIÓN EN WEKA



Una vez que se ha abierto el fichero el programa nos muestra un resumen de cada variable, además de un gráfico de barras.

FIGURA Nº 15: MENÚ DE WEKA



En el Menú/Classifiers/Bayes están recogidos los algoritmos de clasificación y de regresión disponibles en WEKA. Los diferentes procedimientos se activarán dependiendo de si la última variable del fichero es categórica o numérica.

FIGURA Nº 16: MÉTODOS DE CLASIFICACIÓN Y REGRESIÓN EN WEKA

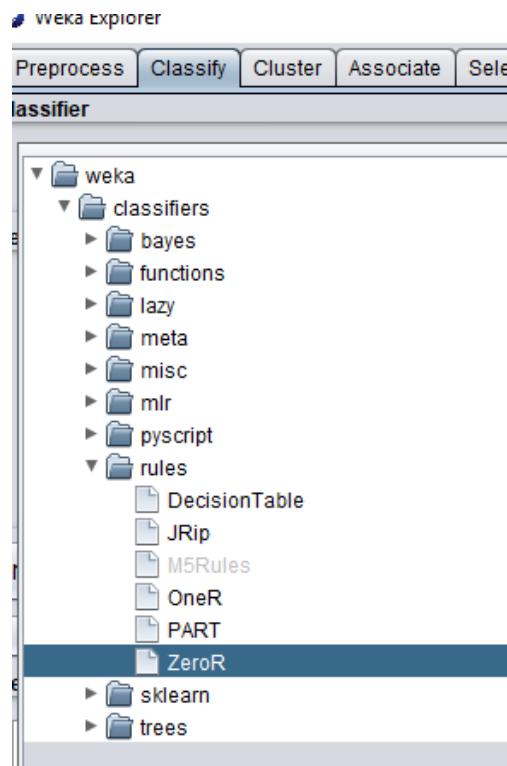
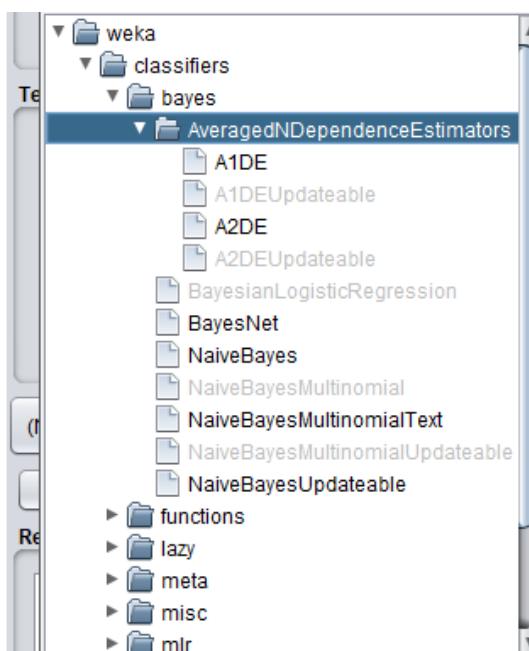


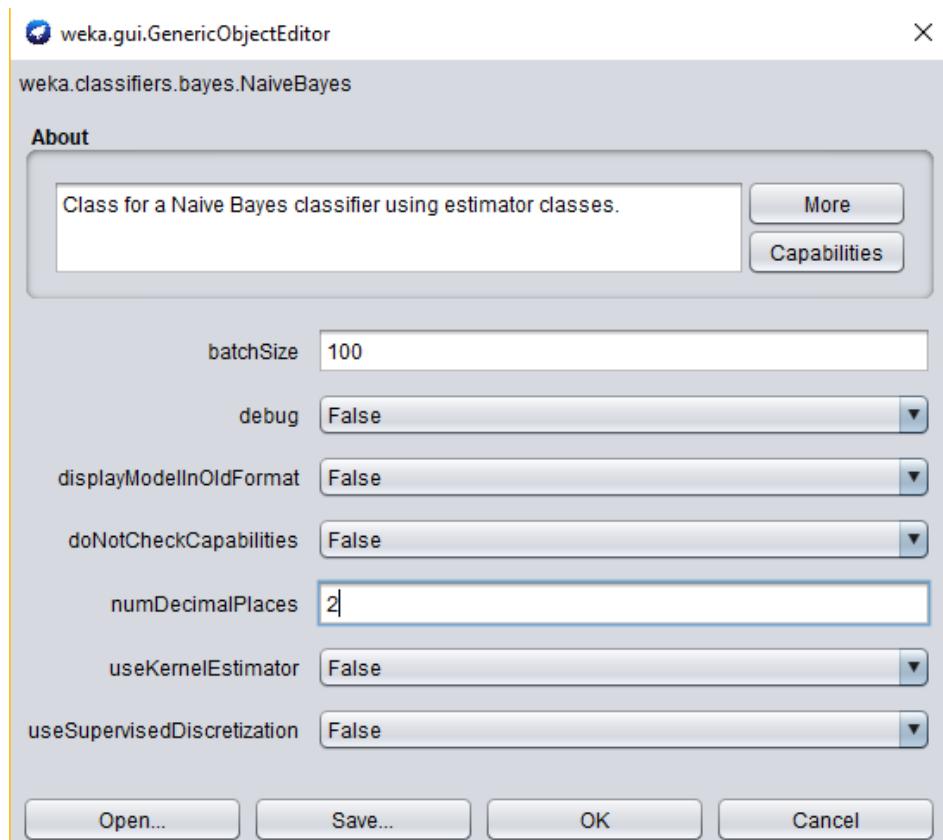
FIGURA Nº 17: MÉTODOS BAYESIANOS EN WEKA



En cada técnica que utilicemos tendremos la posibilidad de configurar diferentes parámetros para obtener el modelo más adecuado.

Vamos a ver primeramente los resultados que se obtienen con el algoritmo Naïves Bayes.

FIGURA Nº 18: CONFIGURACIÓN DE UN NAIVE BAYES



Parte de la salida que se obtiene al pulsar OK es la siguiente.

==== Stratified cross-validation ====

==== Summary ====

Correctly Classified Instances	754	75.4	%
Incorrectly Classified Instances	246	24.6	%
Kappa statistic	0.3813		
Mean absolute error	0.2936		
Root mean squared error	0.4201		
Relative absolute error	69.8801 %		
Root relative squared error	91.6718 %		
Total Number of Instances	1000		

== Detailed Accuracy By Class ==

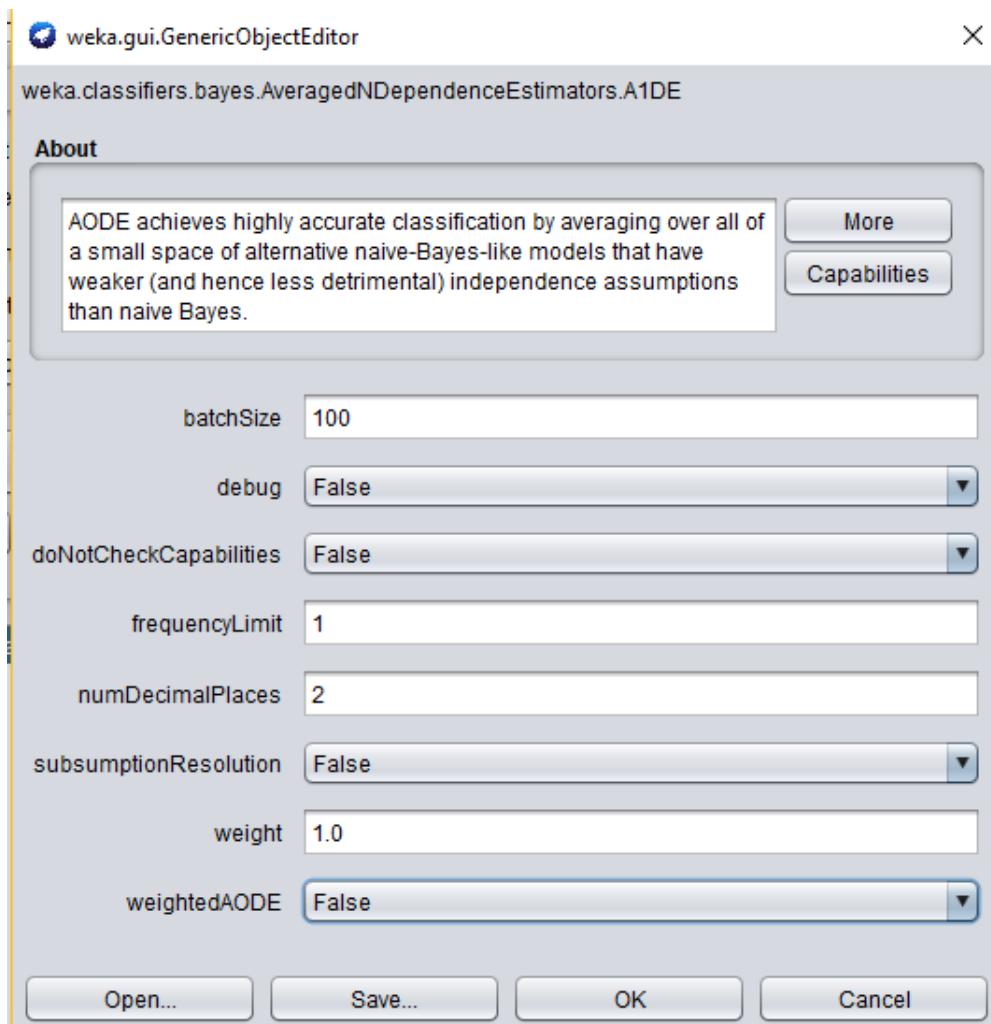
TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC	Class
0,864	0,503	0,800	0,864	0,831	0,385	0,787	0,891	good
0,497	0,136	0,611	0,497	0,548	0,385	0,787	0,577	bad
. 0,754	0,393	0,743	0,754	0,746	0,385	0,787	0,797	Weighted Avg

== Confusion Matrix ==

a b <- classified as
605 95 | a = good
151 149 | b = bad

Procedemos de igual forma para extraer los resultados con el resto de técnicas: configuración de los parámetros y obtención de los resultados.

FIGURA N° 19: CONFIGURACIÓN DEL MÉTODO AVERAGE ONE-DEPENDENCE ESTIMATOR



==== Detailed Accuracy By Class ====

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,884	0,523	0,798	0,884	0,839	0,397	0,789	0,892	good
0,477	0,116	0,638	0,477	0,546	0,397	0,789	0,584	bad
0,762	0,401	0,750	0,762	0,751	0,397	0,789	0,800	Weighted Avg.

==== Confusion Matrix ====

a b <-- classified as

619 81 | a = good

157 143 | b = bad

Redes bayesianas

Las redes bayesianas exigen configurar más parámetros que en los modelos anteriores. Podemos elegir diferentes algoritmos de búsqueda así como las métricas de evaluación. Con algunos métodos también se puede configurar el número de padres. También se puede activar el Manto de markov.

FIGURA N° 20: CONFIGURACIÓN DE UNA RED BAYESIANA

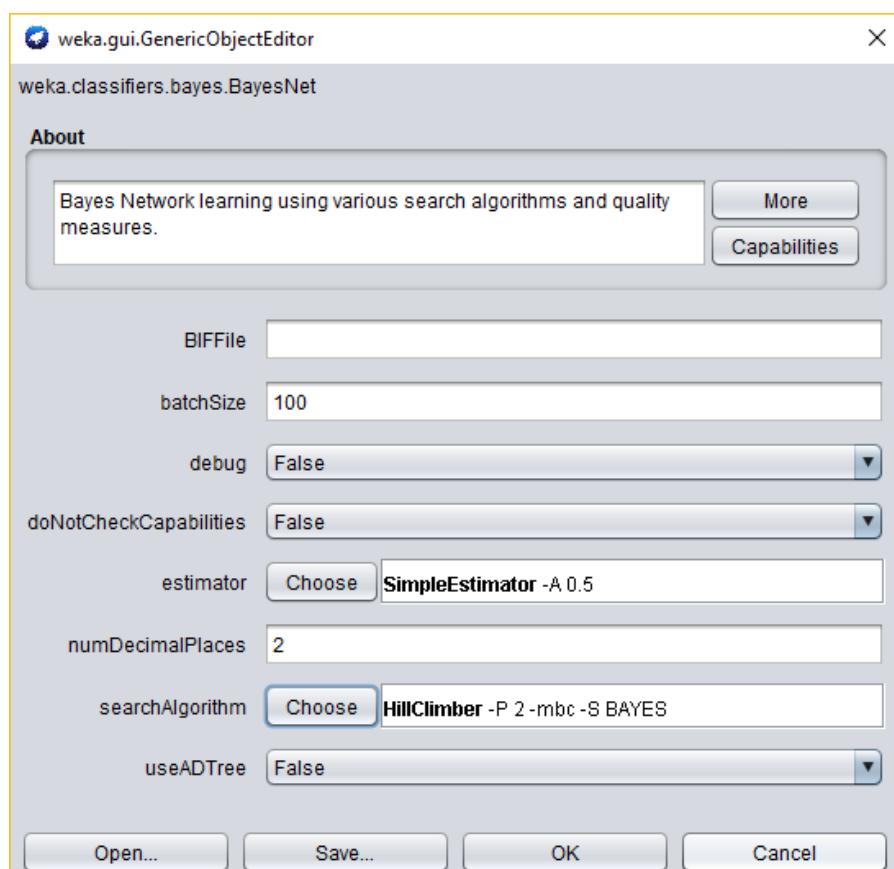


FIGURA N° 21: MÉTODO DE BÚSQUEDA DE UNA RED BAYESIANA

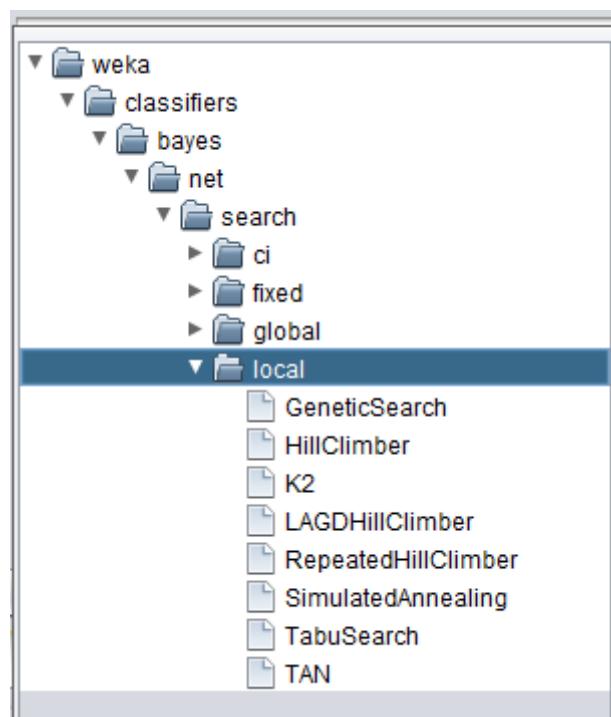


FIGURA N° 22: CONFIGURACIÓN DE UNA RED BAYESIANA CON BÚSQUEDA HILL CLIMBING

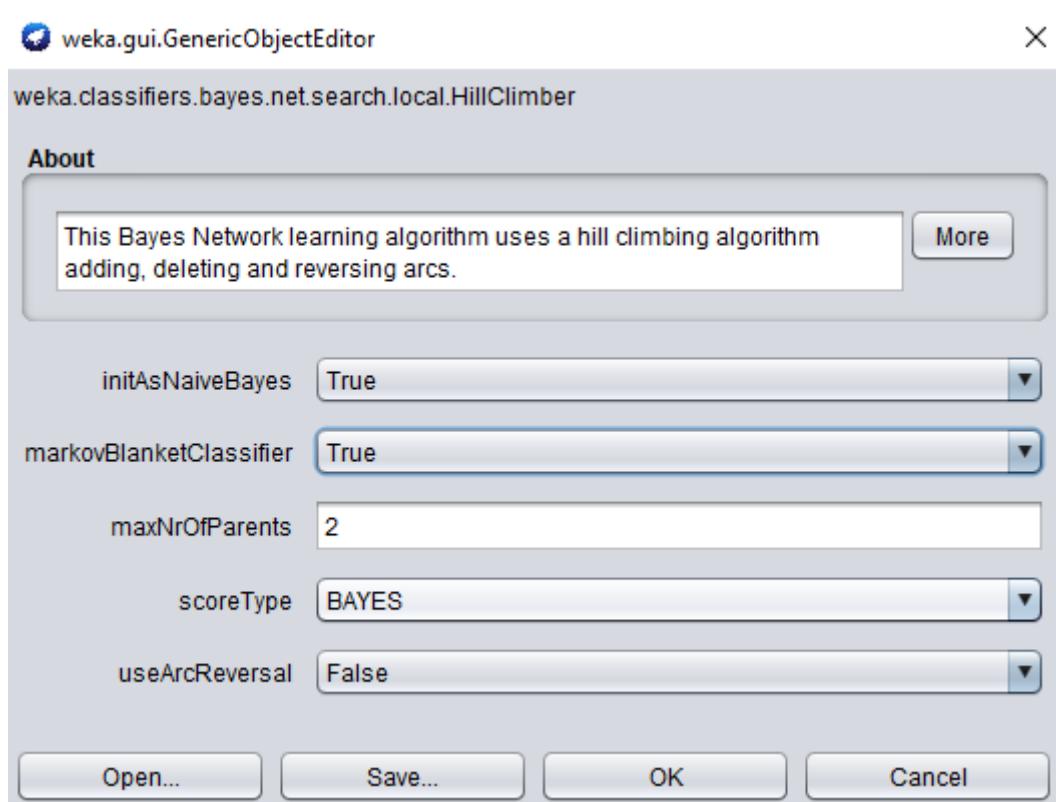
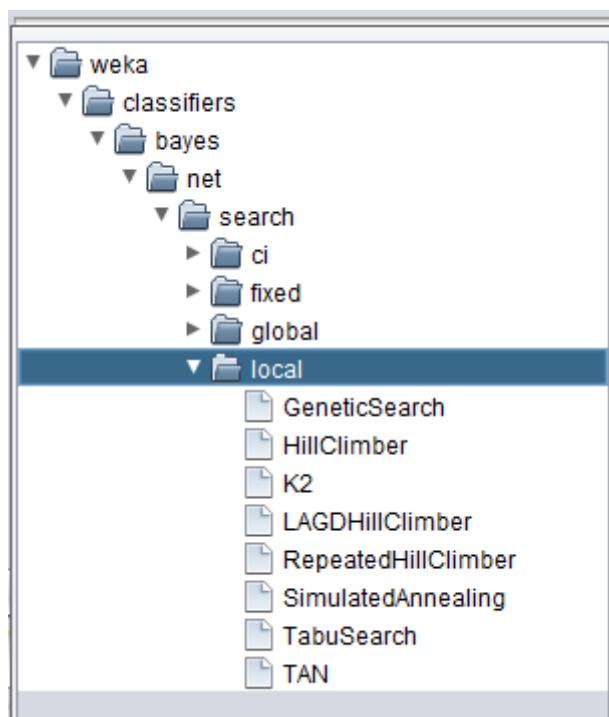


FIGURA Nº 23: MÉTODOS DE BÚSQUEDA PARA LAS REDES BAYESIANAS.



Los resultados que se obtiene con la configuración anterior son los siguientes:

LogScore Bayes: -14269.650656063288

LogScore BDeu: -15163.69208443938

LogScore MDL: -15061.670114806797

LogScore ENTROPY: -14084.222742830823

LogScore AIC: -14367.222742830823

Time taken to build model: 0.13 seconds

==== Stratified cross-validation ===

==== Summary ===

Correctly Classified Instances 740 74 %

Incorrectly Classified Instances 260 26 %

Kappa statistic 0.3408

Mean absolute error 0.3187

Root mean squared error 0.4231

Relative absolute error 75.8471 %

Root relative squared error 92.3198 %

Total Number of Instances 1000

== Detailed Accuracy By Class ==

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,860	0,540	0,788	0,860	0,822	0,345	0,765	0,880	good
0,460	0,140	0,585	0,460	0,515	0,345	0,765	0,533	bad
0,740	0,420	0,727	0,740	0,730	0,345	0,765	0,776	Weighted Avg.

== Confusion Matrix ==

a b -- classified as

602 98 | a = good

162 138 | b = bad

Weka dispone de un gráfico asociado a las redes bayesianas, así como el conjunto de tablas de probabilidades. La riqueza y expresividad de las redes bayesianas se muestra, aparte de en su estructura gráfica, en su tabla de probabilidad

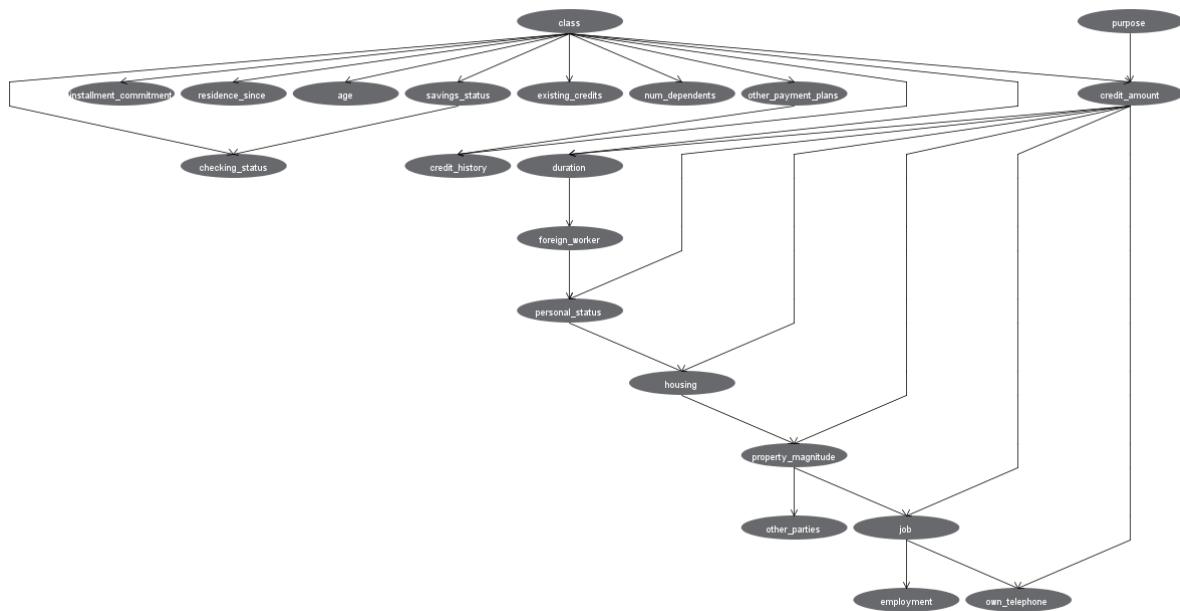
Cada algoritmo de búsqueda de los que se pueden utilizar genera una estructura de red bayesiana sobre la que se pueden calcular, para cada variable, las probabilidades condicionales a sus respectivos padres.

Se puede aprovechar la estructura de dependencias de la red bayesiana para reducir el número de variables utilizando el Manto de Markov.

Dado que en una red bayesiana se cumple que toda variable explicativa x es independiente del resto de variables explicativas dado su manto (envolvente) de Markov, se pueden seleccionar exclusivamente aquellas variables predictoras que realmente sean útiles en el modelo buscado. El Manto de Markov está formado por la unión de (padres (X) U hijos (X) U padres (hijos (X))).

Los gráficos de la red bayesiana con el algoritmo de búsqueda Hill Climber, con y sin Manto de Markov, con dos padres se ven en las figuras 24 y 25.

FIGURA N° 24: RED BAYESIANA HILL CLIMBER CON DOS PADRES



==== Detailed Accuracy By Class ====

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,859	0,590	0,772	0,859	0,813	0,296	0,741	0,867	good
0,410	0,141	0,554	0,410	0,471	0,296	0,741	0,534	bad
0,724	0,455	0,707	0,724	0,711	0,296	0,741	0,767	Weighted Avg

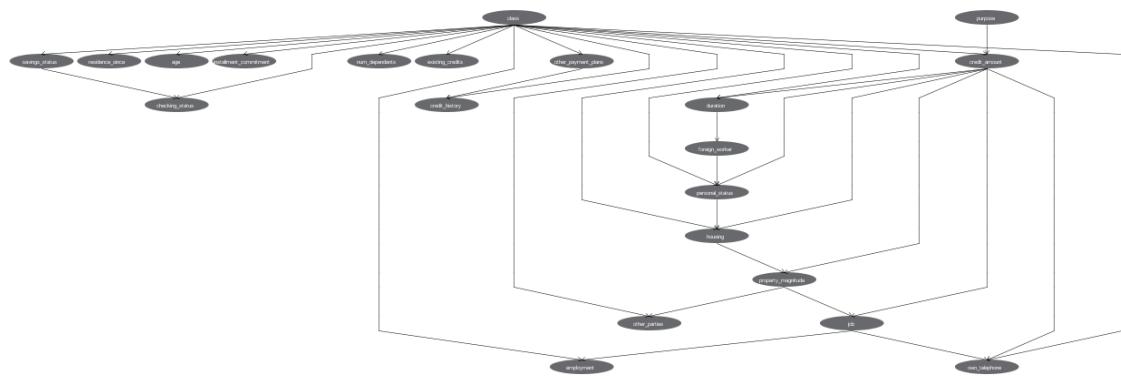
==== Confusion Matrix ====

a b <-- classified as

601 99 | a = good

177 123 | b = bad

FIGURA N° 25 RED BAYESIANA HILL CLIMBER CON DOS PADRES Y MANTO DE MARKOV.



== Detailed Accuracy By Class ==

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,860	0,540	0,788	0,860	0,822	0,345	0,765	0,880	good
0,460	0,140	0,585	0,460	0,515	0,345	0,765	0,533	bad
0,740	0,420	0,727	0,740	0,730	0,345	0,765	0,776	Weighted Avg.

== Confusion Matrix ==

a b <-- classified as

602 98 | a = good

162 138 | b = bad

Algunas de las tablas de probabilidad condicional asociadas a ese modelo son las siguientes:

Probability Distribution Table For savings_status

class	<100	100<=X<500	500<=X<1000	>=1000	'no known s...
good	0,55	0,099	0,075	0,06	0,216
bad	0,719	0,114	0,038	0,021	0,107

Probability Distribution Table For foreign_worker

duration	yes	no
'(-inf-15.5]"	0,929	0,071
0,987	0,013	

4.11. Redes bayesianas con R y Python

Redes Bayesianas en R.

En R podemos utilizar la librería bnclassify para obtener modelos de redes bayesianas. Los modelos de TAN, AODE y TAN Hill Climbing se pueden usar a través de las funciones que vienen en el paquete bnclassify, que puede verse en el enlace a su documentación: <https://cran.r-project.org/web/packages/bnclassify/index.html>

```
bnclassify: Learning Discrete Bayesian Network Classifiers from Data
State-of-the-art algorithms for learning discrete Bayesian network classifiers from data, including a number of those described in Bielza & Larrañaga (2014) <doi:10.1145/2576868>,
Version: 0.4.7
Depends: R (≥ 3.2.0)
Imports: assertthat (≥ 0.1), entropy (≥ 1.2.0), matrixStats (≥ 0.14.0), rpart (≥ 4.1-8), Rcpp
LinkingTo: Rcpp, BH
Suggests: graph (≥ 1.42.0), gRain (≥ 1.2-3), gRbase (≥ 1.7-0.1), mlr (≥ 2.2), testthat (≥ 0.8.1), knitr (≥ 1.10.5), ParamHelpers (≥ 1.5), Rgraphviz (≥ 2.8.1), rmarkdown (≥ 2022-11-16)
Published: 2022-11-16
Author: Mihaljevic Bojan [aut, cre], Bielza Concha [aut], Larrañaga Pedro [aut], Wickham Hadley [ctb] (some code extracted from memoise package)
Maintainer: Mihaljevic Bojan <boki.mihaljevic@gmail.com>
BugReports: https://github.com/bmihaljevic/bnclassify/issues
License: GPL-2 | GPL-3 [expanded from: GPL (≥ 2)]
URL: https://github.com/bmihaljevic/bnclassify
NeedsCompilation: yes
SystemRequirements: C++11
Citation: bnclassify\_citation\_info
Materials: README NEWS
In views: GraphicalModels
CRAN checks: bnclassify\_results

Documentation:
Reference manual: bnclassify.pdf
Vignettes: methods
overview
usage

Downloads:
Package source: bnclassify\_0.4.7.tar.gz
```

Para obtener ayuda de la librería podemos invocarla con ??bnclassify

Structure learning algorithms:

- [nb](#): Naive Bayes (Minsky, 1961)
- [tan_c1](#): Chow-Liu's algorithm for one-dependence estimators (CL-ODE) (Friedman et al., 1997)
- [fssj](#): Forward sequential selection and joining (FSSJ) (Pazzani, 1996)
- [bsej](#): Backward sequential elimination and joining (BSEJ) (Pazzani, 1996)
- [tan_hc](#): Hill-climbing tree augmented naive Bayes (TAN-HC) (Keogh and Pazzani, 2002)
- [tan_hcsp](#): Hill-climbing super-parent tree augmented naive Bayes (TAN-HCSP) (Keogh and Pazzani, 2002)
- [aode](#): Averaged one-dependence estimators (AODE) (Webb et al., 2005)

Parameter learning methods ([lp](#)):

- Bayesian and maximum likelihood estimation
- Weighting attributes to alleviate naive bayes' independence assumption (WANBIA) (Zaidi et al., 2013)
- Attribute-weighted naive Bayes (AWN) (Hall, 2007)
- Model averaged naive Bayes (MANB) (Dash and Cooper, 2002)

Model evaluating:

- [cv](#): Cross-validated estimate of accuracy
- [loglik](#): Log-likelihood
- [AIC](#): Akaike's information criterion (AIC)
- [BIC](#): Bayesian information criterion (BIC)

Predicting:

- [predict](#): Inference for complete and/or incomplete data (the latter through [gRain](#))

Inspecting models:

- [plot](#): Structure plotting (through [Rgraphviz](#))
- [print](#): Summary
- [params](#): Access conditional probability tables
- [nparams](#): Number of free parameters
- and more. See [inspect_bnc_dag](#) and [inspect_bnc_bn](#).

References

A continuación, ponemos algún ejemplo de diferentes formas de invocar a las funciones directamente, de forma que estas llamadas a las funciones se encargan de construir el modelo para luego poder hacer predicciones.

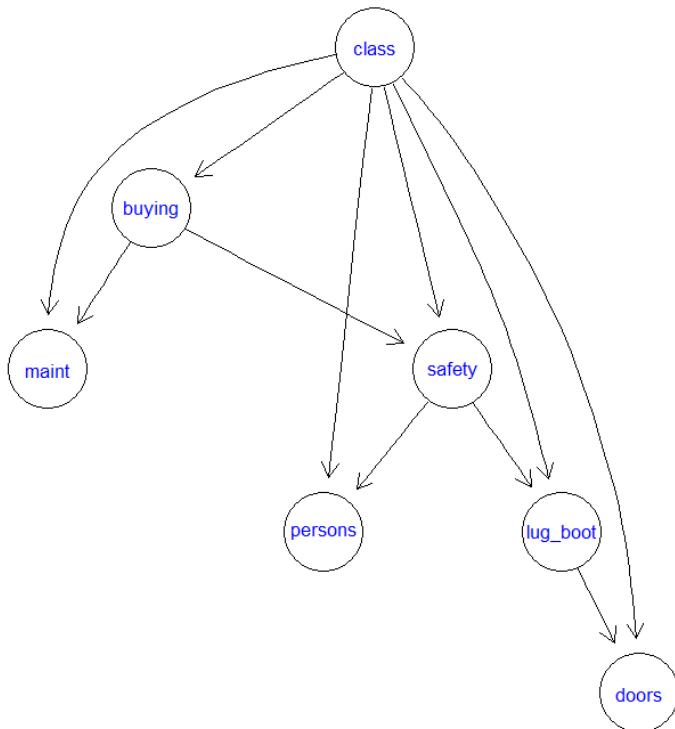
```
nb1 <- bnc('tan_cl', 'y', datosEntrenamientoBN, smooth = 25, dag_args = lis
t(score = 'aic'))
nb1 <- tan_cl('y', car, score = 'aic')

nb2 <- bnc('tan_hc', 'y', datosEntrenamientoBN, smooth = 25)
nb2 <- tan_hc('y', datosEntrenamientoBN, k = 5, epsilon = 0)

nb3 <- bnc('aode', 'y', datosEntrenamientoBN, smooth = 25)
nb3 <- aode('y', datosEntrenamientoBN, k = 5, epsilon = 0)
```

Con el fichero car obtenemos una red bayesiana con los siguientes comandos.

```
library(bnclassify)
data(car)
nb <- tan_cl('class', car)
nb
plot(nb)
```



```
nb <- bnc('nb', 'class', car, smooth = 1)
```

```
logLik(nb, car)
```

```
log Lik. -13518.21 (df=63)
```

```
AIC(nb, car)
```

```
-13581.21
```

```
BIC(nb, car)
```

```
-13753.03
```

Existen múltiples programas para la estimación de estructuras de redes bayesianas en R. El trabajo de fin de grado de Angulo Montes (2020) que se puede bajar de la siguiente dirección de internet: http://cig.fi.upm.es/wp-content/uploads/2022/10/TFM_LUIS_EDUARDO_ANGULO_MONTES.pdf analiza nueve paquetes disponibles de redes bayesianas en R y presenta un conjunto de tablas de características de los programas para que podamos hacernos una idea precisa de lo que nos conviene para el estudio de las redes bayesianas. Las nueve librerías de R que describe y analiza son las siguientes: catnet, sparsebn, pcalg, deal, bnclassify, bnstruct, bnlearn, gRain y BayesNetBP.

Para poder aplicar las redes bayesianas es necesario discretizar las variables para que se puedan usar con estas funciones. A continuación, se muestra un ejemplo de como realizar este trabajo con un determinado dataframe.

```
# Creamos la función que se encarga de discretizar
# Se le pasa como parámetro el vector asociado a una variable, y los tres cuantiles
# Nos devuelve el vector discretizado usando los cuantiles

convierteDiscreto <- function(x, cuantil1, cuantil2, cuantil3) {
  if (x<= cuantil1) {return('Tipo 1')}
  else
    if( x> cuantil1 && x <= cuantil2 ) {return('Tipo 2')}
  else
    if (x> cuantil2 && x < cuantil3) {return('Tipo 3')}
  else
    {return('Tipo 4')}
}

# Definimos cuales son las variables numéricas
variablesNumericas <- c("age","duration","campaign","previous","cons.price.idx","cons.conf.idx","nr.employed")
discretos <- NULL

# Recorremos todas las variables numéricas de forma que
# calculamos sus cuantiles e invocamos a nuestra función de discretizar
for( i in 1:length(variablesNumericas)) {

  cuantil1 <- quantile(dbAFFinal[,variablesNumericas[i]],0.25)
  cuantil2 <- mean(dbAFFinal[,variablesNumericas[i]])
  cuantil3 <- quantile(dbAFFinal[,variablesNumericas[i]],0.75)

  if( is.null(discretos)){
    # La primera vez creamos el data frame sin nada anterior
    discretos <- as.data.frame(as.factor(sapply(dbAFFinal[,variablesNumericas[i]],FUN=convierteDiscreto,cuantil1=cuantil1,cuantil2=cuantil2,cuantil3=cuantil3)))
    print(str(discretos))
  }
  else{
    # Ahora ya añadimos a lo que hay en discretos, lo nuevo
    discretos <- as.data.frame(cbind( discretos, as.factor(sapply(dbAFFinal[,variablesNumericas[i]],FUN=convierteDiscreto,cuantil1=cuantil1,cuantil2=cuantil2,cuantil3=cuantil3))))
  }
}

discretos <- as.data.frame(discretos)
str(discretos)
colnames(discretos) <- variablesNumericas

# Finalmente juntamos las variables discretas con el resto
dbaAFFinalDiscretos <- data.frame(discretos, dbAFFinal[,c(2,3,4,5,6,7,8,9,13,17)])
str(dbaAFFinalDiscretos)
```

Funcionamiento del modelo TAN en caret

A la hora de invocar a la función train de caret, normalmente pasamos como parámetro el data frame de los datos data y la fórmula $y \sim$. indicando que la variable y es la variable explicada.

La implementación de TAN en caret parece que no es correcta y no recoge el parámetro de la fórmula.

Lo que tenemos que hacer es invocar a la función train pasándole como parámetros el dataframe de las explicativas y como segundo parámetro el vector que es la explicada. De esta manera la función TAN recogerá correctamente los parámetros y funcionará perfectamente.

Veamos a continuación como le pasamos al método train los dos parámetros en lugar de pasar la fórmula

```
train(datosEntrenamientoBN,
      datosEntrenamientoBN$y,
      method="tan",
      metric="ROC",
      trControl=trainControl(method = "repeatedcv", number = 5, repeats = 2
, verboseIter=FALSE, classProbs = TRUE, summaryFunction=estadisticasExtra))
```

Creación de nuevos modelos en caret

Para poder solventar el problema de no tener los modelos de **AODE** y **Hill Climbing**, sabiendo que existen funciones que realizan este trabajo, se puede construir un modelo de caret para de esta manera poder usarlo y así usar toda la parte de **Evaluación de Modelos** de caret y hacer una buena comparativa con el reto de algoritmos.

La forma de construir nuevos modelos en caret está documentada en <http://topepo.github.io/caret/using-your-own-model-in-train.html> donde se explica cómo construir nuestros propios modelos.

Los elementos principales son:

- Parámetros que va a haber (los que se usarán con el expand.grid) + Función de ajuste (fit) que es la que crea el modelo y tiene que invocar a la función del paquete bnclassify.
- Función de predicción (predict) que es la que invoca caret para las predicciones y tiene que invocar a la de predicción propia en el paquete bnclassify.

Modelo Hill Climbing Tree Augmented Naive Bayes Classifier

```
AlgTANHC <- list(label = "Hill Climbing Tree Augmented Naive Bayes Classifier",
                  library = "bnclassify",
                  type = "Classification",

                  parameters = data.frame(parameter = c('epsilon', "smooth"),
                                           class = c("numeric", "numeric"),
                                           label = c('Epsilon Parameter', "Smoothi
ng Parameter")),

                  grid = function(x, y, len = NULL, search = "grid") {
                    if(search == "grid") {
                      out <- expand.grid(epsilon = c(0.01, 0.02),
```

```

                    smooth = 0:2)
}
else {
  out <- data.frame(smooth= 0:2,
                     epsilon = sample(c(0.01, 0.02),
                     size = len,
                     replace = TRUE))
}
out
},
loop = NULL,
fit = function(x, y, wts, param, lev, last, classProbs, ...) {
  dat <- if(is.data.frame(x)) x else as.data.frame(x)
  dat$outcome <- y

  #print("fit tan_hc")
  bnclassify::bnc('tan_hc', class = '.outcome', dataset = dat,
smooth = param$smooth,dag_args = list(k = 2,param$epsilon ), ...)
},
# La función predict realmente se usará la de bnclassify ya que
el modelFit
# es un objeto de su clase y se invocará su predict
predict = function(modelFit, newdata, submodels = NULL) {
  if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata)
  predict(modelFit, newdata)
},
prob = function(modelFit, newdata, submodels = NULL) {
  if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata)
  predict(modelFit, newdata, prob = TRUE) },
levels = function(x) x$obsLevels,
predictors = function(x, s = NULL, ...) x$xNames,
tags = c("Bayesian Model TAN HC", "Categorical Predictors Only"
),
sort = function(x) x[order(x[,1]),])

```

Modelo AODE

```

AODE <- list(label = "AODE Naive Bayes Classifier",
              library = "bnclassify",
              type = "Classification",
              parameters = data.frame(parameter = c("smooth"),
                                         class = c("numeric"),
                                         label = c("Smoothing
Parameter"))),
              grid = function(x, y, len = NULL, search = "grid") {
                if(search == "grid") {
                  out <- expand.grid(smooth = 0:(len-1))
                }
                else {
                  out <- data.frame(smooth= runif(len, min = 0, max =
10),
                                 size = len,

```

```

                    replace = TRUE)
    }
    out
},
loop = NULL,

# En la función tan_hc colocamos manualmente como primer
parámetro el nombre de la variable explicada
fit = function(x, y, wts, param, lev, last, classProbs,
...) {
    dat <- if(is.data.frame(x)) x else as.data.frame(x)
    dat$outcome <- y
    bnclassify::bnc('aode','y', class = '.outcome',
dataset = dat, smooth = param$smooth)
},
# La función predict realmente se usará la de bnclassify
ya que el modlFit
# es un objeto de su clase y se invocará su predict
predict = function(modelFit, newdata, submodels = NULL)
{
    if(!is.data.frame(newdata))
        newdata <- as.data.frame(newdata)
    predict(modelFit, newdata) ,

prob = function(modelFit, newdata, submodels = NULL) {
    if(!is.data.frame(newdata)) newdata <-
as.data.frame(newdata)
    predict(modelFit, newdata, prob = TRUE)
},
levels = function(x) x$obsLevels,
predictors = function(x, s = NULL, ...) x$xNames,
tags = c("Bayesian Model AODE", "Categorical Predictors
Only"),
sort = function(x) x[order(x[,1]),])

```

Uso de los modelos creados en caret

Una vez que hemos construido nuestros modelos, la forma de invocarlos es usando en el parámetro `method` el nombre de la variable lista que hemos creado con nuestro modelo, en lugar de poner una cadena de texto como hacemos con los modelos que tiene caret.

Hay que tener en cuenta que tal y como están construidos estos modelos, la variable explicativa tiene que llamarse "y". Si es necesario se cambia en el dataframe o se puede cambiar el modelo con el nombre que se quiera.

A continuación, vemos como se invocaría a los dos modelos.

```
train(datosEntrenamientoBN,
      datosEntrenamientoBN$y,
      method = AlgTANHC,
      metric = "ROC",
      trControl=trainControl(method = "repeatedcv", number = 5, repeats = 2
, verboseIter=FALSE, classProbs = TRUE, summaryFunction=estadisticasExtra))
```

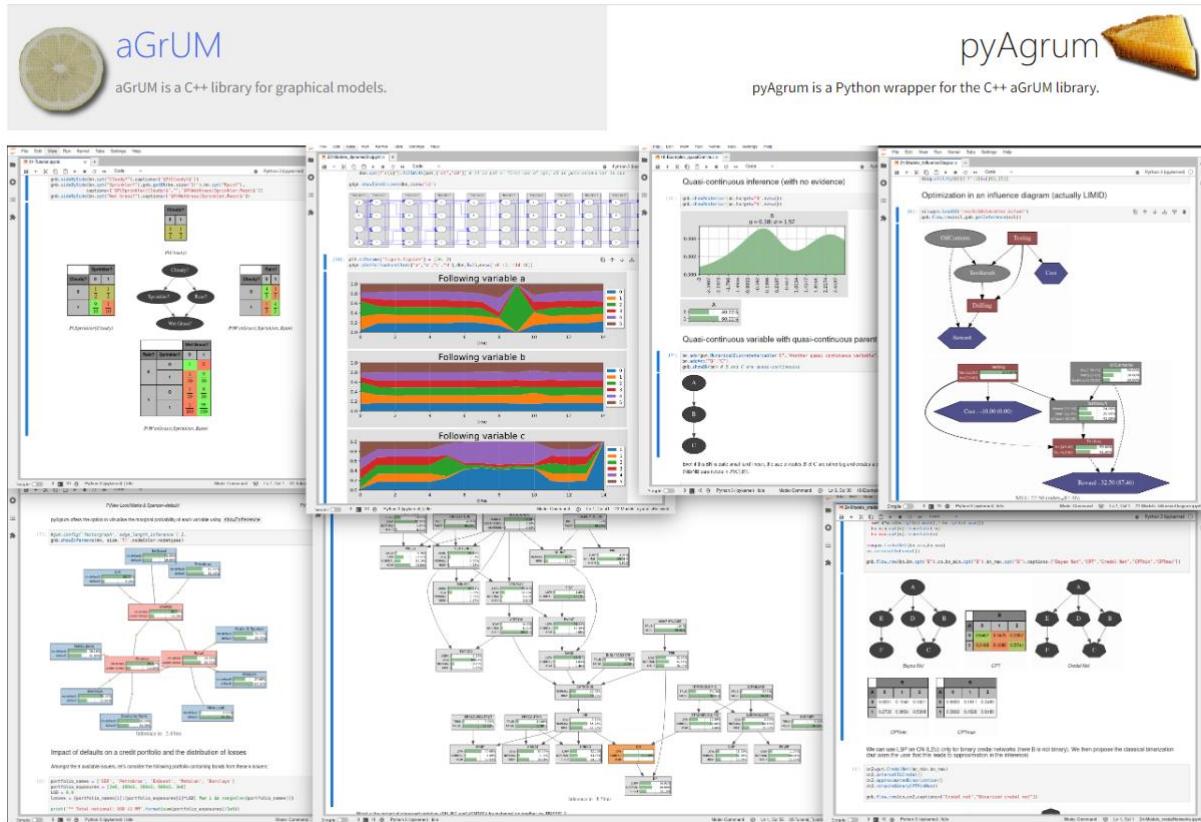
```
train(datosEntrenamientoBN,
      datosEntrenamientoBN$y,
      method = AODE,
      metric="ROC",
      trControl=trainControl(method = "repeatedcv", number = 5, repeats = 2
, verboseIter=FALSE, classProbs = TRUE, summaryFunction=estadisticasExtra))
```

Redes bayesianas con Python

Para realizar redes bayesianas en python disponemos de la librería scikit learn y también de la biblioteca científica pyAgrum (<http://agrum.org>) que es un contenedor de Python para la biblioteca aGrUM de C++. Proporciona una interfaz de alto nivel a la parte de aGrUM que permite crear, modelar, aprender, usar, calcular e integrar redes bayesianas y otros modelos gráficos probabilísticos como las redes de Markov o los modelos relacionales probabilísticos. La imagen siguiente es la página principal del proyecto pyAgrum (<https://agrum.gitlab.io/pages/pyagrum.html>) donde se encuentran los diversos modelos que implementa esta librería.

pyAgrum

A Graphical Universal Modeler (<https://gitlab.com/agrumery/aGrUM>)



Se han agregado varios complementos a pyAgrum (como módulos de python puros que usan pyAgrum) lo que nos va a permitir realizar algunas operaciones muy importantes:

- Aplicar clasificadores probabilísticos compatibles con Scikit-learn basados en redes bayesianas.
- Experimentar con Modelos de Causalidad probabilística (redes causales)
- Trabajar con Redes Bayesiana dinámicas.
- Incorporar herramientas para la explicabilidad de las redes bayesianas.

En esta pagina web existen muchos ejemplos de la utilización de esta biblioteca, así como una documentación excelente que se puede descargar. A continuación, se muestran algunos comandos con el fichero german credit.

Importamos las librerías

```
import pyAgrum.skbn as skbn
import pyAgrum.lib.notebook as gnb
import os
import pandas as pd
```

Cargamos la base de datos de german credit.

```
datos = pd.read_csv('G:/1_Master_2023/MODULO_7/PYAGRUM/credit_g.csv')
```

```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   checking_status  1000 non-null    object 
 1   duration         1000 non-null    int64  
 2   credit_history   1000 non-null    object 
 3   purpose          1000 non-null    object 
 4   credit_amount    1000 non-null    int64  
 5   savings_status   1000 non-null    object 
 6   employment       1000 non-null    object 
 7   installment_commitment 1000 non-null  int64  
 8   personal_status  1000 non-null    object 
 9   other_parties    1000 non-null    object 
 10  residence_since 1000 non-null    int64  
 11  property_magnitude 1000 non-null  object 
 12  age              1000 non-null    int64  
 13  other_payment_plans 1000 non-null  object 
 14  housing          1000 non-null    object 
 15  existing_credits 1000 non-null    int64  
 16  job              1000 non-null    object 
 17  num_dependents  1000 non-null    int64  
 18  own_telephone    1000 non-null    object 
 19  foreign_worker   1000 non-null    object 
 20  class            1000 non-null    object 
dtypes: int64(7), object(14)
memory usage: 164.2+ KB
```

Primera red bayesiana. Especificamos diferentes parámetros para la red bayesiana.

```
BNTest= skbn.BNClassifier(learningMethod = 'Chow-Liu', aPriori= 'Smoothing',
aPrioriWeight = 0.5, discretizationStrategy = 'quantile', usePR = True, significant_digit = 13)
```

Ajuste del modelo

```
BNTest.fit(filename = 'G:/1_Master_2022/MODULO_7_2022/PYAGRUM/credit_g.csv',
targetName = 'class')
```

Generación de los intervalos creados

```
for i in BNTest.bn.nodes():
print(BNTest
.bn.variable(i))
```

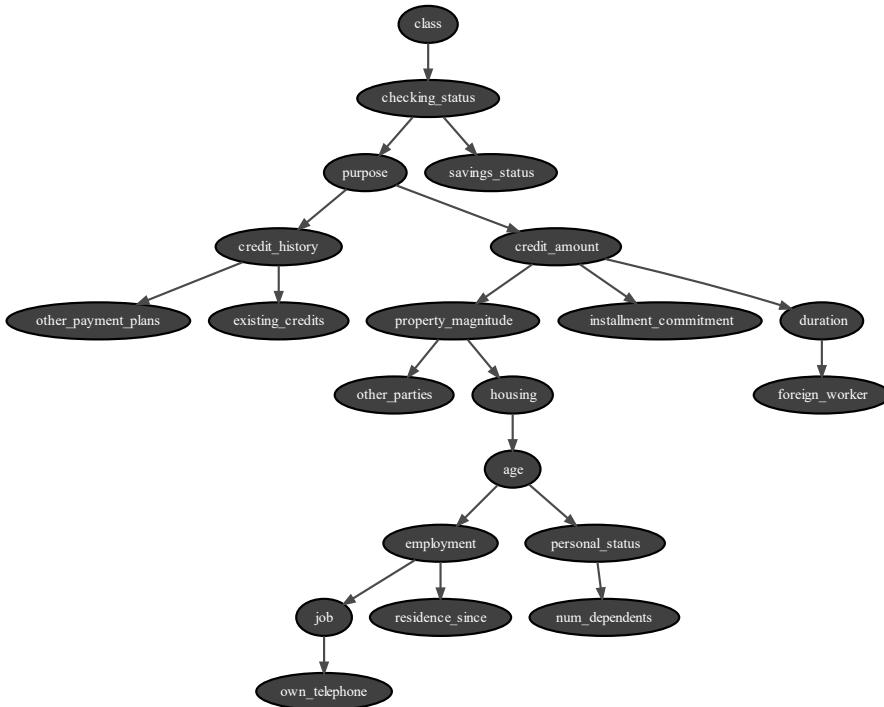
```

class:Labelized({bad|good})
checking_status:Labelized({'no checking'|0<=X<200|<0|>=200})
duration:Discretized(<(4;12],[12;15],[15;24],[24;30],[30;72>)
credit_history:Labelized({'all paid'|'critical/other existing credit'|'delayed previously'|'existing paid'|'no credits/all paid'})
purpose:Labelized({'domestic appliance'|'new car'|'used car'|business|education|furniture/equipment|other|radio/tv|repairs|retraining})
credit_amount:Discretized(<(250;1262],[1262;1906.8],[1906.8;2852.4],[2852.4;4720],[4720;18424>)
savings_status:Labelized({'no known savings'|100<=X<500|500<=X<1000|<100|>=1000})
employment:Labelized({1<=X<4|4<=X<7|<1|>=7|unemployed})
installment_commitment:Range([1,4])
personal_status:Labelized({'female div/dep/mar'|'male div/sep'|'male mar/wid'|'male single'})
other_parties:Labelized({'co applicant'|guarantor|none})
residence_since:Range([1,4])
property_magnitude:Labelized({'life insurance'|'no known property'|'real estate'|car})
age:Discretized(<(19;26],[26;30],[30;36],[36;45],[45;75])
other_payment_plans:Labelized({bank|none|stores})
housing:Labelized({for free|own|rent})
existing_credits:Range([1,4])
job:Labelized({'high qualif/self emp/mgmt'|'unemp/unskilled non res'|'unskilled resident'|skilled})
num_dependents:Range([1,2])
own_telephone:Labelized({none|yes})
foreign_worker:Labelized({no|yes})

```

Visualización de la estructura de la red bayesiana

gnb.showBN(BNTest.bn)



```

xTrain, yTrain = BNTest.XYfromCSV(filename =
'G:/1_Master_2023/MODULO_7/PYAGRUM/credit_g.csv', target = 'class')

```

```

scoreCSV2 = BNTest2.score('G:/1_Master_2023/MODULO_7/PYAGRUM/credit_g.csv', y = yTrain)
print("{0:.2f}% de acierto de la predicción".format(100*scoreCSV2))

```

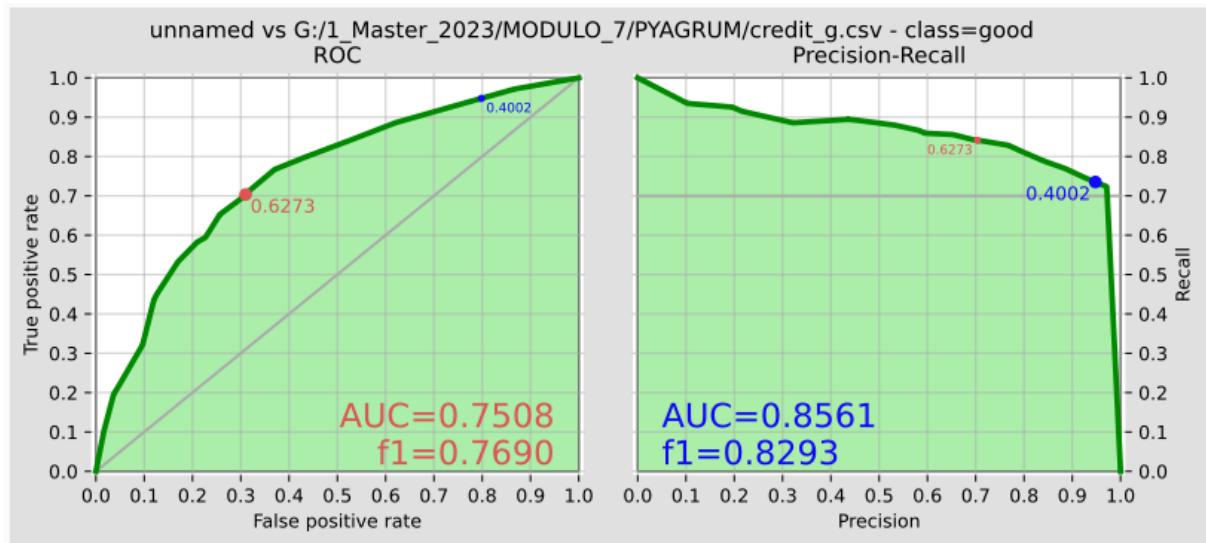
```

print("{0:.2f}% de acierto de la predicción".format(100*scoreCSV1))

```

80.70% de acierto de la predicción

Visualización de la Curva ROC



Segunda red bayesiana. Especificamos diferentes parámetros de la red bayesiana. Respecto a la red anterior sólo se modifica el contenido de learningMethod.

```
BNTest2= skbn.BNClassifier(learningMethod = 'MIIC', aPriori= 'Smoothing', aPrioriWeight = 0.5,
discretizationStrategy = 'quantile', usePR = True, significant_digit = 13)
```

```
xTrain, yTrain = BNTest2.XYfromCSV(filename =
'G:/1_Master_2023/MODULO_7/PYAGRUM/credit_g.csv', target = 'class')
```

```
BNTest2.fit(xTrain, yTrain)
```

```
gnb.showBN(BNTest2.bn)
```



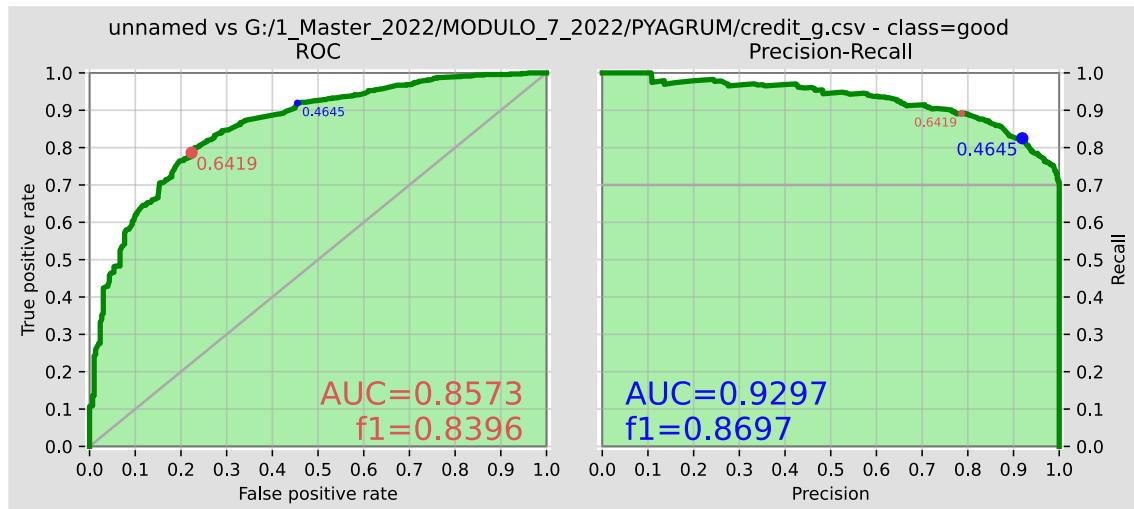
```

scoreCSV2 = BNTest2.score('G:/1_Master_2022/MODULO_7_2022/PYAGRUM/credit_g.csv', y = yTrain)
print("{0:.2f}% de acierto de la predicción".format(100*scoreCSV2))

```

80.70% de acierto de la predicción

```
BNTest.showROC_PR('G:/1_Master_2022/MODULO_7_2022/PYAGRUM/credit_g.csv')
```



Las diferentes especificaciones del algoritmo se pueden encontrar en la siguiente dirección:

<https://pyagrum.readthedocs.io/en/0.19.2/skbnClassifier.html>

Por ejemplo, la clase pyAgrum.skbn.BNClassifier dispone de los siguientes parámetros:

```

class pyAgrum.skbn.BNClassifier(learningMethod='GHC', aPriori=None, scoringType='BIC',
constraints=None, aPrioriWeight=1, possibleSkeleton=None, DirichletCsv=None,
discretizationStrategy='quantile', discretizationNbBins=5, discretizationThreshold=25,
usePR=False, significant_digit=10)

```

En el método **learningMethod** se pueden especificar los siguientes valores: Chow-Liu, NaiveBayes, TAN, MIIC + (MDL o NML), GHC, 3off2 + (MDL o NML), Tabu. GHC es el método Greedy Hill Climbing, MIIC son las iniciales de Multivariate Information Based Inductive Causation, TAN significa Tree-augmented NaiveBayes y Tabu designa a Tabu list searching.

Al sustituir el valor de learningMethod por GHC el porcentaje de acierto es del 72% mientras que si lo cambiamos por NaiveBayes alcanza el 77,6% de precisión.

La biblioteca de Python permite emplear diferentes métodos para discretizar las variables. Los valores posibles son: 'quantil', 'uniforme', 'kmeans', 'NML', 'CAIM' y 'MDLP'. CAIM son las iniciales de Class-Attribute Interdependency Maximization. El método de discretización MDLP fue propuesto por Fayyad e Irani (1993) y utiliza la medida de información mutua para definir de

forma recursiva los mejores intervalos. Si se emplea el método NML, este parámetro establece el número máximo de intervalos hasta que el algoritmo NML busca el número óptimo.

Naïve Bayes con scikit- Learn

A continuación, se presenta el código de un ejemplo del clasificador Naïve Bayes realizado con la biblioteca scikit-learn de Python.

```
# Cargamos los datos de trabajo
```

```
os.chdir('D:\\MASTER_2020\\MODULO_5_2020') #directorio  
datos=pd.read_csv('credit_g.csv',encoding = 'ISO-8859-1', index_col=None)
```

```
#Todas las variables son categóricas salvo:
```

```
# duration  
# credit_amount  
# residence_since  
# age  
# existing_credits  
# num_dependents  
# conversión a variables categóricas
```

```
datos['checking_status']=datos['checking_status'].astype('category')  
datos['credit_history']=datos['credit_history'].astype('category')  
datos['purpose']=datos['purpose'].astype('category')  
datos['savings_status']=datos['savings_status'].astype('category')  
datos['employment']=datos['employment'].astype('category')  
datos['personal_status']=datos['personal_status'].astype('category')  
datos['other_parties']=datos['other_parties'].astype('category')  
datos['property_magnitude']=datos['property_magnitude'].astype('category')  
datos['other_payment_plans']=datos['other_payment_plans'].astype('category')  
datos['housing']=datos['housing'].astype('category')  
datos['job']=datos['job'].astype('category')  
datos['property_magnitude']=datos['property_magnitude'].astype('category')  
datos['own_telephone']=datos['own_telephone'].astype('category')  
datos['foreign_worker']=datos['foreign_worker'].astype('category')  
datos['class']=datos['class'].astype('category')
```

```
# La variable class es una variable reservada en diferentes módulos de Python -> reemplazar por  
por target
```

```
datos.rename(columns={'class': 'target'}, inplace=True)  
datos['target']=np.where(datos['target']=='good', 0, 1) # cambio en la codificación por sencillez en  
el preprocesado
```

```
# Definición de la muestra de trabajo
```

```
datos_entrada=datos.drop('target', axis=1) # Datos de entrada  
datos_entrada= pd.get_dummies(datos_entrada, drop_first=True) #conversión a variables  
dummy
```

```
# Datos de salida
```

```
respuesta=datos.loc[:, 'target']
```

```
# Escalado de las variables, partición de la muestra y Cross Validation
```

```
# Escalado de los datos de entrada
```

```
entrada_esc=StandardScaler().fit_transform(datos_entrada)  
entrada_esc=pd.DataFrame(entrada_esc, columns=datos_entrada.columns)
```

```
# Partición de la muestra
```

```
test_size=0.3 #muestra para el test  
seed=222 #semilla  
x_train, x_test, y_train, y_test = train_test_split(entrada_esc, respuesta, test_size=test_size,  
random_state=seed, stratify=respuesta)
```

```
# cross_validation
```

```
cv=RepeatedStratifiedKFold(n_splits=10, n_repeats=2, random_state=seed)
```

Bernoulli Naive Bayes

```
from sklearn.naive_bayes import BernoulliNB
```

```

bernoulli_nb=BernoulliNB()
grid=[{'alpha': list(np.arange(0,1,0.1)), 'binarize': [0.3, 0.1, 0.0]}]

```

Definición del modelo con hiperparámetros

```

gs_bernoulli_nb=GridSearchCV(estimator=bernoulli_nb, param_grid=grid, scoring='accuracy',
cv=cv, n_jobs=1, return_train_score=False)
gs_bernoulli_nb=gs_bernoulli_nb.fit(x_train, y_train)
print('Naive-Bayes (Bernoulli) (parámetros):', gs_bernoulli_nb.best_params_) #parámetros del
modelo final
bernoulli_nb=gs_bernoulli_nb.best_estimator_ #modelo final

```

Resultados importantes de estos algoritmos (acceso dentro del objeto del modelo)

```

print(bernoulli_nb.class_log_prior_) # logaritmo de la probabilidad de cada clase
bernoulli_nb.feature_log_prob_ # logaritmo de la probabilidad de la variable dada la clase ( $P(X_i|Y)$ )

```

```

Naive-Bayes (Bernoulli) (parámetros): {'alpha': 0.1, 'binarize': 0.3}
[-0.35667494 -1.2039728]

```

```

array([[-1.66113929, -1.55911408, -0.8098598 , -0.85200086, -1.08222345,
       -0.97933438, -1.84971019, -1.45773821, -1.62942416, -2.63661234,
      -1.07622104, -2.50343714, -0.64146636, -3.88227805, -1.62942416,
      -2.06606748, -2.40835369, -3.01260163, -1.72775659, -4.7838265 ,
      -1.19692575, -3.98653906, -4.10294941, -2.55459919, -2.8240753 ,
      -0.59602121, -2.75760565, -1.62942416, -1.93072613, -1.28879823,
      -2.85904389, -3.48011873, -2.26102897, -0.58132047, -2.7902883 ,
      -0.08979705, -2.15000935, -1.1507434 , -1.12528054, -0.16388786,
      -3.29890153, -0.28917918, -1.81153762, -3.78786836, -1.5494615 ,
      -0.46764004, -0.8762034 , -8.49739856],
      [-1.09813666, -1.07000541, -0.66500273, -0.83610165, -1.26882853,
      -1.12708225, -1.78985833, -1.01601119, -0.84713929, -3.13978504,
      -1.87920343, -2.50898099, -0.59360757, -2.25248185, -1.26882853,
      -2.56924019, -2.20822684, -2.50898099, -1.56159968, -4.21665735,
      -1.60801172, -3.93707248, -7.65064455, -2.12519161, -3.13978504,
      -0.3706363 , -3.93707248, -2.20822684, -1.47477728, -1.60801172,
      -2.34733964, -2.70188466, -2.63336471, -0.71236007, -3.2561954 ,
      -0.09526261, -1.49578646, -1.68193699, -1.07000541, -0.33011759,
      -2.63336471, -0.53506243, -1.37588253, -3.93707248, -1.60801172,
      -0.47975607, -0.9900694 , -7.65064455]])

```

Predicciones muestra entrenamiento y test

```

preds_train = bernoulli_nb.predict(x_train)

```

```

preds_test = bernoulli_nb.predict(x_test)

# Cálculo métricas bondad de ajuste

print('Accuracy')
print('-----')
print('Entrenamiento (cv):', round(gs_bernoulli_nb.best_score_,5))
accuracy_test=accuracy_score(y_test, preds_test)
print('Test:', round(accuracy_test,5))

```

```
# AUC - test y curva roc (final)
```

```

y_pred_test = bernoulli_nb.predict_proba(x_test)
fp_rate_test, tp_rate_test, thresholds = roc_curve(y_test, y_pred_test[:,1])
auc_test=auc(fp_rate_test, tp_rate_test)

```

```

Accuracy
-----
Entrenamiento (cv): 0.75143
Test: 0.72333

```

```
# Bondad de ajuste: matriz de confusión y curva roc para los datos de test
```

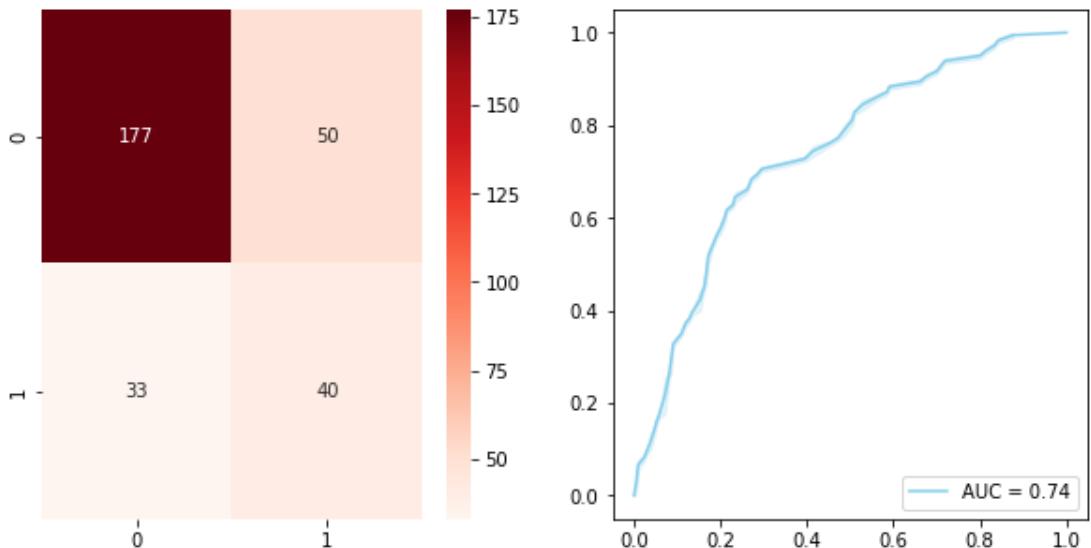
```

f, axes = plt.subplots(1, 2, figsize=(10,5))

sns.heatmap(confusion_matrix(preds_test, y_test), annot=True, cmap=plt.cm.Reds, fmt='0f',
ax=axes[0]) # matriz de confusión
sns.lineplot(fp_rate_test, tp_rate_test, color='skyblue', label='AUC = %0.2f' % auc_test,
ax=axes[1]) # curva roc

plt.legend(loc="lower right")
plt.show()

```



Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gaussian_nb=GaussianNB()
grid=[{'var_smoothing': list(np.arange(0,0.1, 0.02))}]
# Definición del modelo con hiperparámetros
gs_gaussian_nb=GridSearchCV(estimator=gaussian_nb, param_grid=grid, scoring='accuracy',
cv=cv, n_jobs=1, return_train_score=False)
gs_gaussian_nb=gs_gaussian_nb.fit(x_train, y_train)
print('Naive-Bayes (Bernoulli) (parámetros):', gs_gaussian_nb.best_params_)
```

```
#parámetros del modelo final
```

```
gaussian_nb=gs_gaussian_nb.best_estimator_ #modelo final
```

```
# Resultados importantes de estos algoritmos (acceso dentro del objeto del modelo)
```

```
gaussian_nb.class_prior_ # probabilidad de cada clase
```

```
Naive-Bayes (Bernoulli) (parámetros): {'var_smoothing': 0.08}
array([0.7, 0.3])
```

```
# predicciones muestra entrenamiento y test
```

```
preds_train = gaussian_nb.predict(x_train)
preds_test = gaussian_nb.predict(x_test)
```

```

# Cálculo métricas bondad de ajuste

print('Accuracy')
print('-----')
print('Entrenamiento (cv):', round(gs_gaussian_nb.best_score_,5))
accuracy_test=accuracy_score(y_test, preds_test)
print('Test:', round(accuracy_test,5))

#AUC - test y curva roc (final)

y_pred_test = gaussian_nb.predict_proba(x_test)
fp_rate_test, tp_rate_test, thresholds = roc_curve(y_test, y_pred_test[:,1])
auc_test=auc(fp_rate_test, tp_rate_test)

Accuracy
-----
Entrenamiento (cv): 0.67286
Test: 0.66

```

```

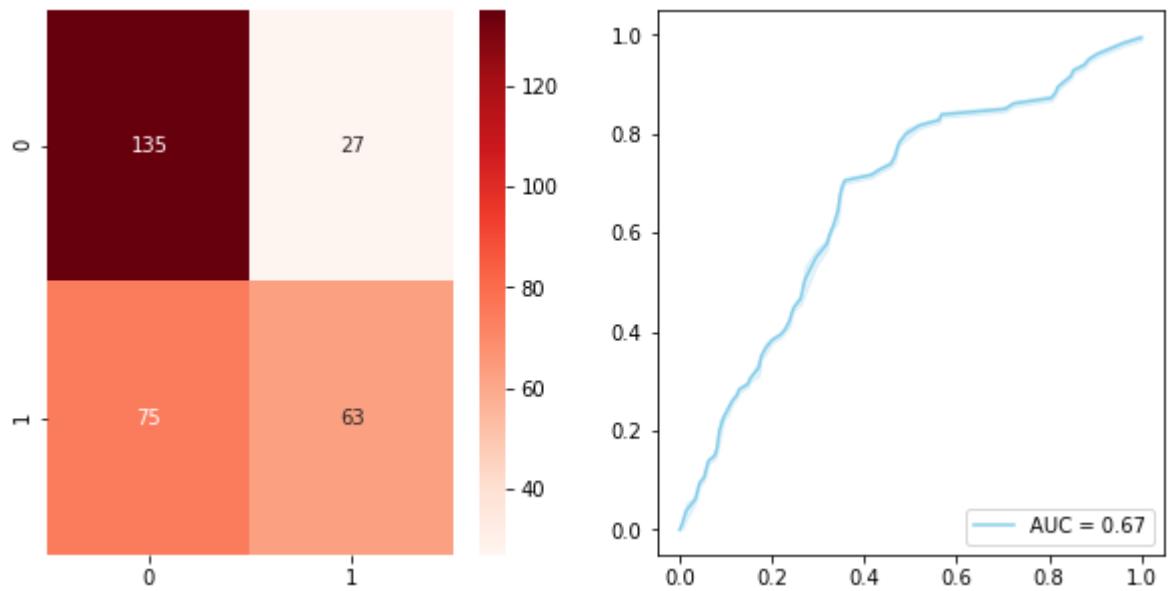
# Bondad de ajuste: matriz de confusión y curva roc para los datos de test

f, axes = plt.subplots(1, 2, figsize=(10,5))

sns.heatmap(confusion_matrix(preds_test, y_test), annot = True, cmap = plt.cm.Reds, fmt='%.0f', ax=axes[0]) # matriz de confusión
sns.lineplot(fp_rate_test, tp_rate_test, color='skyblue', label='AUC = %.2f' % auc_test, ax=axes[1]) # curva roc

plt.legend(loc="lower right")
plt.show()

```



5. Algoritmos evolutivos

5.1. Introducción

Los algoritmos evolutivos son uno de los métodos más comunes en minería de datos.

Se inspiran en el proceso natural de selección y evolución tal y como se describe por la teoría evolucionista de la selección natural postulada por Darwin. Los principios sobre los que se asientan los algoritmos evolutivos y, en concreto, los genéticos son:

- Los individuos mejor adaptados al entorno son aquellos que tienen una probabilidad mayor de sobrevivir y, por ende, de reproducirse.
- Los descendientes heredan características de sus progenitores.
- De forma esporádica y natural se producen mutaciones en el material genético de algunos individuos, provocando cambios permanentes.

Los algoritmos genéticos son adecuados para obtener buenas aproximaciones en problemas de búsqueda, aprendizaje y optimización [Marczyk. 2004].

De forma esquemática un algoritmo genético es una función matemática que tomando como entrada unos individuos iniciales (población origen) selecciona aquellos ejemplares (también llamados genes) que recombinándose por algún método generarán como resultado la siguiente generación. Esta función se aplicará de forma iterativa hasta verificar alguna condición de parada, bien pueda ser un número máximo de iteraciones o bien la obtención de un individuo que cumpla unas restricciones iniciales.

Los algoritmos genéticos fueron propuestos por Holland (1975), quién intentando simular los procesos naturales de adaptación desarrolló por primera vez la idea de los algoritmos genéticos en los años 60. No obstante, no fue hasta 15 años más tarde cuando un pupilo suyo, David Goldberg (1989) les aplicó por primera vez a un problema real y les popularizó. En 1985 se creó la primera conferencia mundial de algoritmos genéticos ICGA que se celebra hasta el día de hoy bianualmente.

5.2. Condiciones para la aplicación de los Algoritmos Genéticos

No es posible la aplicación en toda clase de problemas Algoritmos genéticos. Para que estos puedan aplicarse, los problemas deben cumplir las siguientes condiciones:

- El espacio de búsqueda¹ debe estar acotado, por tanto ser finito.
- Es necesario poseer una función de aptitud, que denominaremos *fitness*, que evalúe cada solución (individuo) indicándonos de forma cuantitativa cuán buena o mala es una solución concreta.
- Las soluciones deben ser codificables en un lenguaje comprensible para un ordenador, y si es posible de la forma más compacta y abreviada posible.

Habitualmente, la segunda condición es la más complicada de conseguir, para ciertos problemas es trivial la función de *fitness* (por ejemplo, en el caso de la búsqueda del máximo de una función) no obstante, en la vida real a veces es muy complicada de obtener y, habitualmente, se realizan conjeturas evaluándose los algoritmos con varias funciones de *fitness*.

5.3. Ventajas e inconvenientes

5.3.1. Ventajas

- No necesitan ningún conocimiento particular del problema sobre el que trabajan, únicamente cada ejemplar debe representar una posible solución al problema.
- Es un algoritmo admisible, es decir, con un número de iteraciones *suficiente* son capaces de obtener la solución óptima en problemas de optimización.
- Los algoritmos genéticos son bastante robustos frente a falsas soluciones ya que al realizar una inspección del espacio solución de forma no lineal (por ejemplo, si quisieramos obtener el máximo absoluto de una función) el algoritmo no recorre la función de forma consecutiva por lo que no se ve afectada por máximos locales.
- Altamente paralelizables (es decir, ya que el cálculo no es lineal podemos utilizar varias máquinas para ejecutar el programa y evaluar así un mayor número de casos).

¹ Recordemos que cualquier método de Data Mining se puede asimilar como una búsqueda en el espacio solución, es decir, el espacio formado por todas las posibles soluciones de un problema.

- Pueden ser incrustables en muchos algoritmos de data mining para formar modelos híbridos. Por ejemplo para seleccionar el número óptimo de neuronas en un modelo de Perceptrón Multicapa.

5.3.2. Inconvenientes

- Su coste computacional puede llegar a ser muy elevado, si el espacio de trabajo es muy grande.
- En el caso de que no se haga un correcto ajuste de los parámetros pueden llegar a caer en una situación de *dominación* en la que se produce un bucle infinito ya que unos individuos *dominan* sobre los demás impidiendo la evolución de la población y por tanto inhiben la diversidad biológica.
- Puede llegar a ser muy complicado encontrar una función de evaluación de cada uno de los individuos para seleccionar los mejores de los peores.

5.4. Fundamentos teóricos (conceptos)

A continuación, se explican, someramente, los conceptos básicos de los algoritmos genéticos.

5.4.1. Codificación de los datos

El primer paso para conseguir que un ordenador procese unos datos es conseguir representarlos de una forma apropiada. En primer término, para codificar los datos, es necesario separar las posibles configuraciones posibles del dominio del problema en un conjunto de estados finito.

Una vez obtenida esta clasificación el objetivo es representar cada estado de forma única con una cadena de caracteres (compuesta en la mayoría de casos por unos y ceros).

A pesar de que cada estado puede codificarse con alfabetos de diferente cardinalidad² uno de los resultados fundamentales de la teoría de algoritmos genéticos es el teorema del esquema, que afirma que la codificación óptima es aquella en la que los algoritmos tienen un alfabeto de cardinalidad, es decir el uso del alfabeto binario.

² La longitud de las cadenas que representen los posibles estados no es necesario que sea fija, representaciones como la de Kitano para representar operaciones matemáticas son un ejemplo de esto.

El enunciado del teorema del esquema es el siguiente:

«Esquemas cortos, de bajo orden y aptitud superior al promedio reciben un incremento exponencial de representantes en generaciones subsecuentes de un Algoritmo Genético.»

Una de las ventajas de usar un alfabeto binario para la construcción de configuraciones de estados es la sencillez de los operadores utilizados para la modificación de estas. En el caso de que el alfabeto sea binario, los operadores se denominan, lógicamente, operadores binarios.

Es importante destacar que variables que estén próximas en el espacio del problema deben preferiblemente estarlo en la codificación ya que la proximidad entre ellas condiciona un elemento determinante en la mutación y reproducibilidad de éstas. Es decir, dos estados que en nuestro espacio de estados del universo del problema están consecutivos deberían estarlo en la representación de los datos, esto es útil para que cuando haya mutaciones los saltos se den entre estados consecutivos. En términos generales cumplir esta premisa mejora experimentalmente los resultados obtenidos con algoritmos genéticos.

En la práctica el factor que condiciona en mayor grado el fracaso o el éxito de la aplicación de Algoritmos Genéticos a un problema dado es una codificación acorde con los datos.

Otra opción muy común es establecer a cada uno de los posibles casos un número natural y luego codificar ese número en binario natural, de esta forma minimizamos el problema que surge al concatenar múltiples variables independientes en el que su representación binaria diera lugar a numerosos *huecos* que produjeran soluciones no válidas.

Por ejemplo, tenemos 3 variables, las dos primeras tienen 3 posibles estados y la última dos, el número posible de estados es $3+3+2 = 8$, combinando las 3 variables podemos codificar todo con 3 bits en comparación con los $2+2+1 = 5$ bits necesarios que utilizaríamos en el caso de realizar el procedimiento anterior. En este ejemplo no sólo ahorraríamos espacio sino que además evitaríamos que se produjeran individuos cuya solución no es factible.

5.4.2. Algoritmo

Un algoritmo genético implementado en pseudo código podría ser el siguiente:

Generar de forma aleatoria una serie de genes.

Mientras (condición de terminación es falsa).

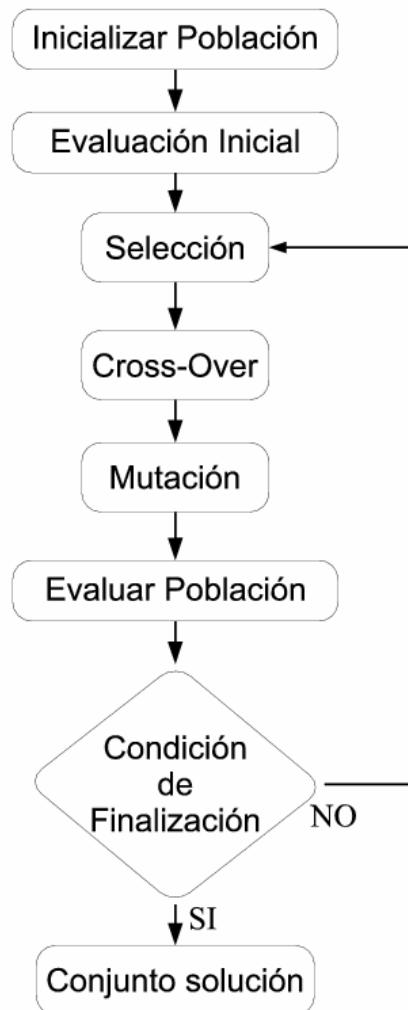
{Evaluar el *fitness* de cada uno de los individuos.

Permitir a cada uno de los individuos reproducirse de acuerdo a su *fitness*.

Emparejar los individuos de la nueva población}.

Un posible esquema que puede representar una posible implementación de algoritmos genéticos se muestra en la figura 26.

FIGURA 26. ESQUEMA DE IMPLEMENTACIÓN DE UN ALGORITMO GENÉTICO



A continuación, en los siguientes apartados, se hará una descripción de las fases anteriormente expuestas:

Iniciar Población

Como ya se ha explicado antes el primer paso es inicializar la población origen.

Habitualmente la inicialización se hace de forma aleatoria procurando una distribución homogénea en los casos iniciales de prueba. No obstante, si se tiene un conocimiento más

profundo del problema es posible obtener mejores resultados inicializando la población de una forma apropiada a la clase de soluciones que se esperan obtener.

Evaluar Población

Durante cada iteración (generación) cada gen se decodifica convirtiéndose en un grupo de parámetros del problema y se evalúa el problema con esos datos. Pongamos por ejemplo que queremos evaluar el máximo de la función $f(x)=x^2$ en el intervalo $[0,1]$ y supongamos que construimos cada gen con 6 dígitos ($2^6=64$), por lo que interpretando el número obtenido en binario natural y dividiéndolo entre 64 obtendremos el punto de la función que corresponde al gen (individuo).

Evaluando dicho punto en la función que queremos evaluar ($f(x)=x^2$) obtenemos lo que en nuestro caso sería el *fitness*, en este caso cuanto mayor *fitness* tenga un gen mejor valorado está y más probable es que prospere su descendencia en el futuro.

No en todas las implementaciones de algoritmos genéticos se realiza una fase de evaluación de la población tal y como aquí está descrita, en ciertas ocasiones se omite y no se genera ningún *fitness* asociado a cada estado evaluado.

Selección

La fase de selección elige los individuos a reproducirse en la próxima generación, esta selección puede realizarse por muy distintos métodos. En el algoritmo mostrado en pseudo código anteriormente el método de selección usado depende del *fitness* de cada individuo.

A continuación, se describen los más comunes:

- **Selección elitista:** Se seleccionan los individuos con mayor *fitness* de cada generación. La mayoría de los algoritmos genéticos no aplican un elitismo puro, sino que en cada generación evalúan el *fitness* de cada uno de los individuos, en el caso de que los mejores de la anterior generación sean mejores que los de la actual éstos se copian sin recombinación a la siguiente generación.
- **Selección proporcional a la aptitud:** los individuos más aptos tienen más probabilidad de ser seleccionados, asignándoles una probabilidad de selección más alta. Una vez seleccionadas las probabilidades de selección a cada uno de los individuos se genera una nueva población teniendo en cuenta éstas.

- **Selección por rueda de ruleta:** Es un método conceptualmente similar al anterior. Se le asigna una probabilidad absoluta de aparición de cada individuo de acuerdo al *fitness* de forma que ocupe un tramo del intervalo total de probabilidad (de 0 a 1) de forma acorde a su *fitness*. Una vez completado el tramo total se generan números aleatorios de 0 a 1 de forma que se seleccionen los individuos que serán el caldo de cultivo de la siguiente generación.
- **Selección por torneo:** se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.
- **Selección por rango:** a cada individuo de la población se le asigna un rango numérico basado en su *fitness*, y la selección se basa en este ranking, en lugar de las diferencias absolutas en el *fitness*. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable. Un ejemplo de esto podría ser que al intentar maximizar una función el algoritmo genético convergiera hacia un máximo local que posee un *fitness* mucho mejor que el de sus congéneres de población lo que haría que hubiera una dominancia clara con la consecuente desaparición de los individuos menos aptos (con peor *fitness*).
- **Selección generacional:** la descendencia de los individuos seleccionados en cada generación se convierte en la siguiente generación. No se conservan individuos entre las generaciones.
- **Selección por estado estacionario:** la descendencia de los individuos seleccionados en cada generación vuelve al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.
- **Búsqueda del estado estacionario:** Ordenamos todos los genes por su *fitness* en orden decreciente y eliminamos los últimos m genes, que se sustituyen por otros m descendientes de los demás. Este método tiende a estabilizarse y converger.
- **Selección jerárquica:** los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al

utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

Recombinación.

Recombinación también llamada *Cross-over*. La recombinación es el operador genético más utilizado y consiste en el intercambio de material genético entre dos elementos al azar (pueden ser incluso entre el mismo elemento). El material genético se intercambia entre bloques. Gracias a la presión selectiva³ irán predominando los mejores bloques génicos.

Existen diversos tipos de *cross-over*:

- Cross-over uniforme. Se genera un patrón aleatorio en binario, y en los elementos que haya un 1 se realiza intercambio genético.
- *Cross-over de n-puntos*. Los cromosomas se cortan por n puntos y el resultado se intercambia.
- *Cross-over especializados*. En ocasiones, el espacio de soluciones no es continuo y hay soluciones que a pesar de que sean factibles de producirse en el gen no lo son en la realidad, por lo que hay que incluir restricciones al realizar la recombinación que impidan la aparición de algunas combinaciones.

Mutación.

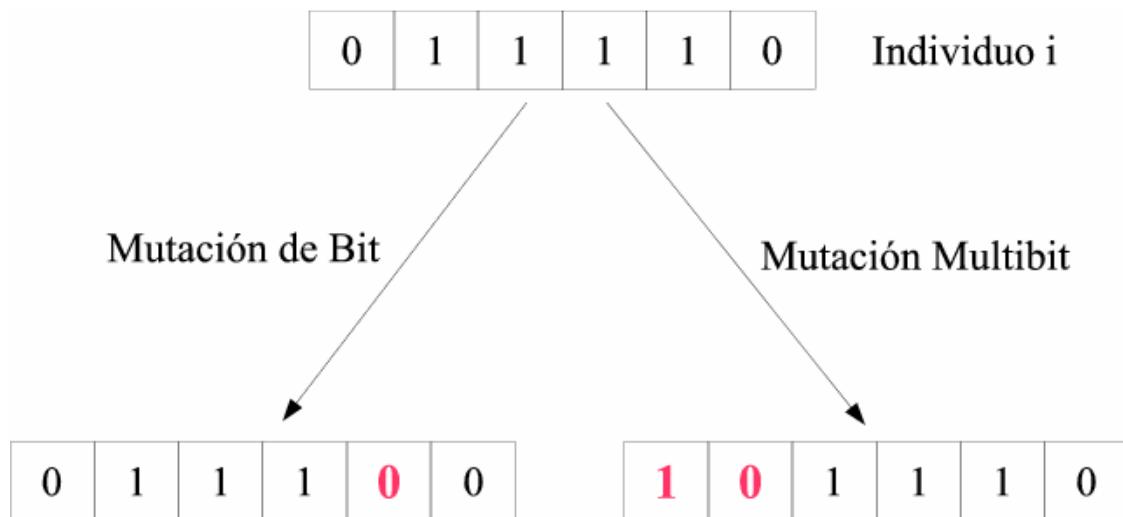
Este fenómeno, generalmente muy raro en la naturaleza, se modela de la siguiente forma: cuando se genera un gen hijo se examinan uno a uno los bits del mismo y se genera un coeficiente aleatorio para cada uno. En el caso de que algún coeficiente supere un cierto umbral se modifica dicho bit. Modificando el umbral podemos variar la probabilidad de la mutación. Las mutaciones son un mecanismo muy interesante por el cual es posible generar nuevos individuos con rasgos distintos a sus predecesores.

Los tipos de mutación más conocidos son:

³ Presión Selectiva es la fuerza a la que se ven sometido naturalmente los genes con el paso del tiempo. Con el sucesivo paso de las generaciones los genes menos útiles estarán sometidos a una mayor presión selectiva produciéndose la paulatina desaparición de estos.

- **Mutación de bit:** existe una única probabilidad de que se produzca una mutación de algún bit. De producirse, el algoritmo toma aleatoriamente un bit, y lo invierte.
- **Mutación multibit:** cada bit tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación multibit.

FIGURA 27. ESKUEMA DE MUTACIÓN MULTIBIT DE UN ALGORITMO GENÉTICO



- **Mutación de gen⁴:** igual que la mutación de bit, sólo que, en vez de cambiar un bit, cambia un gen completo. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.
- **Mutación multigen:** igual que la mutación de multibit, solamente que, en vez de cambiar un conjunto de bits, cambia un conjunto de genes. Al igual que el anterior puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.
- **Mutación de intercambio:** Se intercambia el contenido de dos bits/genes aleatoriamente.
- **Mutación de barajado:** existe una probabilidad de que se produzca una mutación.

De producirse, toma dos bits o dos genes aleatoriamente y baraja de forma aleatoria los bits, o genes, según hubiéramos escogido, comprendidos entre los dos.

- **CREEP:** Este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.

⁴ Gen e Individuo en este contexto es lo mismo.

Condición de finalización

Una vez que se ha generado la nueva población se evalúa la misma y se selecciona a aquel individuo o aquellos que por su *fitness* se consideran los más aptos. Seleccionados éstos, se toman y evalúan, y si satisfacen la condición de terminación finaliza el algoritmo.

5.4.3. Otros Operadores

Los operadores descritos anteriormente suelen ser operadores generalistas (aplicables y de hecho aplicados a todo tipo de problemas), sin embargo, para ciertos contextos suele ser más recomendable el uso de operadores específicos para realizar un recorrido por el espacio de solución más acorde a la solución buscada.

Modificadores de la longitud de los individuos.

En ocasiones las soluciones no son una combinación de todas las variables de entrada, en estas ocasiones los individuos deberán tener una longitud variable⁵.

Lógicamente, en este tipo de casos, es necesario modificar la longitud de los individuos, para ello haremos uso de los operadores añadir y quitar, que añadirán o quitarán a un individuo un trozo de su carga génica (es decir, un trozo de información).

5.4.4. Parámetros necesarios al aplicar Algoritmos Genéticos

Cualquier algoritmo genético necesita ciertos parámetros que deben fijarse antes de cada ejecución, como:

- **Tamaño de la población:** Determina el tamaño máximo de la población a obtener.

En la práctica debe ser de un valor lo suficientemente grande para permitir diversidad de soluciones e intentar llegar a una buena solución, pero siendo un número que sea computable en un tiempo razonable.

⁵ En muchas ocasiones, se realizan estudios de minería de datos sobre todos los datos existentes, encontrándose en ellos variables espúreas, es decir, variables que no aportan nada de información para el problema evaluado.

- **Condición de terminación:** Es la condición de parada del algoritmo. Habitualmente es la convergencia de la solución (si es que la hay), un número prefijado de generaciones o una aproximación a la solución con un cierto margen de error.
- **Individuos que intervienen en la reproducción de cada generación:** se especifica el porcentaje de individuos de la población total que formarán parte del acervo de padres de la siguiente generación. Esta proporción es denominada proporción de cruces.
- **Probabilidad de ocurrencia de una mutación:** En toda ejecución de un algoritmo genético hay que decidir con qué frecuencia se va a aplicar la mutación. Se debe de añadir algún parámetro adicional que indique con qué frecuencia se va a aplicar dentro de cada gen del cromosoma. La frecuencia de aplicación de cada operador estará en función del problema; teniendo en cuenta los efectos de cada operador, tendrá que aplicarse con cierta frecuencia o no. Generalmente, la mutación y otros operadores que generen diversidad se suelen aplicar con poca frecuencia; la recombinación se suele aplicar con frecuencia alta.

5.5. Selección de atributos con Algoritmos Genéticos

5.5.1. Introducción. Selección de Atributos

Un problema muy común en cualquier estudio en el que se tenga una gran cantidad de variables es determinar qué relación hay entre las mismas y la *importancia* de éstas en el problema a tratar.

Pongamos como ejemplo el problema de calificar una persona como obesa o no.

Podemos disponer de muchas variables sobre dicha persona tales como su sexo, la raza, el color de ojos, la altura, el peso, etc. Muchas de estas variables son irrelevantes o muy poco útiles para el problema que nos ocupa, por lo que convendría descartarlas para poder disminuir el tamaño de conjunto de elementos a procesar. En este ejemplo las variables irrelevantes serían: sexo, raza y color de ojos; sin embargo, variables relevantes en este problema serían el peso y la altura.

Las ventajas obtenidas por una buena selección de atributos son:

- **Eliminar el ruido:** Eliminando el ruido aumenta la precisión de los datos, y con ello, la capacidad explicativa de las predicciones del modelo.
- **Eliminar variables irrelevantes:** Solamente atendiendo a las variables relevantes se reducen los costes de la toma de datos y el tamaño de las bases de datos.

- **Eliminar redundancias:** Evitando las redundancias se evitan problemas de inconsistencias y de información duplicada.

En términos más formales, el problema de selección de atributos es el de encontrar un subconjunto de los datos tal que aplicando un algoritmo de inducción se maximice la eficiencia de éste.

5.5.2. Subconjunto de atributos óptimo

Sea un algoritmo de aprendizaje L y un conjunto de instancias X con atributos X_1, X_2, \dots, X_n con una distribución D del espacio de instancias. Se denomina subconjunto óptimo X_{opt} al subconjunto de atributos que consiguen que la eficiencia del clasificador $C=L(D)$ sea máxima.

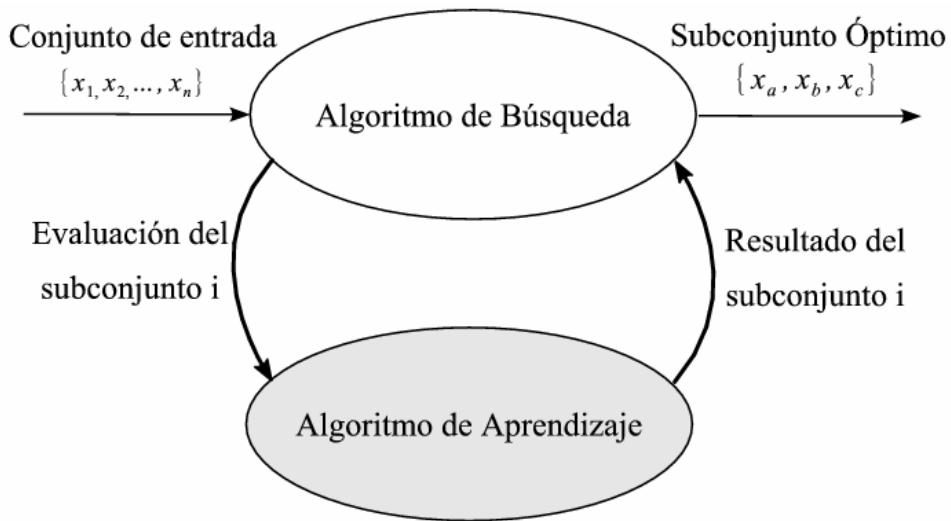
Pero, el subconjunto de atributos óptimo no tiene por qué ser único, es posible que haya combinaciones de atributos que consigan la misma eficiencia del clasificador. Un ejemplo de esto se da cuando existen dos atributos perfectamente correlacionados, en estos casos da igual el atributo que seleccionemos. Habitualmente se selecciona, si es posible, el subconjunto de atributos óptimo que sea mínimo.

Un atributo es relevante en un conjunto dado, cuando éste es significativo respecto a los demás.

El procedimiento utilizado para la selección de atributos es utilizar un método de envoltorio (*wrapper*) que se compone de un algoritmo de búsqueda, en nuestro caso Algoritmos genéticos junto a un algoritmo de aprendizaje que, en este contexto es el algoritmo que calcula el *fitness* asociado a cada uno de los individuos (subconjuntos de prueba) presentados por el algoritmo de búsqueda.

El esquema de este método se muestra en la siguiente figura.

FIGURA 28. SELECCIÓN DE ATRIBUTOS A TRAVÉS UN ALGORITMO GENÉTICO



Como algoritmo de aprendizaje utilizaremos el denominado *cfssubseteval* que evalúa el *fitness* calculando la correlación entre el individuo presentado y la clasificación, eligiendo los más correlacionados, pero penalizando la intercorrelación entre los miembros de una misma iteración (generación). Este método es muy interesante ya que es un buen partidario para utilizarlo para cualquier tipo de problemas gracias a su simplicidad.

No obstante, no es ni mucho el óptimo, recordemos que en el Data Mining la eficiencia de los métodos se debe en gran medida al conocimiento del problema en sí, o al menos del dominio de este.

5.6. Conclusiones

Los algoritmos genéticos es uno de los enfoques más originales en data mining. Su sencillez, combinada con su flexibilidad les proporciona una robustez que les hace adecuados a infinidad de problemas. No obstante, su simplicidad y sobre todo independencia del problema hace que sean algoritmos poco específicos.

Recorriendo este capítulo hemos visto los numerosos parámetros y métodos aplicables a los algoritmos genéticos que nos ayudan a realizar una adaptación de los algoritmos genéticos más concreta a un problema.

En definitiva, la implementación de esquemas evolutivos tal y como se describen en biología podemos afirmar que funciona.

5.7. Selección de variables a través de algoritmos genéticos con WEKA

Weka dispone de una amplia variedad de procedimientos para llevar a cabo la selección de variables de modelo. Una de ellas es a través de los algoritmos genéticos.

En Menu>Select attributes tenemos el conjunto de los que dispone WEKA. Elegimos el método de búsqueda y el Attribute Evaluator.

Hay que ser conscientes de que el proceso de selección de las variables para estimar los modelos es una fase vital y transcendente de la minería de datos.

En la literatura de selección de variables existen dos métodos generales para escoger las mejores características de la base de datos: métodos de filtro y métodos basados en modelos. En los primeros se filtran los atributos irrelevantes antes de aplicar las técnicas de minería de datos. El criterio que establece las variables óptimas se basa en una medida de calidad que se calcula a partir de los datos mismos. En los métodos basados en modelos, también conocidos como métodos de envolvente o wrapper, la bondad de la selección de las variables se evalúa a través de un modelo utilizando, lógicamente, un método de validación.

En el caso de la selección de atributos debemos definir un algoritmo que evaluará cada atributo individualmente del conjunto de datos inicial, que se denomina “attribute evaluator” y un método de búsqueda que realizará una búsqueda en el espacio de posibles combinaciones de todos los subconjuntos del conjunto de atributos.

De esta forma podremos evaluar independientemente cada una de las combinaciones de atributos y, con ello, seleccionar aquellas configuraciones de atributos que maximicen la función de evaluación de atributos.

Para resolver el problema de plantear combinaciones de atributos, la función que evalúa cada subconjunto de atributos es utilizar un algoritmo de búsqueda que recorre el espacio de posibles combinaciones de una forma organizada, o adecuada al problema.

Habitualmente en las situaciones en la que se emplea selección de atributos no es posible hacer un recorrido exhaustivo en el espacio de combinaciones por lo que la selección adecuada de un algoritmo de búsqueda resulta crítica.

Además del método de las componentes principales existen dos tipos de evaluadores: evaluadores de subconjuntos o selectores (SubSetVal) y prorrataedores de atributos (AttributeEval).

Los SubSetVal necesitan una estrategia de búsqueda (Search Method) y los AttributeEval ordenan las variables según su relevancia, así que necesitan un Ranker.

FIGURA Nº 29: MÉTODO DE BÚSQUEDA.

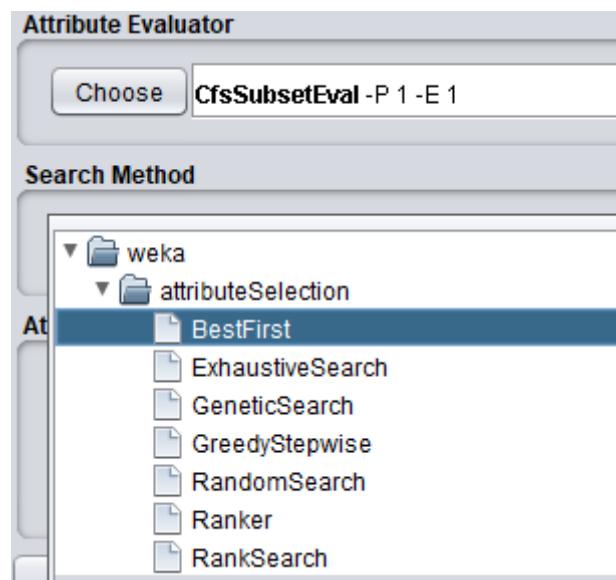
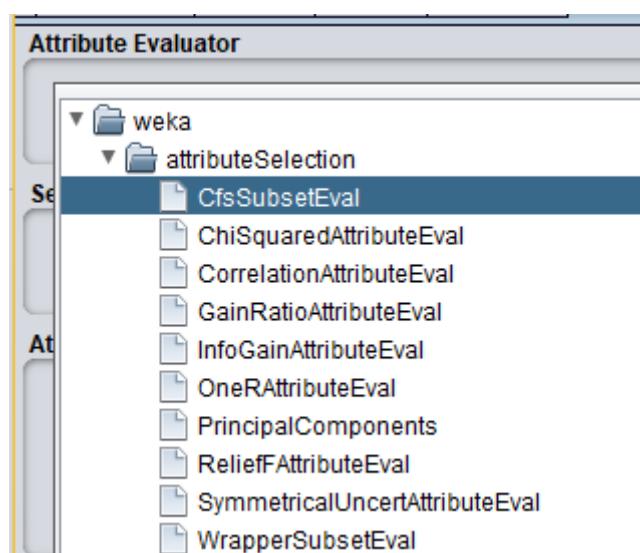


FIGURA Nº 30: EVALUADOR DE ATRIBUTOS



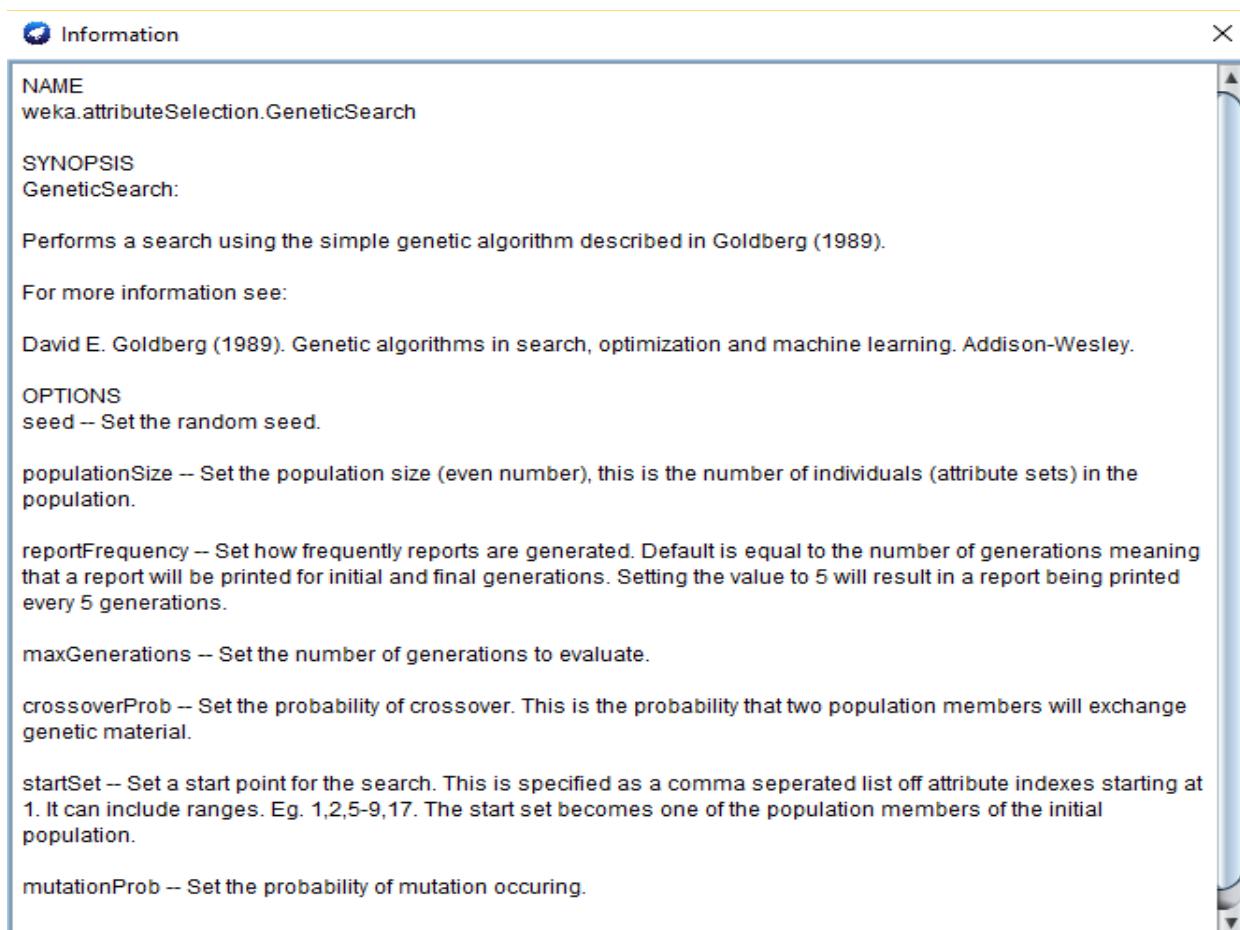
La configuración de los parámetros de los algoritmos genéticos se realiza a través de la siguiente pantalla:

FIGURA Nº 31: CONFIGURACIÓN DEL ALGORITMO GENÉTICO



Disponemos de información muy interesante en el mismo programa, activando la opción “More” y que para este método de búsqueda es la siguiente:

FIGURA Nº 32: CONFIGURACIÓN DEL ALGORITMO GENÉTICO



En este método con la configuración elegida se seleccionan tan sólo tres variables.

Attribute Subset Evaluator (supervised, Class (nominal): 21 class):

CFS Subset Evaluator

Including locally predictive attributes

Selected attributes: 1,2,3 : 3

checking_status, duration, credit_history.

La selección de atributos es una fase muy importante y también compleja del preprocesamiento de los datos. Para ver la importancia de las variables y poder comparar con otro procedimiento, a continuación, se van a ofrecer los resultados del método de búsqueda “Ranker” con el evaluador infoGain:

Search Method:

Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 21 class):

Information Gain Ranking Filter

Ranked attributes:

0.094739	1 checking_status
0.043618	3 credit_history
0.0329	2 duration
0.028115	6 savings_status
0.024894	4 purpose
0.018709	5 credit_amount
0.016985	12 property_magnitude
0.013102	7 employment
0.012753	15 housing
0.011278	13 age
0.008875	14 other_payment_plans
0.006811	9 personal_status
0.005823	20 foreign_worker
0.004797	10 other_parties
0.001337	17 job
0.000964	19 own_telephone
0	18 num_dependents
0	8 installment_commitment
0	11 residence_since
0	16 existing_credits

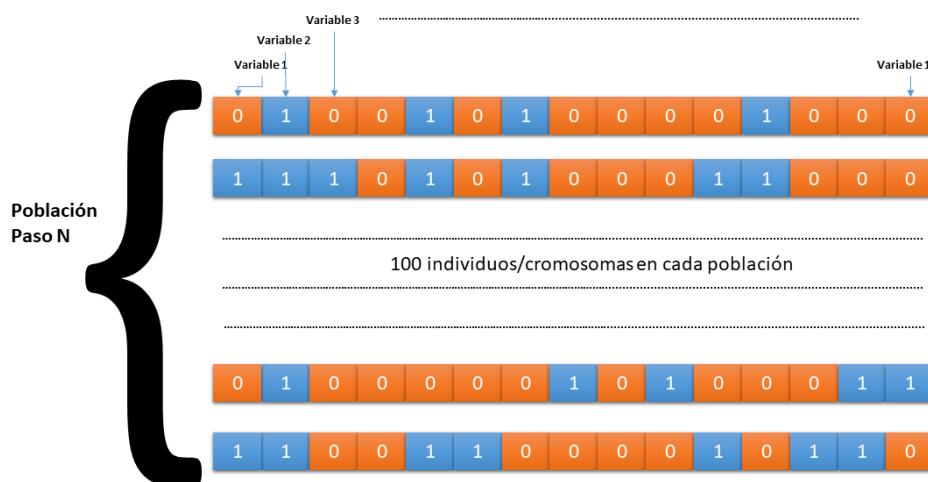
Selected attributes: 1,3,2,6,4,5,12,7,15,13,14,9

5.8. Algoritmos genéticos con R y Python

Algoritmos evolutivos con R – Selección de variables

El objetivo del ejemplo es ver cómo podemos usar un **Algoritmo Evolutivo** para hacer una **selección de variables**, quedándonos sólo con unas pocas. Para ver que estamos acertados en la selección de variables vamos a tomar el ejemplo del dataset de **Iris**, que es un problema de clasificación con 3 clases, cuenta con 150 muestras y **4 variables explicativas**. Como queremos usarlo para selección de variables lo que vamos a realizar es meter de forma **sintética 10 variables** más, que siguen una distribución **normal (0,1)** y veremos el comportamiento del algoritmo a ver si realmente aparecen las variables originales como parte de las más importantes.

Tendremos que, para el algoritmo evolutivo, nuestro **cromosoma o individuo** será un vector de **tamaño 14 (14 genes)**, que representa las 14 variables del dataset que hemos preparado.



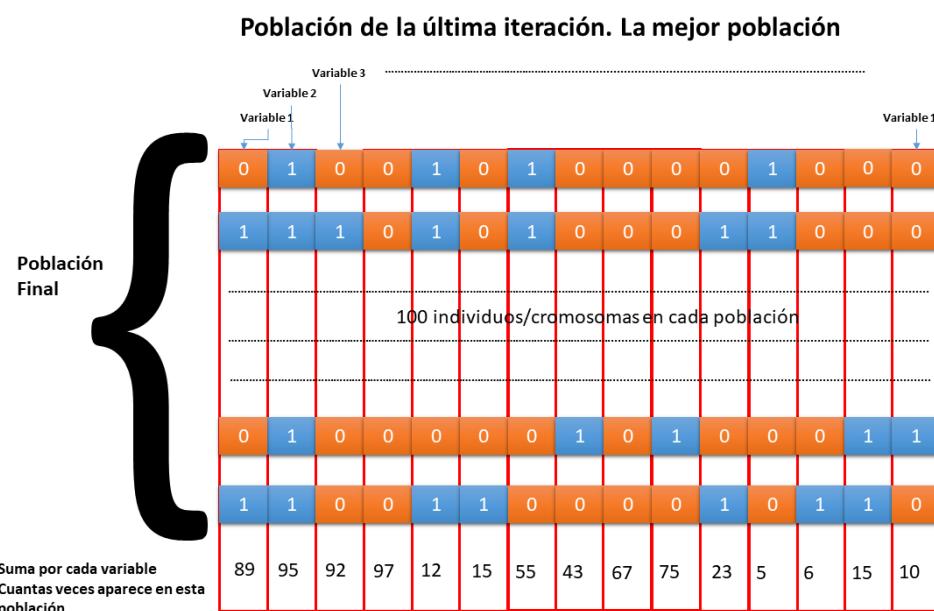
Función Fitness (Evaluación)

Cuando estamos trabajando con selección de variables, el objetivo es conseguir el conjunto de variables que mejor modelo construyan según nuestro dataset. En este caso, al ser un problema de clasificación, veremos cual es la combinación de variables que nos da menos errores al clasificar.

Nuestra función **fitness** deberá seguir estos pasos: - Recibe una variable llamada **índices** que tiene el tamaño del numero de variables (el tamaño del cromosoma) que hay en el dataframe (en nuestro caso 14) de datos. - Los valores son 1 si esa variable se va a usar y 0 si no se va a usar. - Se

construye un modelo, en este caso usamos **LDA (Análisis Discriminante Lineal)** con las variables que tienen valor 1. - Calculamos el **error** que queremos minimizar (**número de fallos**) - Para este caso del LDA cogemos los valores **\$posterior** que nos dan la probabilidad de cada clase para cada entrada de la muestra - Calculamos cual es el máximo y así le asignamos esa clase como su solución. También podríamos coger directamente el valor de **\$class** con la clase dada como predicción. - Verificamos cuantos hemos fallado y lo dividimos por el número de muestras para ver el porcentaje de fallos - Devolvemos el **porcentaje de fallos**

El resultado de la ejecución del algoritmo evolutivo (**rbga.bin()**) nos dará un objeto del que tendremos que obtener que variables son las que queremos usar.



Una vez que nuestro algoritmo pare, deberíamos tener la población que mejor se ha adaptado según el **fitness** que habíamos definido. En nuestro caso estarán las 100 mejores combinaciones de variables, que dan el menor error al clasificar. De esta manera si para cada variable contamos cuantas veces ha salido en cada elemento de la población, sabremos cuantas veces se ha usado en las combinaciones de variables de esta última iteración (que es la mejor hasta ese momento). Con lo cual podremos saber cuales han sido las variables más usadas en la población final.

El objeto **modelo_evolutivo**, que nos devuelve **rbga.bin**, tiene una variable **population** de dimensiones **tamaño_población x numero_variables**, en nuestro caso de **100x14**, que tiene la información de la población de la última iteración del algoritmo, que en un principio debería ser la mejor.

Este **population** contiene para cada fila (elemento en la población), **0 o 1** en la posición que corresponde a cada variable. De forma que para ver cuales son las variables que más se han usado tenemos que **sumar por columnas** y ese dato nos dará para cada columna (corresponde con una variable) la cantidad de veces que se ha usado en esta población.

Una vez tenemos estos datos ya podemos quedarnos con el número de variables que deseemos cogiendo las que más alto valor tienen.

Algunos de los parámetros que tenemos que definir y que algunos pueden afectar al rendimiento:

- **size**: Número de genes, en nuestro caso, número de variables
- **popsize**: Tamaño de la población con la que se trabaja en cada iteración. Cuanto más alto sea este valor, más tiempo tardará la ejecución. En este caso vamos a usar 100 elementos en la población, es decir, en cada iteración habrá 100 combinaciones de variables.
- **iters**: Número de iteraciones del algoritmo. Va a depender del problema pero seguramente entre 50 y 100 tendríamos una buena solución. Cuanto mayor sea esta variable, mayor será el tiempo de ejecución.
- **zeroToOneRatio**: Con esta variable le indicamos al algoritmo cual es el ratio de ceros frente a unos cuando se generan elementos de la población, es decir, más o menos nos tiene que dar una idea de cuantas variables se van a usar a la vez (las que tienen unos). Podríamos estimar que si tenemos 14 variables y queremos reducir a tener alrededor de 4, deberíamos intentar en cada iteración tener un rango parecido a ese. La forma de conseguirlo sería poner un valor a esta variable de 1, es decir, cada 1 cero hay 1 uno. Lo que más o menos nos dejaría alrededor de 7 variables con 1. Como esto se genera de forma aleatoria, más o menos estaremos en un rango de variables útil parecido a lo que queremos.
- **verbose**: Esta variable si la ponemos a TRUE se encargará de sacarnos información de la evolución del algoritmo. La recomendación es dejarla a FALSE ya que incrementa mucho el tiempo de ejecución.

```
library(genalg)
library(MASS)

data(iris)
set.seed(999)
# Alas variables reales del dataset, Les añadimos 10 variables más ficiticas (normal 0,1)
# Ponemos estas variables al principio
X <- cbind(scale(iris[,1:4]),matrix(rnorm(10*150), 150, 10))
Y <- iris[,5]

iris.evaluate <- function(indices) {

  result = 1
  # Tiene que haber al menos 2 variables
```

```

if (sum(indices) > 2) {

  # Creamos un modelo de clasificación con LDA usando sólo las variables que vienen
  # marcadas en la variable indices con valor 1
  # El LDA tiene el valor $posterior que devuelve la probabilidad de cada clase
  # (tenemos tres clases en Y)
  # Podríamos usar el valor de $class y así tendríamos directamente la clase.
  modelo_lda <- lda(X[,indices==1], Y, CV=TRUE)$posterior

  # Cogemos la probabilidad más alta apply(modelo_lda, 1,function(x)which(x == max(x))), para cada fila (150 muestras)
  # Comprobamos cuantos hemos fallado y lo dividimos por el tamaño de Y (150 muestras)
  # El objetivo es que sea mínimo este número de fallos
  result = sum(Y != dimnames(modelo_lda)[[2]][apply(modelo_lda, 1,function(x)which(x == max(x)))] / length(Y))

  result
}

# Ejecutamos el algoritmo evolutivo
system.time({
  modelo_evolutivo <- rbga.bin(size=14, mutationChance=0.05, zeroToOneRatio=1, evalFunc=iris.evaluate, verbose=FALSE, iters = 50, popSize = 100)
})

##      user    system elapsed
##    17.05     0.72   22.03

```

Veamos como ha quedado el resultado de la población según la gráfica que nos da cuantas veces aparece cada variable en la población final.

```

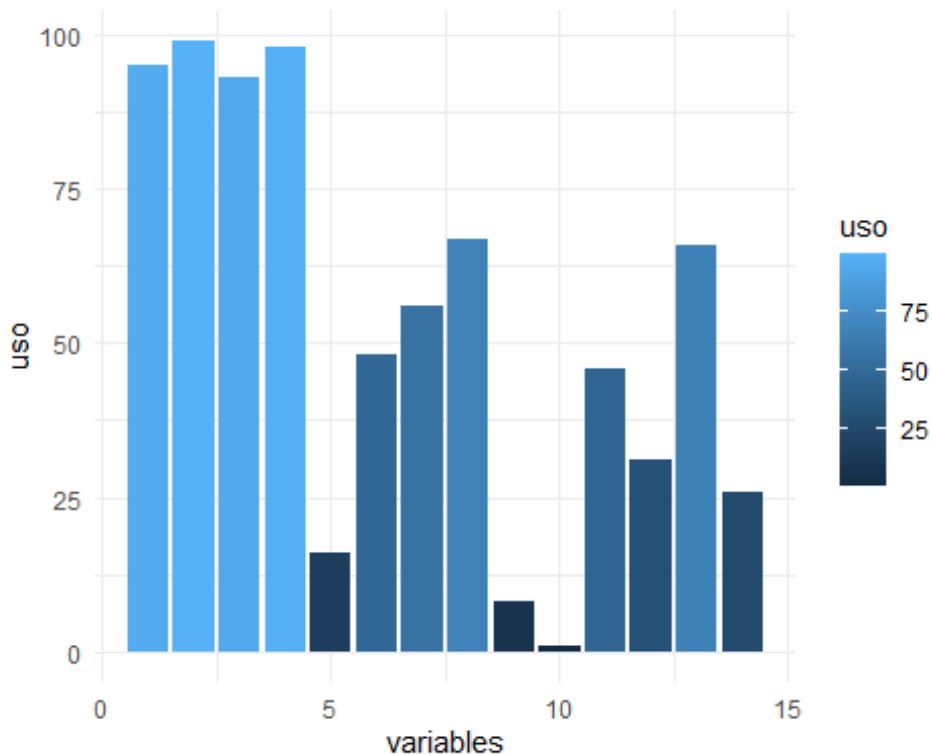
library(ggplot2)

# Mostramos cual es el que ha aparecido más veces en la última iteración
uso_variables <- colSums(modelo_evolutivo$population)

# Visualizamos cuanto se ha usado cada variable en esta última población
# Construimos un dataframe con los datos
datos <- data.frame(variables=c(1:14), uso=uso_variables)

ggplot(datos,aes(x=variables,y=uso, fill=uso)) + geom_bar(stat="identity") + theme_minimal()

```



Obtenemos ahora cuales son las variables con los valores más altos. Primero vemos las suma de las columnas, que nos dará los valores de cuantas veces aparece cada variable, y luego hacemos una función que nos dice cuales son las variables más usadas pasándole como parámetro el vector de **uso_variables** y luego cuantas variables queremos quedarnos.

```
# Creamos una función que nos da de un vector Los índices de posición
# donde están las X variables más usadas
# Le pasamos como parámetro el vector y cuantas variables queremos que devuelva

posicion_maximos <- function(datos, cuantos) {
  variables <- NULL
  if( cuantos>0)
  {
    for( i in 1:cuantos )
    {
      variables[i] <- which.max(datos)
      datos[variables[i]] <- 0
    }
  }
  variables
}

# Obtenemos ahora cuales son las 6 variables más usados
posicion_maximos(uso_variables, 6)

## [1] 2 4 1 3 8 13
```

Algoritmos evolutivos con Python

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from genetic_selection import GeneticSelectionCV

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
# Cargar datos de trabajo
os.chdir('C:/Users/p_san/Desktop/Máster_2020/Módulo_5') #directorio
datos=pd.read_csv('german_credit.csv',encoding = 'ISO-8859-1', index_col=None)
```

```
# Todas las variables son categóricas salvo:
# duration
# credit_amount
# residence_since
# age
# existing_credits
# num_dependents
```

```
# Conversión a variables categóricas
datos['checking_status']=datos['checking_status'].astype('category')
datos['credit_history']=datos['credit_history'].astype('category')
datos['purpose']=datos['purpose'].astype('category')
datos['savings_status']=datos['savings_status'].astype('category')
datos['employment']=datos['employment'].astype('category')
datos['personal_status']=datos['personal_status'].astype('category')
datos['other_parties']=datos['other_parties'].astype('category')
datos['property_magnitude']=datos['property_magnitude'].astype('category')
```

```

datos['other_payment_plans']=datos['other_payment_plans'].astype('category')
datos['housing']=datos['housing'].astype('category')
datos['job']=datos['job'].astype('category')
datos['property_magnitude']=datos['property_magnitude'].astype('category')
datos['own_telephone']=datos['own_telephone'].astype('category')
datos['foreign_worker']=datos['foreign_worker'].astype('category')
datos['class']=datos['class'].astype('category')

# La variable class es una variable reservada en diferentes módulos de Python -> reemplazar por por target
datos.rename(columns={'class': 'target'}, inplace=True)
datos['target']=np.where(datos['target']=='good', 0, 1) # cambio en la codificación por sencillez en el preprocesado

```

```

# Definición de la muestra de trabajo
datos_entrada=datos.drop('target', axis=1) # Datos de entrada
datos_entrada= pd.get_dummies(datos_entrada, drop_first=True) #conversión a variables dummy

# datos de salida
respuesta=datos.loc[:, 'target']

```

```

# Escalado de las variables, partición de la muestra y Cross Validation
seed=123

# Escalado de los datos de entrada
x_esc=StandardScaler().fit_transform(datos_entrada)
x_esc=pd.DataFrame(x_esc, columns=datos_entrada.columns)

# Partición de la muestra
test_size=0.3 #muestra para el test
x_train, x_test, y_train, y_test = train_test_split(x_esc, respuesta, test_size=test_size, random_state=seed, stratify=respuesta)

```

Usando un modelo Cart

```

from sklearn.tree import DecisionTreeClassifier
cart=DecisionTreeClassifier(max_depth=5, random_state=seed)
cart_algoritmo_gen=GeneticSelectionCV(cart,
cv=5, # 5 particiones
verbose=0, # no se muestran los resultados en la pantalla
scoring="roc_auc", # ejemplo métrica para evaluar
max_features=15, # número de variables máximas en la selección de
# características
n_population=50, # tamaño de la población
crossover_proba=0.5, # probabilidad de cruce entre parejas de genes
mutation_proba=0.2, #probabilidad de mutación
n_generations=40, #número de generaciones
crossover_independent_proba=0.5, # prob. cruce para genes
# independientes
mutation_independent_proba=0.05, # prob. mutación de genes
# independientes
tournament_size=3, #tamaño de los grupos
n_gen_no_change=10, # genes que se mantienen -> no pasan a
# la segunda generación
caching=True,
n_jobs=1)

cart_algoritmo_gen=cart_algoritmo_gen.fit(x_train, y_train)
#ajuste del modelo usando algoritmo genéticos para la selección de variables
# Variables Seleccionadas
print('Num. Var:', cart_algoritmo_gen.n_features_)
# print(cart_algoritmo_gen.support_)
# Resultados matriz numpy -> mala visualización. Los resultados se
# convierten a df de pandas
df=pd.DataFrame(cart_algoritmo_gen.support_, columns=['Variables'], index=x_train.columns)
df=df.loc[~df['Variables'].isin([False])]
#se elimina del df las variables no seleccionadas por el algoritmo
list(df.index) # variables seleccionadas por el algoritmo

```

Num. Var: 9

['checking_status_0<=X<200',

```
'checking_status_<0',
"credit_history_ 'critical/other existing credit'",
"purpose_ 'used car'",
'savings_status_>=1000',
"personal_status_ 'male single'",
'other_parties_guarantor',
'other_payment_plans_stores',
'job_skilled']
```

```
# Resultados test - predicción & Matriz de confusión (modelo CART con selección de variables a través de Algoritmos Genéticos)
```

```
pred=cart_algoritmo_gen.predict(x_test)

confusion_matrix(y_test, pred) # Matriz de confusión
```

```
array([[173,   37],
       [ 46,   44]], dtype=int64)
```

6. Lógica Difusa

6.1. Historia

Ya en el año 380 A.C., Aristóteles propone la existencia de grados de verdad o falsedad.

En el siglo XVIII, en el campo de la filosofía:

- David Hume habla de la lógica del sentido común (razonamiento basado en la experiencia que la gente comúnmente adquiere de sus vivencias por el mundo).
- Charles Sander Pierce considera la vaguedad en vez de la dicotomía cierto-falso, como una forma de enmarcar cómo el mundo y las personas funcionan.

En 1920 el filósofo polaco Jan Lukasiewicz propone la primera lógica de vaguedad. Desarrolló conjuntos con posibles valores de pertenecía 0, $\frac{1}{2}$ y 1 (lógica trivaluada). Posteriormente los extendió hacia un número infinito de valores entre 0 y 1 (lógica multivaluada).

En 1962 Lotfi Zadeh cuestiona la efectividad de las matemáticas tradicionales, las cuales resultaban intolerantes ante la imprecisión y ante verdades parciales. Dos años más tarde, en la Universidad de California en Berkeley, presenta un memorándum introduciendo la noción de **conjuntos difusos**, el cuál es publicado al año siguiente, 1965, bajo el título "**Fuzzy Sets**" en la revista "**Information and Control**". Este artículo, considerado como el inicio de la **Lógica Difusa**, describe, a través de la teoría matemática de conjuntos, como poder trabajar matemáticamente con expresiones imprecisas, tal como lo hace el ser humano. Se define el concepto de **subset difuso** como una generalización de un **subset exacto (crisp subset)** tradicional.

En realidad, todo comenzó como una broma, cuando un día el Dr. Zadeh se enzarzó con un amigo en una larga discusión acerca de cuál de sus dos mujeres era la más guapa. Cada uno consideraba que su mujer era más bonita que la del otro. Por supuesto, no existe forma objetiva de medir la belleza. El concepto de "belleza" varía mucho entre las personas. A pesar de que la discusión se prolongó durante mucho tiempo, no pudieron llegar a una conclusión satisfactoria. Esta discusión disparó el deseo de Zadeh de lograr expresar numéricamente conceptos difusos como "más bonita" o "menos bonita".

Según Zadeh, no debería considerarse la teoría difusa como una simple teoría, sino que se debería considerar el proceso de difusificación (en inglés fuzzification) como una metodología para generalizar cualquier teoría desde su versión ordinaria (discreta) a una nueva versión continua (borrosa). Así, puede hablarse de "cálculo borroso", "ecuaciones diferenciales borrosas", "autómatas borrosos", "sistemas dinámicos borrosos", etc.

En 1971, Zadeh publica el artículo, "**Quantitative Fuzzy Semantics**", en donde introduce los elementos formales que acabarán componiendo el cuerpo de la doctrina de la lógica difusa y sus aplicaciones tal como se conocen en la actualidad.

6.2. Aplicaciones

En 1974, el Británico Ebrahim Mandani, demuestra la aplicabilidad de la lógica difusa en el campo del control. Desarrolla el primer sistema de control Fuzzy práctico, la regulación de un motor de vapor.

A finales de los 70, los ingenieros daneses Lauritz Peter Holmblad y Jens-Jurgen Ostergaard desarrollan el primer sistema de control difuso comercial, destinado a una planta de cemento. Los japoneses empiezan a explotar la lógica difusa de forma masiva, apareciendo toda una serie de investigadores, tales como Sugeno, Togai o Bart Kosko (el fuzzsensei), entre otros.

En 1986, Yamakawa publica el artículo "Fuzzy Controller hardware system", en el que desarrolla controladores fuzzy en circuitos integrados.

En 1987, se inaugura en Japón el subterráneo de Sendai, uno de los más espectaculares sistemas de control difuso creados por el hombre. En "FUZZY BOOM", se comercializan multitud de productos basados en la lógica difusa (sobre todo en Japón).

Actualmente existen todo tipo de instrumentos, máquinas y procedimientos controlados borrosamente, adaptándose "inteligentemente" a cada situación particular: acondicionadores de aire, frigoríficos, lavadoras/secadoras, aspiradoras, hornos microondas, mantas eléctricas, ventiladores, autoenfoques fotográficos, estabilizadores de imágenes en grabadoras de vídeo, transmisiones de automóviles, suspensiones activas, controles de ascensores, dispensadores de anticongelantes para los aviones en los aeropuertos, sistemas de toma de decisiones industriales o económicas, y un largo etcétera.

- **Productos al consumidor:** lavadoras, hornos de microondas, procesadores de arroz, limpiadores al vacío, cámaras de video, televisores, sistemas térmicos, traductores. Es también utilizada en algunos correctores de voz para sugerir una lista de probables palabras a reemplazar en una mal dicha.
- **Sistemas:** elevadores, trenes, automóviles (máquinas, transmisiones, frenos), controles de tráfico.
- **Software:** diagnóstico médico, seguridad, compresión de datos.

6.3. Definición y principales conceptos

La lógica difusa es una metodología que proporciona una manera simple y elegante de obtener una conclusión a partir de información de entrada vaga, ambigua, imprecisa, con ruido o incompleta.

En general, la lógica difusa imita cómo una persona toma decisiones basada en información con las características mencionadas. Una de las ventajas de la lógica difusa es la posibilidad de implementar sistemas basados en ella, tanto en hardware como en software, o en combinación de ambos.

Se fundamenta en los denominados conjuntos difusos y un sistema de inferencia difuso basado en reglas de la forma “SI...ENTONCES....”

En contraste con la lógica tradicional, que utiliza conceptos absolutos para referirse a la realidad, la lógica difusa se define en grados variables de pertenencia a los mismos, siguiendo patrones de razonamiento similares a los del pensamiento humano.

Así, por ejemplo, mientras dentro del marco rígido de la lógica tradicional o formal un recinto está solamente "oscuro" (0) o claro (1), para la lógica difusa son posibles también todas las condiciones relativas intermedias percibidas por la experiencia humana como "muy claro", "algo oscuro", "ligeramente claro", "extremadamente oscuro", etc. Las condiciones extremas o absolutas asumidas por la lógica tradicional son sólo un caso particular dentro del universo de la lógica difusa. Esta última, nos permite ser relativamente imprecisos en la representación de un problema y, aun así, llegar a la solución correcta.

Se definen a continuación los principales conceptos:

- **LOGICA DIFUSA:** Es un sistema matemático que modela funciones no lineales, que convierte unas entradas en salidas acorde con los planteamientos lógicos que usan el **razonamiento aproximado**.
- **LOGICA DIFUSA EN INTELIGENCIA ARTIFICIAL:** Método de razonamiento de máquina similar al pensamiento humano, que puede procesar información incompleta o incierta, característico de muchos sistemas expertos. Con la lógica difusa o borrosa se puede gobernar un sistema por medio de reglas de “sentido común” las cuales se refieren a cantidades indefinidas. En general, la lógica difusa se puede aplicar tanto a sistemas de control como para modelar cualquier sistema continuo de ingeniería, física, biología o economía.
- **CONJUNTOS DIFUSOS:** Son imprecisos, es decir, tienen implícito un cierto grado de difusidad en la descripción de su naturaleza. Esta imprecisión puede estar asociada con su forma, posición, momento, color, textura, o incluso en lo que son.

6.4. Características

La Lógica Difusa presenta las siguientes características:

1. Se basa en palabras y no en números, las verdades de los valores son expresados lingüísticamente. Por ejemplo: caliente, muy frío, verdad, lejano, cercano, rápido, lento, medio, etc.
2. Genera algunos modificadores del predicado como, por ejemplo: mucho, más o menos, poco, suficientemente, medio, etc.
3. También procesa un sistema amplio de cuantificadores como, por ejemplo: pocos, varios, alrededor, generalmente.
4. Hace uso de las probabilidades lingüísticas como, por ejemplo: probable, improbable, que se interpretan como números borrosos, y son manipuladas por su aritmética.
5. Maneja todos los valores entre 0 y 1, tomando éstos solamente como límite.

Así, podemos decir que la Lógica Difusa, de acuerdo con sus características:

- Usa una representación de conocimiento explícito.
- Realiza verificación y optimización de manera fácil y eficiente.
- No se puede entrenar, esto es, que sea capaz de obtener nuevo conocimiento.

6.5. Etapas

1. **Fusificación (Fuzzification).** Las funciones de pertenencia definidas para las variables de entrada se aplican a sus valores actuales correspondientes, para poder determinar el grado de verdad para cada regla de la premisa.
2. **Inferencia Lógica.** El valor de verdad para la premisa de cada regla se calcula, y aplica a la parte de conclusiones de cada regla. Este resultado se asigna a un subconjunto difuso para ser asignado a cada variable de salida para cada regla.
3. **Difusificación (Defuzzification).** La cual es usada cuando se desea convertir la salida difusa en un valor puntual numérico. Existen muchos métodos.

6.6. Conjuntos difusos

La Lógica Difusa se configura como una extensión de la Lógica Multivaluada que está relacionada y fundamentada en la teoría de Conjuntos difusos, según la cual el grado de pertenencia de un elemento a un conjunto va a estar determinado por una función de pertenencia, que puede tomar todos los valores reales comprendidos en el intervalo $[0, 1]$.

Según Cantor, se definen los Conjuntos clásicos como "... cualquier reunión en un todo M de determinados objetos bien distinguidos m de nuestra intuición o pensamiento...". Esto significa que la existencia del conjunto depende de la determinación precisa de cuáles elementos pertenecen, y cuáles no, a dicho conjunto. En los Conjuntos difusos la pertenencia de un elemento a un conjunto no es tan drástica. El elemento puede tener un grado de membresía a dicho conjunto.

Los Conjuntos clásicos se pueden representar de 3 formas:

1. Enumerando los elementos del conjunto: $A = \{a, e, i, o, u\}$.
2. A través de una expresión que los miembros cumplan: $A = \{x \mid x \text{ es una letra vocal}\}$.
3. Por una función característica: $\Psi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$. Esta función mapea los elementos del conjunto universo a los elementos del conjunto $\{0, 1\}$.

En los Conjuntos difusos, esta función mapea los elementos al intervalo real $[0, 1]$. Asumiendo que X es un set, un set difuso A en X es asociado con una función característica:

$$\mu_A(x): X \rightarrow [0, 1] \quad [46]$$

Esta función característica es denominada habitualmente como **función de pertenencia** o **membership function**. El set difuso es el set de pares ordenados:

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$

Ejemplos:

- A: Conjunto de las mujeres jóvenes
- B: Conjunto de las mujeres de edad media
- C: Conjunto de las mujeres mayores

$$D = \{(x, \mu_D(x)) \mid x \in X\}, \quad \mu_D(x) = \frac{1}{1 + (\frac{x}{9} - 4)^2}$$

$X = \{0, 1, 2, 3, 4, 5, 6, 7\}$ set de hijos que puede tener una familia

$E = \{(0, 0.1), (1, 0.3), (2, 0.7), (3, 1), (4, 0.7), (5, 0.3), (6, 0.2), (7, 0.1)\}$ set difuso de número razonable de hijos que puede tener una familia

Por tanto, si X es una colección de objetos en el cual $x \in X$, un set difuso es un mapa

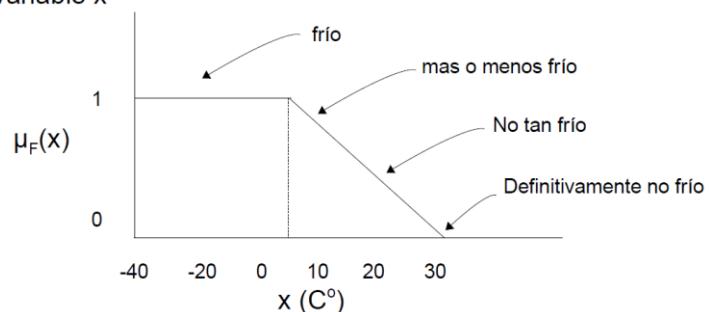
$$\mu_F(x): X \rightarrow [0, \alpha] \quad [47]$$

en el cual a cada valor x la función $\mu_F(x)$ le asigna un número entre los valores 0 a α .

El valor $\mu_F(x)$ nos informa sobre el grado de pertenencia de x al atributo F . Esto tiene que ver con el grado de ambigüedad sobre la característica de la variable que se está midiendo, pero no es una probabilidad. Un ejemplo de función de pertenencia sería el siguiente:

FIGURA 33: EJEMPLO DE FUNCIÓN DE PERTENENCIA

Ej: $\mu_F(x)$ corresponde al nivel de frío medido en la variable x

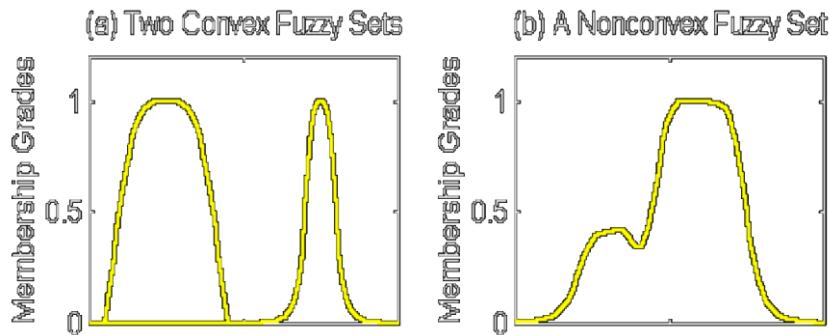


6.6.1. Definiciones sobre conjuntos difusos

- Un set difuso A en X es **convexo** si $\forall x_1, x_2 \in X$ y $\lambda \in [0, 1]$, entonces:

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2)) \quad [48]$$

FIGURA 34



- Un set difuso A en X se denomina **normal** si $\exists x \in X / \mu_A(x) = 1$. Un set difuso que no es normal se denomina **subnormal**.
- Se denomina **altura (height)** de un set difuso A , al miembro más grande de $\mu_A(x)$, es decir:

$$height(A) = \text{Max}(\mu_A(x)) \quad [49]$$

- Se denomina **soporte (support)** de un set difuso A , al subset exacto de X consistente en todos los miembros con valor de pertenencia > 0 , es decir:

$$supp(A) = \{x \mid \mu_A(x) > 0 \text{ y } x \in X\} \quad [50]$$

- Se denomina **medula (core)** de un set difuso A , al conjunto de $x \in X$ con valor de pertenencia igual a 1, es decir:

$$core(A) = \{x \mid \mu_A(x) = 1, x \in X\} \quad [51]$$

- Si A y B son dos fuzzy sets en X , A es un **subset** de B ($A \subset B$) si $\mu_B(x) \geq \mu_A(x) \quad \forall x \in X$.
- Si A y B son dos fuzzy subsets en X , $A = B$ si A es un subset de B y B es un subset de A .
- El **α -level set** de A es el crisp set en X consistente de los elementos de X para los cuales $\mu_A(x) \geq \alpha$, es decir:

$$A_\alpha = \{x \mid \mu_A(x) > \alpha, x \in X\} \quad [52]$$

- **Exponentes:** dado que $X = \{a, b, c, \dots\}$.

Si

$$A = \{x^1/a, x^2/b, x^3/c, \dots\}$$

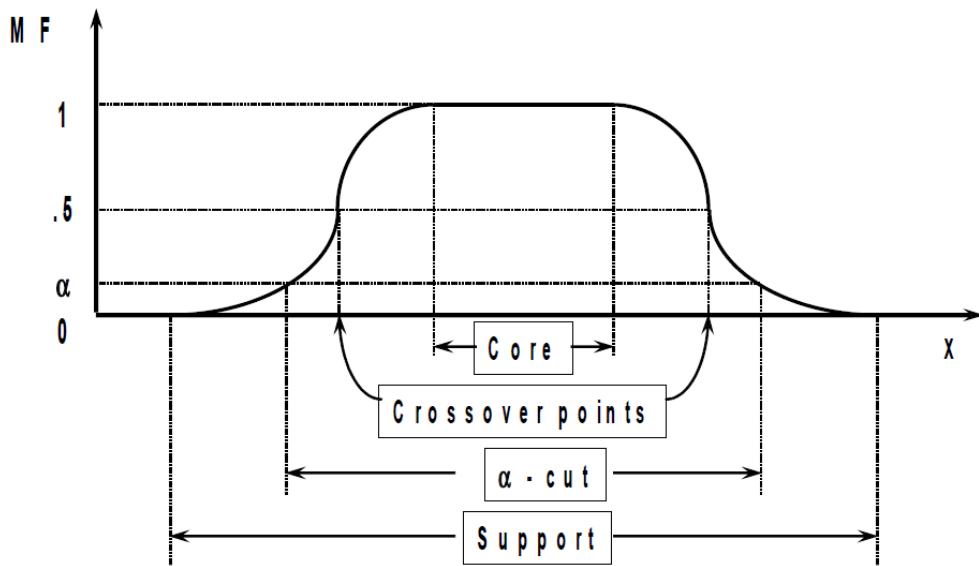
[53]

Entonces

$$A^n = \{x^{1^n}/a, x^{2^n}/b, x^{3^n}/c, \dots\}$$

[54]

FIGURA 35



6.7. Variables lingüísticas

Una variable lingüística es caracterizada por una quíntupla

$$(x, T(x), X, G, M)$$

Donde:

- x : Variable base (nombre de la variable).
- $T(x)$: Conjunto de términos lingüísticos de x que refieren a la variable base.
- X : Conjunto universo.
- G : Es una regla sintáctica (gramática) para generar términos lingüísticos.
- M : Es una regla semántica que asigna cada término con su significado $t \in T$.

La velocidad puede ser interpretada como una variable lingüística:

$$T(\text{velocidad}) = \{\text{lento}, \text{moderado}, \text{rápido}, \text{muy lento}, \text{más o menos rápido}, \dots\}$$

Cada término puede ser caracterizado por un número difuso definido sobre un conjunto universal $X=[0, 100]$. Podemos además interpretar las etiquetas:

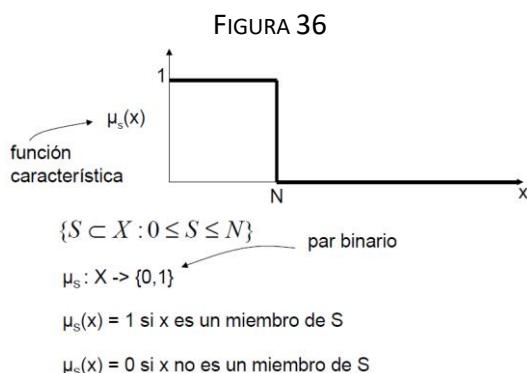
- *Lento* como “una velocidad menor a 40 Km/h”.
- *Moderado* como “una velocidad cercana a 55 Km/h”.
- *Rápido* como “una velocidad alrededor de 70 Km/h”.

Podemos encontrar el número difuso “muy lento” o “más o menos lento” a partir de “lento”:

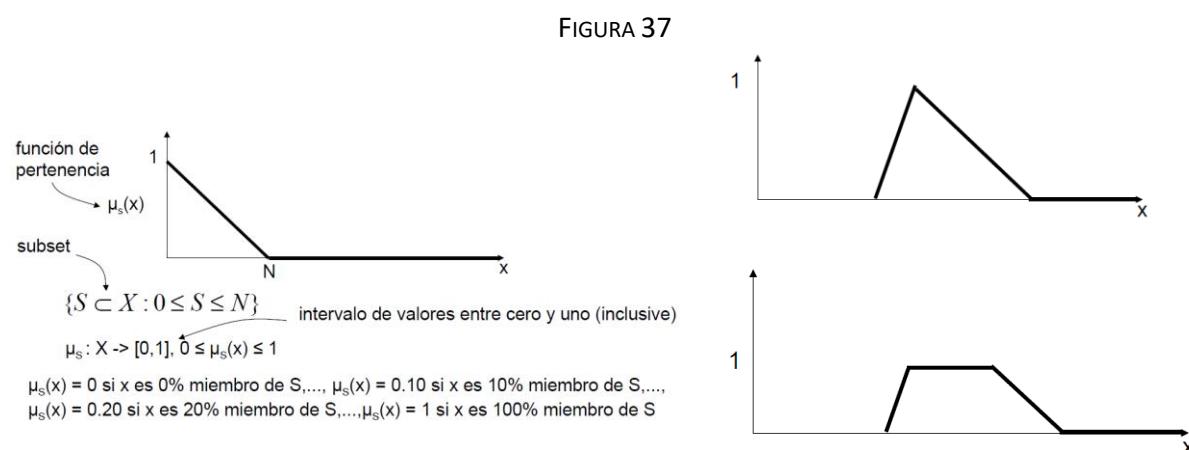
- $(\text{muy lento})(x) = (\text{lento})^p(x)$ $p > 1$.
- $(\text{más o menos lento})(x) = (\text{lento})^p(x)$ $0 < p < 1$.

6.8. Función de pertenencia o membresía

La función característica de un set exacto (crisp set) sería del tipo:

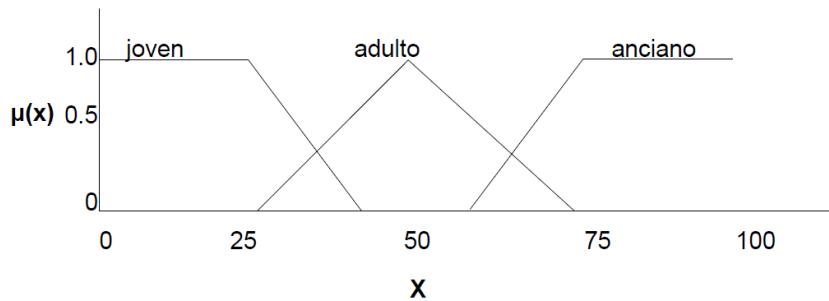


Mientras que en el caso de los sets difusos (fuzzy sets) la función de pertenencia puede tener diferentes formas como, por ejemplo:



En el caso de **variables lingüísticas** (supongamos que X = Edad y definimos los conjuntos difusos: joven, adulto, anciano), la función de pertenencia podría ser:

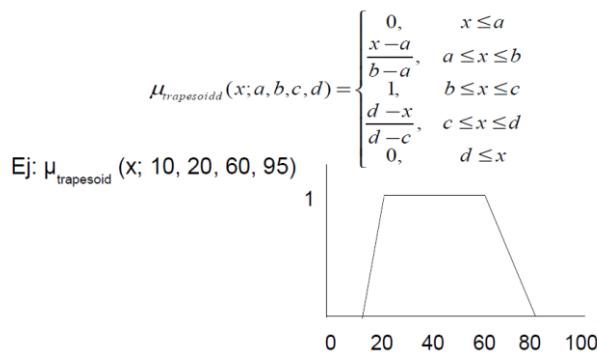
FIGURA 38:



Las principales funciones de pertenencia utilizadas son las siguientes:

FIGURA 39

TRAPEZOIDAL



TRIANGULAR

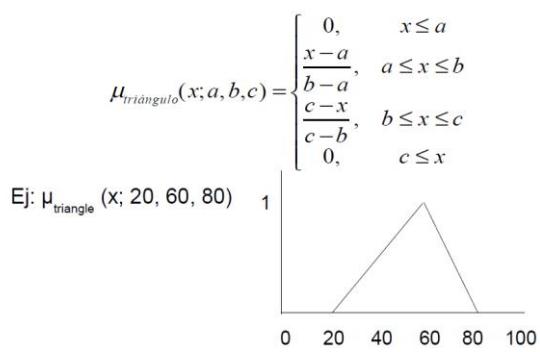
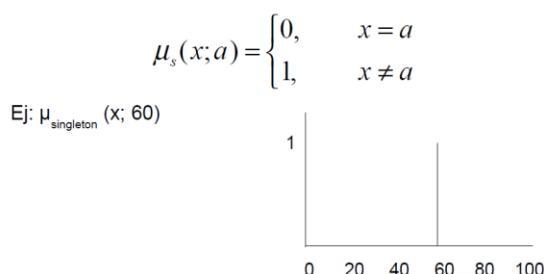
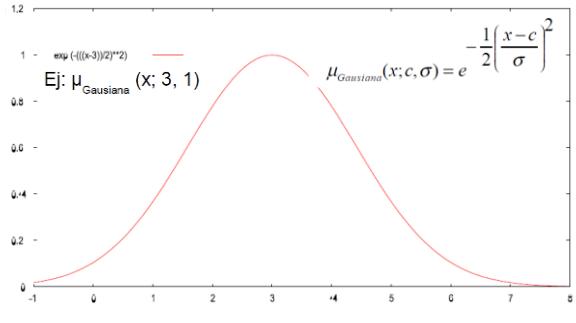


FIGURA 40

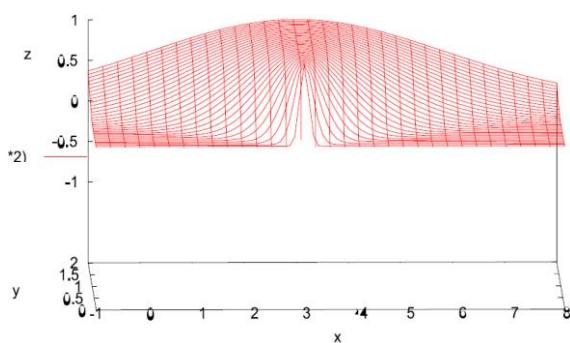
SINGLETON



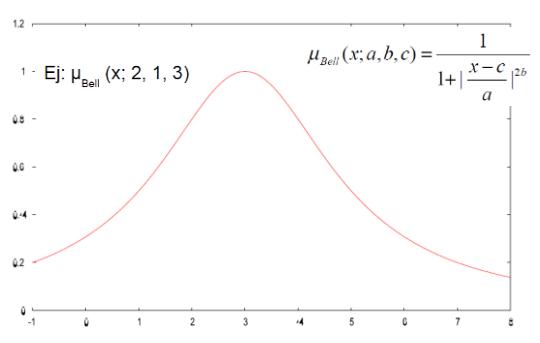
GAUSIANA



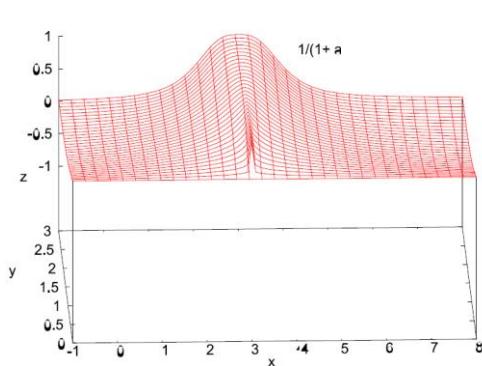
$\mu_{\text{Gausiana}}(x; 3, y)$



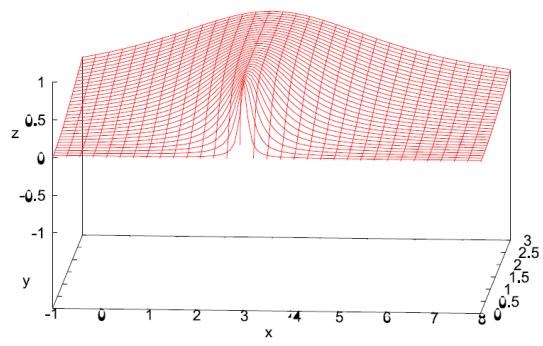
BELL



$$\mu_{Bell}(x; 1, y, 3)$$

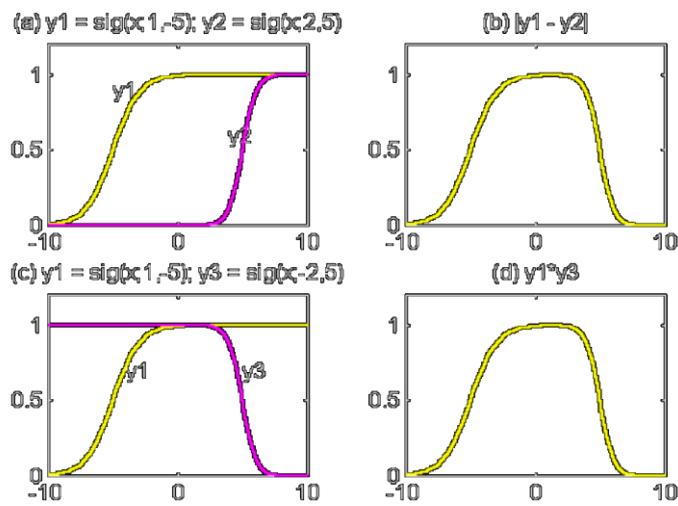


$$\mu_{Bell}(x; y, 1, 3)$$



SIGMOIDE

$$sigm f(x; a, c) = \frac{1}{1+e^{-a(x-c)}} \quad [55]$$



LR

$$L R (x; c, \alpha, \beta) = \begin{cases} F_L \left(\frac{c - x}{\alpha} \right), & x < c \\ F_R \left(\frac{x - c}{\beta} \right), & x \geq c \end{cases}$$

$$F_L(x) = \sqrt{\max(0, 1 - x^2)} \quad F_R(x) = \exp(-|x|^3)$$

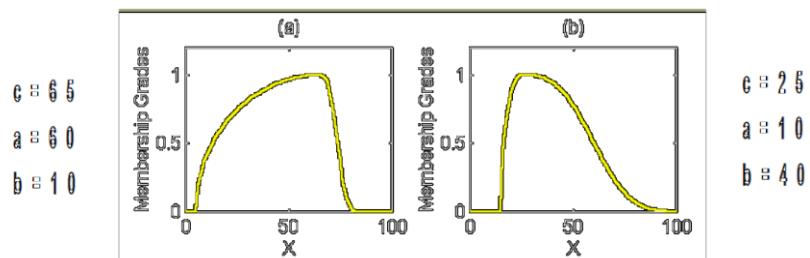
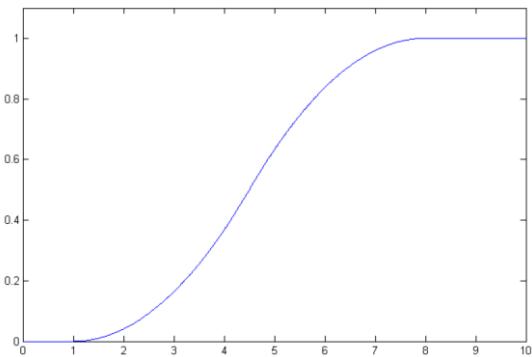


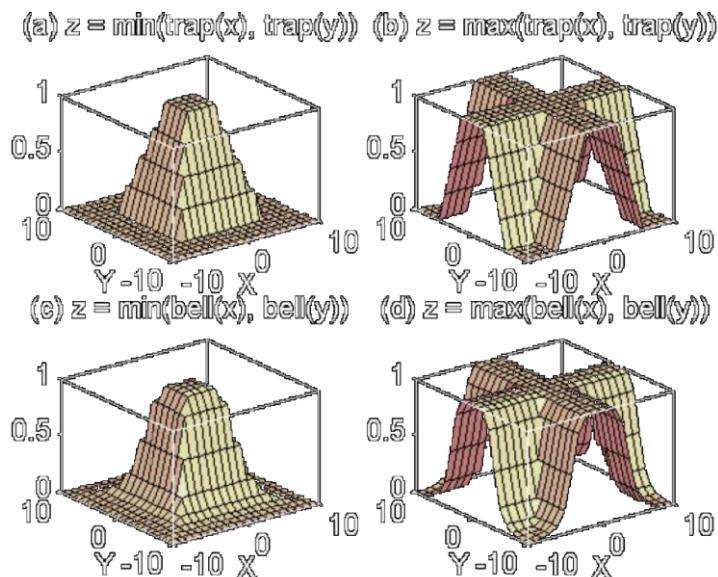
FIGURA 41: CURVAS BASADAS EN SPLINES



Las funciones de pertenencia, dado que generalmente tendremos varios inputs, pueden ser de varias dimensiones, definiéndose entonces:

$$C = \{(x, y, \mu_C(x, y)) \mid x \in X, y \in Y\} \quad [56]$$

FIGURA 42: FUNCIONES DE MEMBRESÍA EN 2D



A la hora de elegir la función de membresía, hay varias formas. El método que elegir depende de la aplicación en particular.

El método más sencillo es el Horizontal, el cual se basa en las respuestas de N expertos. La pregunta tiene el siguiente formato: ¿Puede ser x considerado compatible con el concepto A? Sólo se acepta un “si” o “no” por respuesta, y luego:

$$\mu_A(x) = (\text{Respuestas afirmativas}) / N$$

Otros métodos serían: Vertical, Método de comparación de parejas (Saaty,1980), Métodos basados en la especificación del problema, Métodos basados en la optimización de parámetros, Métodos basados en la Agrupación difusa (fuzzy clustering), Algoritmo “Fuzzy Isodata” (Bezdek,1981).

6.9. Operaciones en sets difusos

Se definen a continuación las operaciones que pueden realizarse con sets difusos:

- a) **Unión.** Sean A y B dos sets difusos en X , se define la unión de ambos como:

$$C = A \cup B \Rightarrow \mu_C(x) = \text{Max}[\mu_A(x), \mu_B(x)] \quad [57]$$

- b) **Intersección.** Sean A y B dos sets difusos en X , se define la intersección de ambos como:

$$C = A \cap B \Rightarrow \mu_C(x) = \text{Min}[\mu_A(x), \mu_B(x)] \quad [58]$$

- c) **Complemento relativo.** Sean A y B dos sets difusos en X , el complemento relativo de B con respecto a A se define como:

$$C = A - B \Rightarrow \mu_C(x) = \text{Max}[0, \mu_A(x) - \mu_B(x)] \quad [59]$$

- d) **Suma limitada (bounded sum).** Sean A y B dos sets difusos en X , se define la suma limitada como:

$$C = A \oplus B \Rightarrow \mu_C(x) = \text{Min}[1, \mu_A(x) + \mu_B(x)] \quad [60]$$

- e) **Complemento o negación.** Sean A un set difuso en X , se define el complemento o negación de A como:

$$\bar{A} = X - A \Rightarrow \mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad [61]$$

La doble negación de A es igual a A .

Propiedades:

1. **Commutatividad:**

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

2. **Idempotencia:**

$$A \cup A = A$$

$$A \cap A = A$$

3. Asociatividad:

$$A \cup (B \cup C) = (A \cup B) \cup C = A \cup B \cup C$$

$$A \cap (B \cap C) = (A \cap B) \cap C = A \cap B \cap C$$

4. Distribución:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

5. Nulo:

$$A \cup \emptyset = A$$

$$A \cap \emptyset = \emptyset$$

6. Unión e intersección con X, donde A es un subset de X:

$$A \cup X = X$$

$$A \cap X = A$$

7. Ley de De Morgan:

$$\overline{(A \cup B)} = \bar{A} \cap \bar{B}$$

$$\overline{(A \cap B)} = \bar{A} \cup \bar{B}$$

Ejemplo:

$$A = \{.1/a, .1/b, .2/c, 0/d, 1/e\}$$

$$B = \{.1/a, 0/b, .2/c, 0/d, .9/e\}$$

- A es un set fuzzy normal en X (por elemento 1/e), B es un set fuzzy subnormal en X
- height(A) = 1, height(B) = .9
- supp(A) = {a, b, c, e}, supp(B) = {a, c, e}
- core(A) = {e}, core(B) = {∅}
- B es un subset de A, $B \subset A$, ya que $\mu_A(x) \geq \mu_B(x) \quad \forall x \in X$
- $C = A \cup B = \text{Max}[\mu_A(x), \mu_B(x)] = \{.1/a, .1/b, .2/c, 0/d, 1/e\}$
- $C = A \cap B = \text{Min}[\mu_A(x), \mu_B(x)] = \{.1/a, .0/b, .2/c, 0/d, .9/e\}$
- $\bar{A} = 1 - A = \{.9/a, .9/b, .8/c, 1/d, 0/e\}$

6.10. Complementos difusos, t-normas y t-conormas

Las operaciones básicas no son únicas. Existen diversos tipos de complementos difusos, de intersecciones difusas, llamadas t-normas, y de uniones difusas, llamadas t-conormas, los cuales, describimos a continuación.

6.10.1. Complementos difusos

Dado un conjunto difuso A definido en X , tal que $x \in X$, por definición el complemento de A se puede interpretar como el grado en que x no pertenece a A .

$$\text{Comp} = C: [0,1] \rightarrow [0,1]$$

Los complementos deben satisfacer los siguientes axiomas:

1. Borde: $C(0) = 1$ y $C(1) = 0$.
2. Monotonicidad: $\forall a, b \in [0,1] \text{ si } a < b \Rightarrow C(a) > C(b)$.
3. Involución: $C[C(a)] = a \forall a \in [0,1]$.
4. C es una función continua.

Tres tipos/clases principales:

- Tipo Umbral:

$$C(a) = \begin{cases} 1 & a \leq t \\ 0 & a > t \end{cases} \quad 0 < t < 1 \quad [62]$$

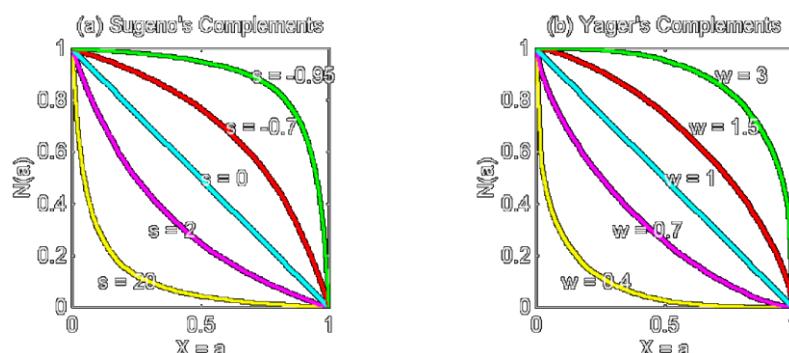
- Clase Sugeno:

$$C_\lambda(a) = \frac{1-\lambda}{1+\lambda a} \quad \lambda \in (-1, \infty) \quad [63]$$

- Clase Yager:

$$C_w(a) = (1 - a^w)^{\frac{1}{w}} \quad w \in (0, \infty) \quad [64]$$

FIGURA 43



6.10.2. T-normas

La intersección de 2 conjuntos A y B es una operación binaria sobre el intervalo unitario:

$$i: [0,1] \times [0,1] \rightarrow [0,1]$$

$$(A \cap B)(x) = i(\mu_A(x), \mu_B(x))$$

Una t-norma satisface los siguientes axiomas:

1. Borde: $i(0, 0) = 0; i(a, 1) = i(1, a) = a.$
2. Monotonicidad: $\forall a, b, c, d \in [0, 1] \text{ si } a < c \wedge b < d \Rightarrow i(a, b) < i(c, d).$
- $\forall a, b, c \in [0, 1] \text{ si } b < c \Rightarrow i(a, b) < i(a, c)$
3. Comutatividad: $i(a, b) = i(b, a).$
4. Asociatividad: $i(a, i(b, c)) = i(i(a, b), c).$
5. i es una función continua.
6. $i(a, a) < a.$

Tipos de t-normas.

- Mínimo: $i(a, b) = \text{Min}(a, b) = A \cap B.$
- Producto algebraico: $i(a, b) = a \cdot b.$
- Producto limitado o Diferencia límite: $i(a, b) = \text{Max}(0, a + b - 1) = 0 \cup (a + b - 1).$
- Producto drástico o Intersección drástica: $i(a, b) = \begin{cases} a & \text{si } b = 1 \\ b & \text{si } a = 1 \\ 0 & \text{a, b < 1} \end{cases}.$
- Yager: $i(a, b) = 1 - \text{Min}(1, [(1 - a)^w + (1 - b)^w]^{\frac{1}{w}}) \quad w > 0.$
- Schweizer & Sklar: $i(a, b) = [\text{Max}(0, a^p + b^p - 1)]^{\frac{1}{p}} \quad p \neq 0.$

6.10.3. T-conormas

La unión de 2 conjuntos A y B es una operación binaria sobre el intervalo unitario:

$$u: [0,1] \times [0,1] \rightarrow [0,1]$$

$$(A \cup B)(x) = u(\mu_A(x), \mu_B(x))$$

Una t-conorma satisface los siguientes axiomas:

1. Borde: $u(1, 1) = 1; u(a, 0) = u(0, a) = a.$
2. Monotonicidad: $\forall a, b, c, d \in [0, 1] \text{ si } a \leq c \wedge b \leq d \Rightarrow u(a, b) \leq u(c, d).$

$\forall a, b, c \in [0, 1]$ si $b \leq c \Rightarrow i(a, b) \leq i(a, c)$

3. Conmutatividad: $u(a, b) = u(b, a)$.
4. Asociatividad: $u(a, u(b, c)) = u(u(a, b), c)$.
5. u es una función continua.
6. $u(a, a) > a$.

Tipos de t-conormas.

- Máximo: $u(a, b) = \text{Max}(a, b) = A \cup B$.
- Suma algebraico: $u(a, b) = a + b - a \cdot b$.
- Suma limitada o Suma límite: $u(a, b) = \text{Min}(1, a + b) = 1 \cap (a + b)$.
- Suma drástica o Unión drástica: $u(a, b) = \begin{cases} a & \text{si } b = 0 \\ b & \text{si } a = 0 \\ 1 & a, b > 0 \end{cases}$.
- Yager: $u(a, b) = \text{Min}(1, [(a)^w + (b)^w]^{\frac{1}{w}})$ $w > 0$.
- Schweizer & Sklar: $u(a, b) = 1 - [\text{Max}(0, (1 - a)^p + (1 - b)^p - 1)]^{\frac{1}{p}}$ $p \neq 0$.

Mediante el siguiente teorema se pueden construir nuevas t-conormas a partir de una dada.

Teorema

Sea u una t-conorma y $g: [0, 1] \rightarrow [0, 1]$ una función tal que sea estrictamente creciente y continua en $(0, 1)$ y que $g(0)=0$ y $g(1)=1$. Entonces la función U definido por:

$$U(a, b) = g^{(-1)}(u(g(a), g(b))) \quad [65]$$

donde:

$$g^{(-1)}(z) = \begin{cases} 0 & \text{si } z \in (-\infty, 0) \\ g^{-1}(z) & \text{si } z \in [0, g(1)] \\ 1 & \text{si } z \in (g(1), \infty) \end{cases}$$

Es una t-conorma.

Ejemplo:

$$u(a, b) = a + b - ab$$

$$g(a) = a^2 \quad g(b) = b^2 \Rightarrow g^{(-1)}(z) = \begin{cases} 0 & \text{si } z \in (-\infty, 0) \\ \sqrt{z} & \text{si } z \in [0, 1] \\ 1 & \text{si } z \in (1, \infty) \end{cases}$$

$$u(g(a), g(b)) = u(a^2, b^2) = a^2 + b^2 - a^2 b^2$$

$$U(a, b) = g^{(-1)}(u(g(a), g(b))) = \sqrt{a^2 + b^2 - a^2 b^2} \quad \forall a, b \in [0, 1]$$

6.10.4. Principio de extensión

El principio de extensión nos da un mecanismo básico para extender las expresiones matemáticas de sets exactos al dominio difuso. Este principio generaliza la idea de un mapeo punto a punto de una función en sets tradicionales $y = f(x)$ a un mapeo entre conjuntos difusos.

Si f es una función $Y = f(X)$ y A es un set difuso sobre X definido como:

$$A = \{x_1/\mu_A(x_1), x_2/\mu_A(x_2), \dots, x_n/\mu_A(x_n)\} \quad [66]$$

Entonces el principio de extensión indica que la imagen del set A bajo la función $f()$ es el set difuso B:

$$B = \{y_1/\mu_B(y_1), y_2/\mu_B(y_2), \dots, y_n/\mu_B(y_n)\} \quad [67]$$

En el cual $y_i = f(x_i)$ y $\mu_B(y) = \text{Max}(\mu_A(x))$

Ejemplo:

$$A = \{-2/0,2, -1/0,5, 0/0,7, 1/0,9, 2/0,4\}$$

$$f(x) = x^2 - 1$$

Entonces, considerando el principio de extensión:

$$B = \{3/0,2, y_2/0,5, \dots, y_n/\mu_B(y_n)\}$$

$$\begin{aligned} B &= \{3/0,2, 0/0,5, -1/0,7, 0/0,9, 3/0,4\} = \{-1/0,7, 0/(0,5 \cup 0,9), 3/(0,2 \cup 0,4)\} \\ &= \{-1/0,7, 0/0,9, 3/0,4\} \end{aligned}$$

6.10.5. Operaciones de agregación

Son operaciones mediante las cuales se puede llevar varios conjuntos difusos a uno sólo. Una clase de operaciones de agregación es la media generalizada:

$$h_\alpha(A_1, A_2, \dots, A_n) = \left(\frac{A_1^\alpha + A_2^\alpha + \dots + A_n^\alpha}{n} \right)^{\frac{1}{\alpha}}$$

Operadores OWA (Ordered weighted averaging operation)

Sea $w = (w_1, w_2, \dots, w_n)$ vector de pesos tal que $w_i \in [0,1]$ y $\sum_{i=1}^n w_i = 1$.

Una operación OWA es la función

$$h_w(A_1, +A_2 + \dots + A_n) = w_1 b_1 + w_2 b_2 + \dots + w_n b_n = \langle W, B \rangle \quad [68]$$

Donde B es una permutación del vector A en el cual los elementos son ordenados tal que b_j es el j-ésimo elemento más grande de los a_i .

Ejemplo:

Para $x=x_0$

$$W=(0.5, 0.2, 0.3) \quad A=(0.2, 0.7, 0.9) \quad B=(0.9, 0.7, 0.2)$$

$$H=0.5*0.9+0.2*0.7+0.3*0.2=0.65$$

6.11. Inferencia usando lógica difusa

6.11.1. Relaciones difusas

Una **relación difusa** sobre un par X, Y se define como el set difuso del producto Cartesiano $X \times Y$:

$$R = \{(x, y), \mu_R(x, y) \mid (x, y) \in X \times Y\} \quad [69]$$

Es decir, las relaciones difusas binarias son mapas difusos en $X \times Y$ que mapean cada elemento en $X \times Y$ a una sola función de pertenencia (entre 0 y 1 inclusive). $\mu_R(x, y)$ puede expresarse como una matriz.

Las relaciones difusas no solo pueden ser binarias, si no que pueden ser generalizadas a n variables.

Si existen un par de sets difusos A y B su **producto cruce** (cross product) cartesiano $A \times B$ es una relación difusa T sobre el set $A \times B$, $T = A \times B$ donde:

$$T(x, y) = \text{Min}[\mu_A(x), \mu_B(y)] \quad [70]$$

Para **combinar las relaciones difusas** se usan operaciones de composición. Así, si R_1 y R_2 son dos relaciones difusas definidas en $X \times Y$ e $Y \times Z$ respectivamente, se definen del siguiente modo:

$$\mu_{R_1 \circ R_2}(x, z) = \text{Max}_Y [\mu_{R_1}(x, y) * \mu_{R_2}(y, z)] \quad [71]$$

donde * representa un operador t-norma.

Las propuestas más habituales son la **max-min** propuesta por Zadeh y la **max-product**:

$$\mu_{R_1 \circ R_2}(x, z) = \text{Max}_Y \text{Min}[\mu_{R_1}(x, y), \mu_{R_2}(y, z)]$$

$$\mu_{R_1 \circ R_2}(x, z) = \text{Max}_Y [\mu_{R_1}(x, y) \times \mu_{R_2}(y, z)]$$

6.11.2. Reglas IF-THEN difusas

Una regla IF-THEN difusa es de la forma:

IF x is A THEN y is B

En la cual A y B son variables lingüísticas definidas por sets difusos en los universos X e Y . La parte *IF x is A* es llamada el **antecedente o premisa**, mientras la parte *THEN y is B* es la **consecuencia o conclusión**.

Ejemplos:

- If presión es alta, then volumen es pequeño.
- If carretera esta mojada, then conducir es peligroso.

Si se quiere utilizar la regla **IF x is A THEN y is B ($A \rightarrow B$)** entonces se puede definir la regla como una relación binaria difusa R en el espacio $X \times Y$. R puede ser visto como un set difuso con una función de pertenencia:

$$\mu_R(x, y) = f(\mu_A(x), \mu_B(y)) \quad [72]$$

Basado en la interpretación de $(A \rightarrow B)$ “ A coupled with B ” o “ A y B ambos están” entonces podemos utilizar las funciones T-norm para resolver la relación R .

Basado en la interpretación de $(A \rightarrow B)$ “ A implies B ” o “ A implica B ” (NOT A OR B) se pueden utilizar otras funciones: Bounded sum, Max-min composition, Boolean fuzzy implicación, Gougen's fuzzy implication.

Otros operadores de implicación difusa:

FIGURA 44

Nombre	Definición $x \rightarrow y =$
Early Zadeh	$\max(1-x, \min(x, y))$
Lukasiewicz	$\min(1, 1-x+y)$
Larsen	$x \cdot y$
Estándar estricta	$\begin{cases} 1 & \text{Si } x \leq y \\ 0 & \text{Otro caso} \end{cases}$
Gödel	$\begin{cases} 1 & \text{Si } x \leq y \\ y & \text{Otro caso} \end{cases}$
Gaines	$\begin{cases} 1 & \text{Si } x < y \\ \cancel{y/x} & \text{Otro caso} \end{cases}$
Yager	y^x

6.11.3. Razonamiento difuso

El Modus Ponens en reglas de lógica tradicional indica que podemos inferir la verdad de la proposición B basados en la verdad de A y en la implicación $A \rightarrow B$:

- *premisa 1 (input):* $x \text{ es } A$
- *premisa 2 (regla):* $\text{if } x \text{ es } A \text{ then } y \text{ is } B,$
- *consecuencia:* $y \text{ es } B$

El proceso de razonamiento difuso utiliza el **Modus Ponens Generalizado (GMP)**:

- *premisa 1 (input):* $x \text{ es } A'$
- *premisa 2 (regla):* $\text{if } x \text{ es } A \text{ then } y \text{ is } B,$
- *consecuencia:* $y \text{ es } B'$

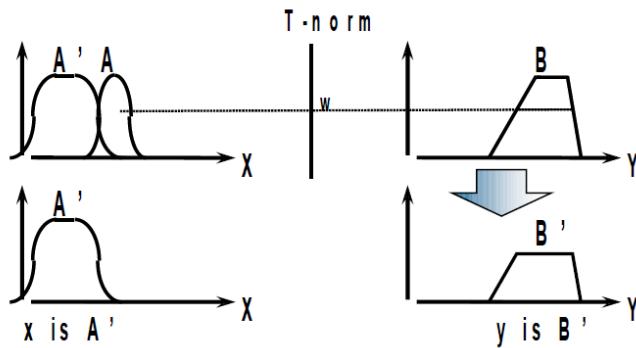
Razonamiento aproximado. Sean A, A' , y B sets difusos de X, X e Y respectivamente. Asumiendo que $(A \rightarrow B)$ se expresa como una relación R en $X \times Y$. Entonces el set difuso inducido por $x \text{ es } A'$ y la regla difusa *if x is A then y is B* se define como:

$$\mu_{B'}(y) = \text{Max}_x \text{Min}[\mu_{A'}(x), \mu_R(x, y)] \quad [73]$$

Si $\mu_{B'}(y) = \text{Max}_x \text{Min}[\mu_{A'}(x), \mu_R(x, y)]$ entonces usando las funciones de implicación de Mamdani y la regla de composición max-min:

$$\mu_{B'}(y) = \text{Max}_x \text{Min}[\mu_{A'}(x), \mu_A(x) \cap \mu_B(y)] = \text{Max}_x [\mu_{A'}(x) \cap \mu_A(x)] \cap \mu_B(y) = w \cap \mu_B(y)$$

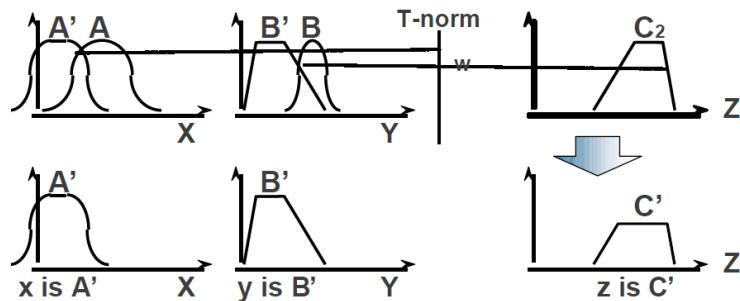
FIGURA 45



En casos con más variables:

- *premisa 1 (input): $x \text{ is } A'$ and $y \text{ is } B'$*
- *premisa 2 (regla): if $x \text{ is } A$ and $y \text{ is } B$ then $z \text{ is } C$,*
- *consecuencia $c \text{ is } C'$*

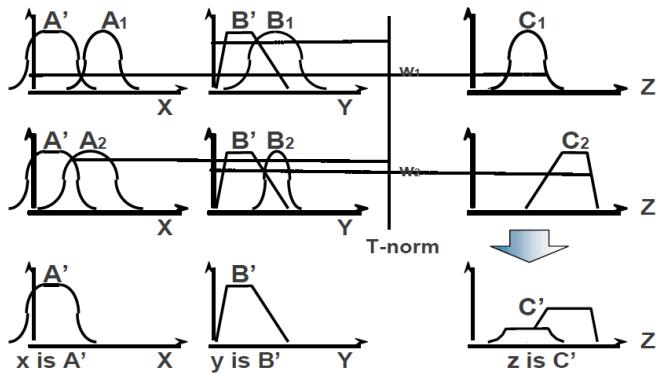
FIGURA 46



Razonamiento con dos reglas: En general se toma como la unión de las relaciones difusas correspondiente a las reglas.

- *Premisas: $x \text{ is } A'$ and $y \text{ is } B'$*
- *Regla 1: if $x \text{ is } A_1$ and $y \text{ is } B_1$ then $z \text{ is } C_1$*
- *Regla 2: if $x \text{ is } A_2$ and $y \text{ is } B_2$ then $z \text{ is } C_2$*
- *Conclusión: $z \text{ is } C'$*

FIGURA 47



El sistema de inferencia difuso se conoce por muchos nombres como: **sistema difuso de reglas, sistema experto difuso, modelo difuso, lógica asociativa difusa, controlador difuso**. Consiste en tres componentes conceptuales:

- reglas difusas,
- diccionario (con funciones de pertenencia),
- mecanismo de raciocinio.

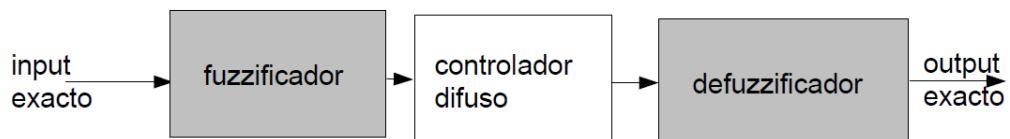
FIGURA 48



Por ejemplo, input = velocidad \rightarrow controlador \rightarrow output = flujo de gasolina.

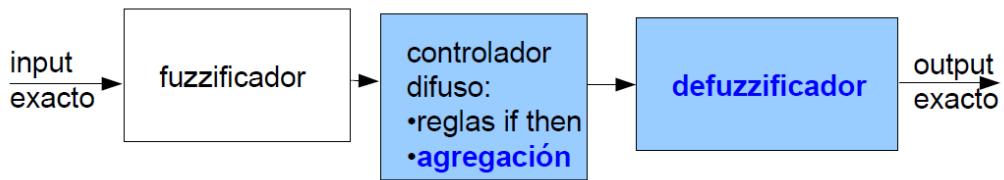
Típicamente los controladores se relacionan con el mundo externo a través de valores exactos (no difusos). Si el controlador usa lógica difusa va a ser necesario alguna conversión. Esto se denomina fusificación (fuzzification) y difusificación (defuzzification).

FIGURA 49



La principal diferencia entre los modelos de inferencia difusa está en las consecuencias de las reglas y en los métodos de agregación y difusificación.

FIGURA 50



6.11.4. Métodos de difusificación

La salida de un proceso de inferencia es un conjunto difuso. En procesos en línea, se requieren usualmente valores crisp. Algunos operadores de difusificación:

- **Operador centro de gravedad (Centroide)**

$$z_0 = \frac{\sum z_j \mu_A(z_j)}{\sum \mu_A(z_j)} \quad [74]$$

- **Primer máximo (Som)**

$$z_0 = \text{Min}(z \mid \mu_A(z) = \text{Max}_w \mu_A(w)) \quad [75]$$

- **Criterio del máximo**

$$z_0 \in \{z \mid \mu_A(z) = \text{Max}_w \mu_A(w)\} \quad [76]$$

- **Media de máximos (Mom).** Se calcula la media de los valores que maximizan a un conjunto difuso A.

- **Centro de Área.** Se calcula el valor que iguala el área de A que queda a la derecha y a la izquierda.

$$\int_{-\infty}^{z_0} \mu_A(x) dx = \int_{z_0}^{\infty} \mu_A(x) dx \quad [77]$$

- **Ultimo máximo(Lom).** Se calcula el mayor valor de los que maximizan A.
- **Bisector.** Retorna el bisector del área de difusificación de A.

6.12. Modelos (mecanismos) de inferencia difusa

Se describen en este apartado los principales mecanismos,

6.12.1. El modelo Mamdani

Fue uno de los primeros métodos de control difuso obtenidos basados en la experiencia de operadores humanos. En el modelo Mamdani se pueden usar diferentes operadores (siempre que sean T-normas o T-conormas).

Para implementar un modelo Mamdani hay que asignar un operador basado en las operaciones seleccionadas:

- AND: (usualmente T-norma) para calcular la fuerza de disparo de una regla con antecedentes que usan AND.
- OR: (usualmente T-conorma) para calcular la fuerza de disparo de una regla con antecedentes que usan OR.
- Implicación: (usualmente T-norma) para calcular consecuentes.
- Agregación: (usualmente T-conorma) para agregar consecuentes y generar una función de pertenencia del output.
- Defuzzificación: para transformar la función de pertenencia (output difuso) a un output exacto.

El modelo Mamdani (original):

- *If x is A₁ and y is B₁ then z is C₁*
- *If x is A₂ and y is B₂ then z is C₂*
- *T-norma = min*
- *T-conorma = max*

El modelo Mamdani II:

- *If x is A₁ and y is B₁ then z is C₁*
- *If x is A₂ and y is B₂ then z is C₂*
- *T-norma = product*
- *T-conorma = max*

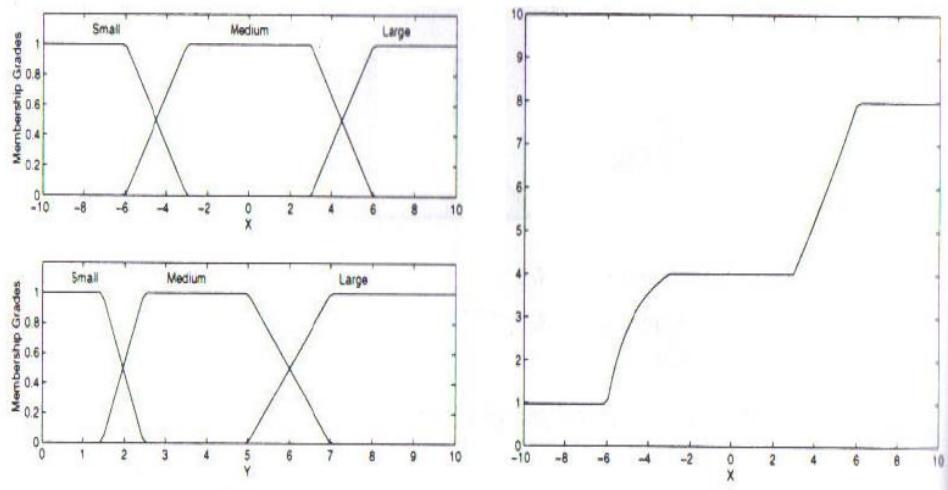
Métodos de defuzzificación usados en Mamdani:

- Centroide (Centroid o COA).
- Bisector de un Área (BOA).
- Más pequeño, medio, máximo de un máximo (SOM, MOM, LOM).

Modelo Mamdani de tres reglas con un input y un output, usando composición max-min, defuzificación centroide:

- *If X is small then Y is small*
- *If X is medium then Y is medium*
- *If X is large then Y is large*

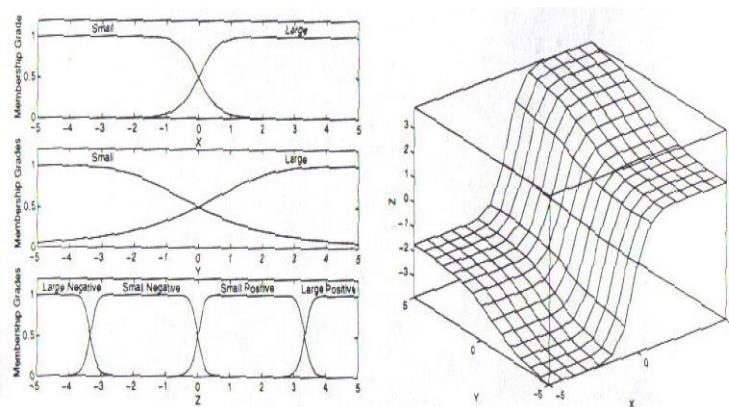
FIGURA 51



Mamdani de cuatro reglas con dos input y un output, usando composición max-min, defuzificación centroide

- *If X is small and Y is small then Z is negative large*
- *If X is small and Y is large then Z is negative small*
- *If X is large and Y is small then Z is positive small*
- *If X is large and Y is large then Z is positive large*

FIGURA 52



6.12.2. El modelo Sugeno

Este modelo utiliza una función como consecuente:

$$If x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = f(x, y) \quad [78]$$

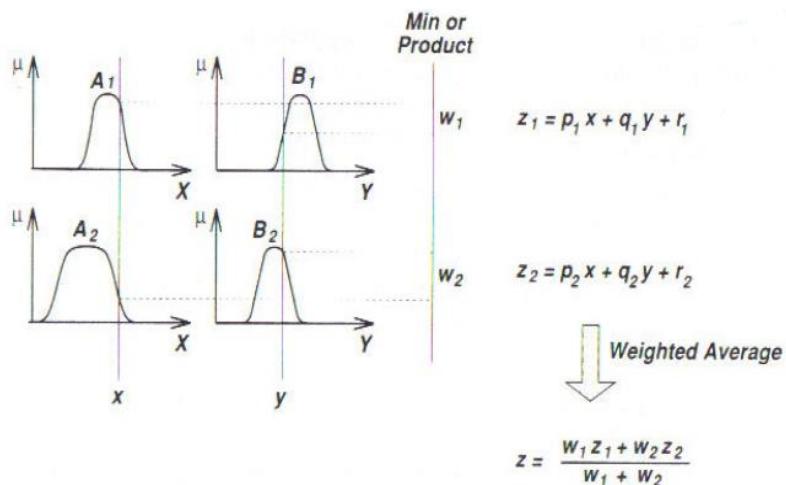
$z = f(x, y)$ es una función exacta en el consecuente, siendo $f(x, y)$ un polinomio:

- Si $f(x, y)$ es constante el modelo Sugeno es de orden cero.
- Si $f(x, y)$ es de primer orden el modelo Sugeno es de orden uno.

En el modelo Sugeno no es necesaria la difusificación, ya que cada regla tiene un output exacto, alternativas son:

- Promedio ponderado de cada regla.
- Suma ponderada de cada regla ($z' = w_1z_1 + w_2z_2$)

FIGURA 53

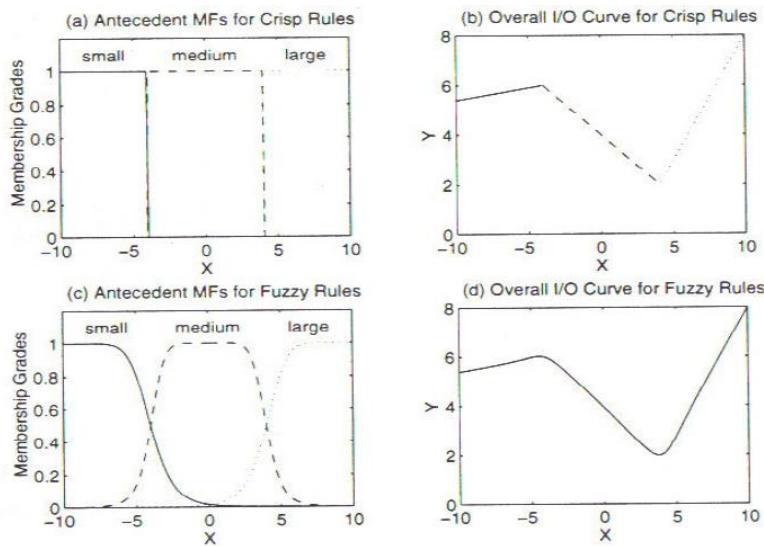


El output continuo del modelo Sugeno de orden cero depende de que las funciones de pertenencia de los antecedentes estén suficientemente solapadas.

Modelo Sugeno con antecedentes exactos y difusos. Modelo de un input:

- If X is small then $Y = 0.1x + 6.4$
- If X is medium then $Y = -0.5x + 4$
- If X is large then $Y = x - 4$

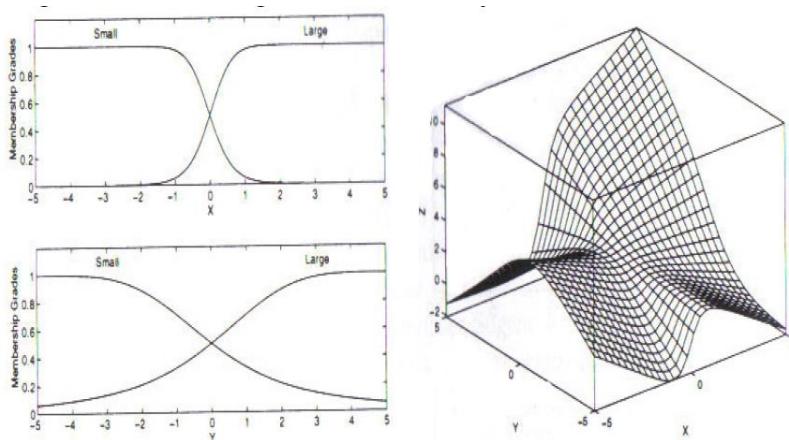
FIGURA 54



Modelo Sugeno con cuatro reglas, dos inputs y un output:

- If X is small and Y is small then $Z = -x + y + 1$
- If X is small and Y is large then $Z = -y + 3$
- If X is large and Y is small then $Z = -x + 3$
- If X is large and Y is large then $Z = x + y + 3$

FIGURA 55



6.12.3. Modelo Tsukamoto:

En este modelo la función consecuente es un set difuso con una función monótonica:

- If x is A and y is B then z is C

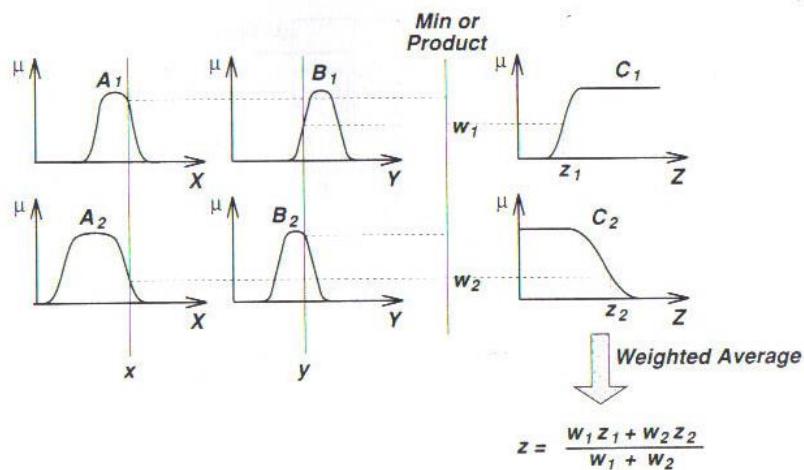
El output de cada regla se define como un valor exacto inducido por la fuerza de disparo de cada regla, es decir, cada regla tiene un output exacto.

Por tanto, no se necesita difusificacion, ya que agrega los outputs exactos de cada regla usando el promedio ponderado.

Modelo Tsukamoto con dos reglas, dos inputs y un output:

- If x is A_1 and y is B_1 then z is C_1
- If x is A_2 and y is B_2 then z is C_2

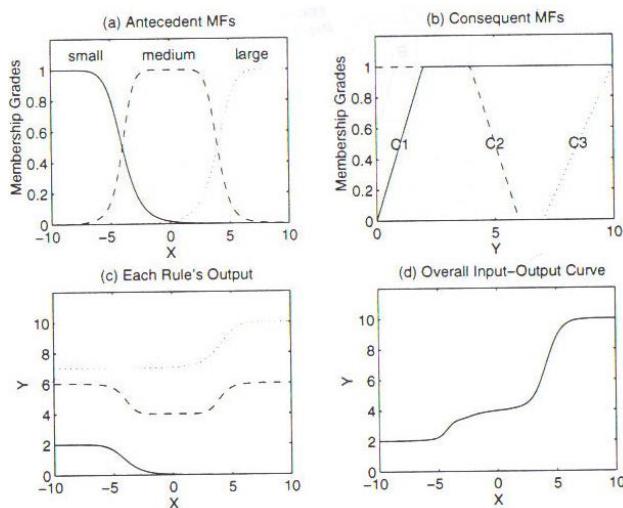
FIGURA 56



Modelo Tsukamoto con tres reglas, un input y un output

- If X is small then Y is C_1
- If X is medium then Y is C_2
- If X is large then Y is C_3

FIGURA 57



6.13. Modelamiento difuso

La idea del modelamiento difuso es dividir (partición) los posibles valores de input (antecedentes). Los consecuentes pueden ser funciones de pertenencia (Mamdani y Tsukamoto), valores constantes (Sugeno de orden cero) o funciones lineales (Sugeno).

Los diferentes consecuentes resultan en diferentes sistemas de inferencia, pero los antecedentes son los mismos.

El sistema difuso se convierte en un sistema experto en el cual las reglas que utiliza son dictadas por la lógica que utiliza el experto original (reglas \leftrightarrow conocimiento del dominio del problema).

Cuando sólo se tienen pares de datos de input \rightarrow output entonces se pueden usar métodos para identificar el sistema y modelarlo (datos numéricos \leftrightarrow conocimiento del dominio del problema).

Los pasos a seguir son los siguientes:

Etapa inicial:

- Seleccionar variables relevantes de input y output.
- Elegir un tipo específico de sistema de inferencia.
- Determinar el número de términos lingüísticos (basados en variables).
- Diseñar una colección de reglas if-then difusas.

Mejora del modelo:

- Elegir funciones de pertenencia correctamente parametrizadas.
- Mejorar las reglas y los parámetros de las funciones de pertenencia.
- Refinar los parámetros de las funciones de pertenencia usando métodos de optimización.

Se describen a continuación los distintos métodos de partición del input.

6.13.1. Grid partition

Se trata de dividir el espacio del input en celdas de igual tamaño e igual distribución. Sufre de un problema de dimensionalidad. Así:

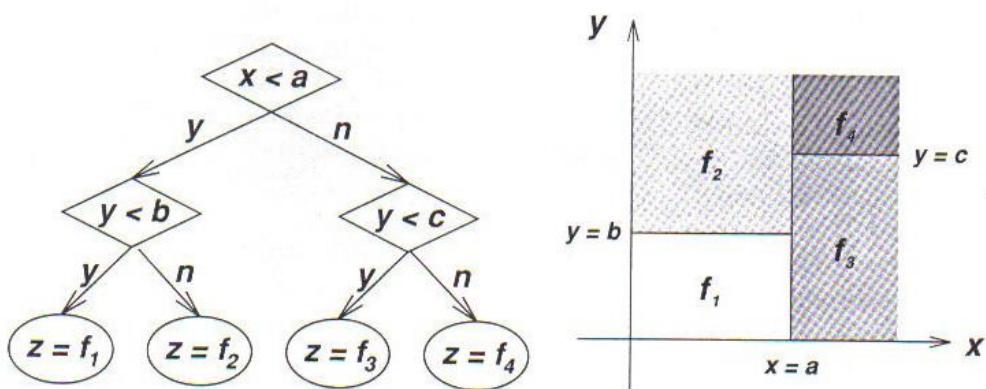
- Modelo con 3 inputs y 2 funciones (large, small) de pertenencia por input: A, B, C \rightarrow $2^3 = 8$ reglas.
- Modelo con 4 inputs y 3 funciones de pertenencia (large, medium, small) por input: A, B, C, D \rightarrow $3^4 = 81$ reglas, ...

6.13.2. Tree partition

Se trata de dividir el espacio de búsqueda en celdas de diferente tamaño. Basado en la lógica de un árbol. No tiene el problema exponencial de grid partition, pero muchas veces el significado de las variables no es tan genérico lingüísticamente como en Grid (no es tan ortogonal).

Se utiliza en el algoritmo CART.

FIGURA 58

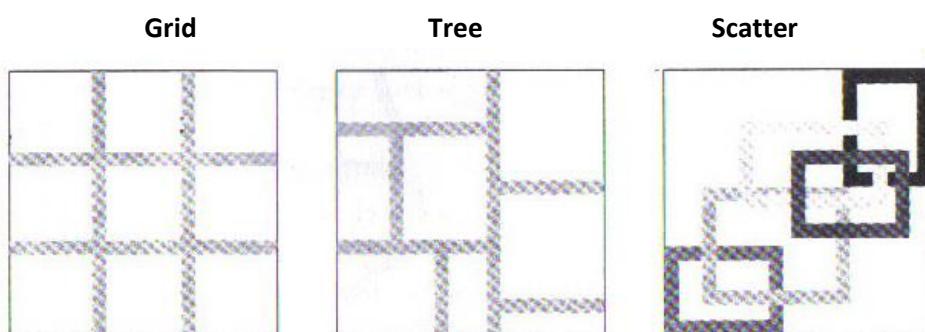


6.13.3. Scatter partition

En este caso no se cubre el espacio de búsqueda completo, sino que solo un subconjunto de éste. La partición es decidida a partir de pares específicos de datos de input-output.

El significado de las variables no es genérico lingüísticamente. No es ortogonal y es posible el solapamiento.

FIGURA 59



6.14. Modelamiento difuso con R y Python

En la actualidad existen bastantes librerías en R y Python para modelar problemas de lógica difusa.

Una muestra de ellos son los que se encuentran en el repositorio de R y que empiezan por 'fuzz':

fuzzr	Fuzz-Test R Functions
FuzzyAHP	(Fuzzy) AHP Calculation
FuzzyClass	Fuzzy and Non-Fuzzy Classifiers
FuzzyDBScan	Run and Predict a Fuzzy DBScan
fuzzyforest	Fuzzy Forests
fuzzyjoin	Join Tables Together on Inexact Matching
FuzzyLP	Fuzzy Linear Programming
FuzzyM	Fuzzy Cognitive Maps Operations
FuzzyMCDM	Multi-Criteria Decision Making Methods for Fuzzy Data
FuzzyNumbers	Tools to Deal with Fuzzy Numbers
FuzzyNumbers_Ext_2	Apply Two Fuzzy Numbers on a Monotone Function
Fuzzy.p.value	Computing Fuzzy p-Value
FuzzyQ	Fuzzy Quantification of Common and Rare Species
FuzzyR	Fuzzy Logic Toolkit for R
fuzzyRankTests	Fuzzy Rank Tests and Confidence Intervals
fuzzyreg	Fuzzy Linear Regression
FuzzyResampling	Resampling Methods for Triangular and Trapezoidal Fuzzy Numbers
fuzzySim	Fuzzy Similarity in Species Distributions
FuzzySimRes	Simulation and Resampling Methods for Epistemic Fuzzy Data
FuzzyStatProb	Fuzzy Stationary Probabilities from a Sequence of Observations of an Unknown Markov Chain
FuzzyStatTra	Statistical Methods for Trapezoidal Fuzzy Numbers
FuzzyStatTraEOO	Package 'FuzzyStatTra' in Encapsulated Object Oriented Programming
FuzzySTs	Fuzzy Statistical Tools
fuzzywuzzyR	Fuzzy String Matching

En este módulo para trabajar con R hemos escogido las opciones que nos presenta la librería que se denomina: *FuzzyR* y que ha sido creada y está soportada por la Universidad de Nottingham dentro del Lab for Uncertainty In Data and decision making (LUCID). En su página web (<https://www.lucidresearch.org>) se encuentran diversas aplicaciones interesantes de lógica difusa tanto para R como para Python.

Entre los módulos que dispone Python para la lógica difusa nos ha parecido interesante la librería *sckit-fuzzy*.

6.14.1. Sistema de inferencia difusa para controlar inundaciones

Vamos a crear un ejemplo ficticio para alertas de inundaciones con Lógica Difusa. Haremos un ejemplo pensando en una cuenca hidrográfica ficticia, donde tenemos problemas de inundación y nos gustaría crear un sistema para alertar a la población de determinados barrios.

Para tal sistema, simplificaremos las cosas usando sólo dos variables:

- Precipitación

- Distancia del río

A partir de ellas, vamos a desarrollar nuestro sistema de inferencia difusa.

En primer lugar, instalamos y cargamos la librería de R con la que vamos a trabajar

Vamos a empezar a ver la forma de programar con esta librería una aplicación de lógica difusa

Las primeras tareas serán cargar la librería, limpiar el entorno, cargar la base de datos y crear las variables. Importamos nuestra tabla de datos (comando `read.csv`) con los datos de precipitación y distancia del río y los dividimos en 2 variables.

```
#---- CARGAR LIBRERIAS Y LIMPIAR ENTORNO

library(FuzzyR)

rm(list = ls())

#---- LECTURA DE DATOS Y CREACION DE VARIABLES -----

Datos <- read.csv("datos.csv", head = TRUE, sep=";")

# Precipitación y distancia del río

prec <- datos$precipitacion_mm

dist_rio <- datos$dist_rio_m
```

A partir de nuestros parámetros de entrada y de salida, crearemos un sistema de reglas de las situaciones posibles, otorgando para cada situación diferentes niveles de alerta. Las implicaciones definidas serán:

- *SI Precipitación = Alerta y Distancia del Río = Alerta ENTONCES Nivel de alerta = Alto*
- *SI Precipitación = Alerta y Distancia del Río = Ok ENTONCES Nivel de alerta = Medio*
- *SI Precipitación = Ok y Distancia del Río = Alerta ENTONCES Nivel de alerta = Medio*
- *SI Precipitación = Ok y Distancia del Río = Ok ENTONCES Nivel de Alerta = Bajo*

Para utilizar este sistema de reglas, necesitamos convertirlo a una tabla. En la primera columna tendremos las posibilidades de la precipitación (1 = Alerta; 2 = Ok); en la segunda, las posibles distancias del río (1 y 2); en la tercera, los niveles de alerta (1 = Alto, 2 = Medio, 3 = Bajo).

Es posible también colocar pesos (cuarta columna) y cambiar el operador de la regla ($Y = 1, O = 2$) (quinta columna). En nuestro caso, vamos a asignar pesos iguales a 1 para todas las reglas y, cómo

estamos utilizando el operador Y, en la quinta columna utilizaremos el valor 1. Las cuatro reglas definidas se registran, por tanto, en la siguiente tabla:

TABLA 2

1	1	1	1	1
1	2	2	1	1
2	1	2	1	1
2	2	3	1	1

De la misma manera que importamos nuestra tabla de datos, vamos a importar nuestro sistema de reglas, transformándolo posteriormente en matriz (comando `as.matrix`)

```
#--- Importamos las reglas

# Leo matriz con reglas

reglas <- read.csv("reglas.csv", header = FALSE, sep = ";", colClasses = "numeric")
reglas <- as.matrix(reglas)

#--- Resultados

salida <- matrix(c(cbind(prec), cbind(dist_rio)), 9, 2)
```

Ahora, vamos a utilizar las funciones del paquete de lógica difusa para crear el sistema de inferencia difusa (comando `newfis`), agregaremos variables de entrada (inputs) y sus funciones de pertenencia (comandos `addvar` y `addmf`, respectivamente). Definimos así dos funciones de pertenencia de tipo trapezoidal (`trapmf`) para cada variable de entrada, es decir, una función para decir cuándo hay riesgo de inundación y cuándo no lo hay.

También seguiremos los mismos procedimientos anteriores para crear los parámetros de salida (outputs) que es el nivel de alerta.

```
#---- CREACIÓN DE FUZZY
# Mecanismo de Mamdani, con t-norma = min y método de difusificación Centroide

SIF_lluvia <- newfis(
    "SIF_lluvia",
    fisType = "mamdani",
    andMethod = "prod",
    defuzzMethod = "centroid"
)
#--- Input - 2 inputs - Precipitacion y Distancia al rio
```

```

# Añado input precipitación

SIF_lluvia <- addvar(SIF_lluvia, "input", "Precipitación", 0:50)

# Considero funciones de pertenencia trapezoidal para cada nivel del input
# Asigno las funciones de pertenencia al input 1

SIF_lluvia <- addmf(SIF_lluvia, "input", 1, 'alerta', 'trapmf', c(0, 25, 50, 50))
SIF_lluvia <- addmf(SIF_lluvia, "input", 1, 'ok', 'trapmf', c(0, 0, 25, 50))

# Input distancia

SIF_lluvia <- addvar(SIF_lluvia, "input", "Distancia del Rio", 0:100)
SIF_lluvia <- addmf(SIF_lluvia, "input", 2, 'alerta', 'trapmf', c(0, 0, 25, 50))
SIF_lluvia <- addmf(SIF_lluvia, "input", 2, 'ok', 'trapmf', c(25, 50, 100, 100) )

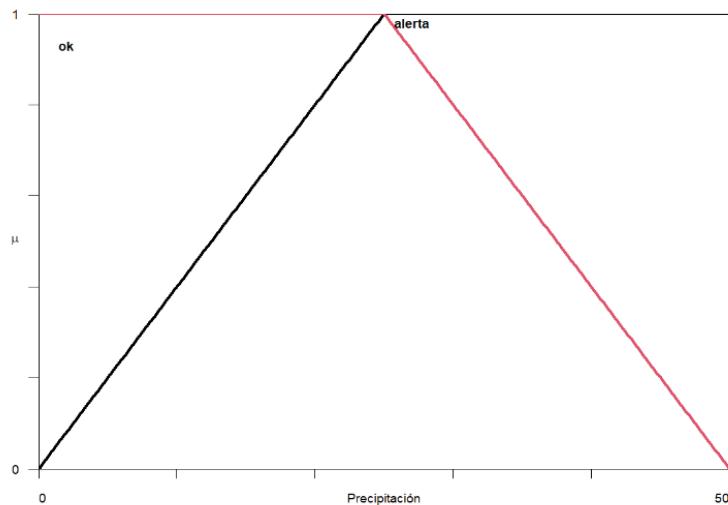
#--- Output - 1 output
# Output

SIF_lluvia <- addvar(SIF_lluvia, "output", "Nivel de Alerta", 0:10)
SIF_lluvia <- addmf(SIF_lluvia, "output", 1, 'Alto', 'trapmf', c(6, 7, 10, 10))
SIF_lluvia <- addmf(SIF_lluvia, "output", 1, 'Medio', 'trapmf', c(3, 5, 6, 7))
SIF_lluvia <- addmf(SIF_lluvia, "output", 1, 'Bajo', 'trapmf', c(0, 0, 3, 5))

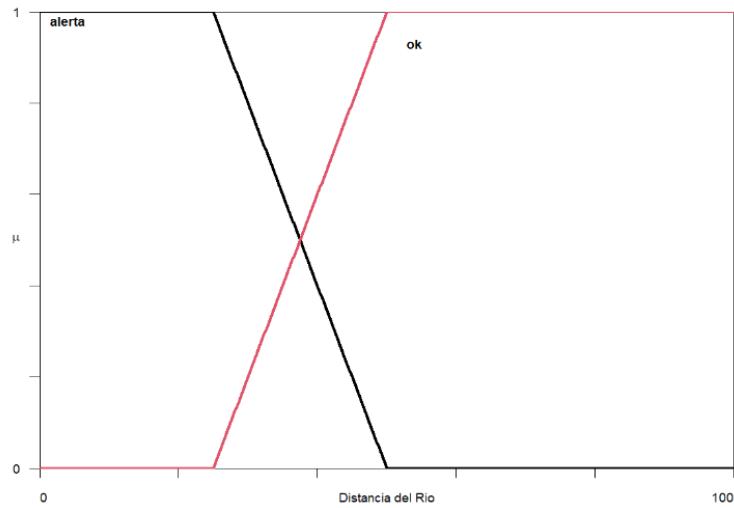
```

Podemos dibujar las funciones de membresía que hemos especificado utilizando con el comando **plotmf**. Se necesita especificar el nombre del sistema de lógica difusa, si es un input o un output y el lugar que ocupa.

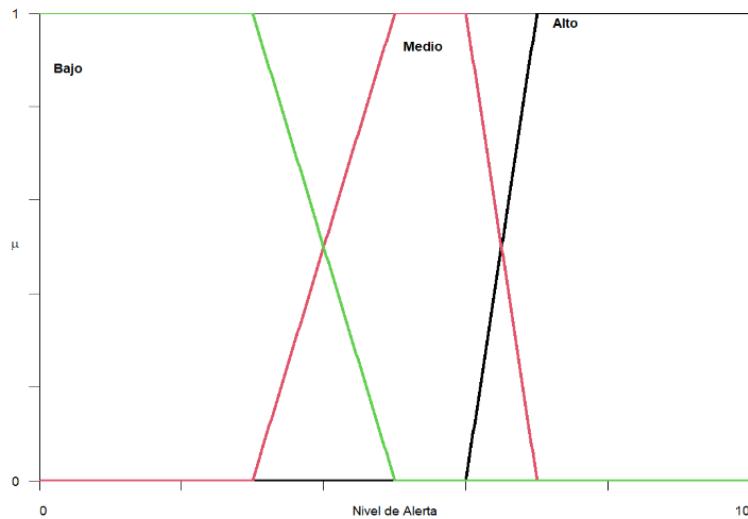
plotmf(SIF_lluvia,"input",1)



```
plotmf(SIF_lluvia,"input",2)
```



```
plotmf(SIF_lluvia,"output",1)
```



Y ahora vamos a agregar las reglas a nuestro sistema de inferencia difusa (comando *addrule*). Con esto, ya tenemos nuestro sistema de inferencia difusa completo.

```
# Incorporación de las reglas al sistema  
SIF_lluvia <- addrule(SIF_lluvia, reglas)
```

Visualizamos los datos de entrada que están almacenados en la variable salida y lo vamos a someter a nuestro sistema (comando evalfis) para verificar si algún barrio está con nivel de alerta alto.

```
print(salida)
```

	[,1]	[,2]
[1,]	30	15
[2,]	30	30
[3,]	30	50
[4,]	30	15
[5,]	30	30
[6,]	30	50
[7,]	30	15
[8,]	30	30
[9,]	30	50

En la primera variable se recoge la precipitación y en la segunda la distancia al río.

```
----- EVALUACION RESULTADOS  
resultados <- evalfIS(salida, SIF_lluvia)  
print(resultados)
```

	[,1]
[1,]	7.092780
[2,]	6.468175
[3,]	3.359259
[4,]	7.092780
[5,]	6.468175
[6,]	3.359259
[7,]	7.092780
[8,]	6.468175
[9,]	3.359259

En el análisis de la variable *resultados*, tenemos tres niveles de alerta (7,09, 6,46 y 3,36), donde el primero es para locales a 15 metros del río, o sea, en esa situación (precipitación de 30 mm) tenemos una alerta de nivel alto (7,09). Los locales a 30 metros también presentan un cierto nivel de alerta (6,46); mientras que los locales a 50 metros del río presentan un riesgo bajo (3,36).

Recordar que definimos que los niveles de alerta varían entre 0 y 10.

Aunque lo mostrado es una aplicación simple, los sistemas de inferencia difusa pueden llegar a ser complejos cuando hay un mayor número de variables y reglas. Sin embargo, en esta simple aplicación, es posible verificar su gran potencial en diversas áreas.

Para finalizar este epígrafe, vamos a ver las salidas y resultados que nos presentan dos comandos de esta librería muy interesantes. Nos van a permitir observar todos los parámetros de nuestra aplicación de lógica difusa y la GUI de esta librería que nos posibilita realizar simulaciones a través del menú de la librería.

```
# Ver los parámetros y opciones que hemos utilizado en esta aplicación
showfis(SIF_lluvia)
```

```

1. Name           SIF_lluvia
2a. Type          mamdani
2b. mfType        t1
3. Inputs/Outputs [ 2 1 ]
4. NumInputMFs   [ 2 2 ]
5. NumOutputMFs  [ 3 ]
6. NumRules       4
7. AndMethod     prod
8. OrMethod      max
9. ImpMethod     min
10. AggMethod    max
11. DefuzzMethod centroid
12. InLabels     Precipitación
13. InLabels     Distancia del Rio
14. OutLabels    Nivel de Alerta
15. InRange       [ 0 50 ]
16. InRange       [ 0 100 ]
17. OutRange      [ 0 10 ]
18. InMFLabels   alerta
19. InMFLabels   ok
20. InMFLabels   alerta
21. InMFLabels   ok
22. OutMFLabels  Alto
23. OutMFLabels  Medio
24. OutMFLabels  Bajo
25. InMFTypes    trapmf
26. InMFTypes    trapmf
27. InMFTypes    trapmf
28. InMFTypes    trapmf
29. OutMFTypes   trapmf
30. OutMFTypes   trapmf
31. OutMFTypes   trapmf
32. InMFParams   [ 0 25 50 50 ]
33. InMFParams   [ 0 0 25 50 ]
34. InMFParams   [ 0 0 25 50 ]
35. InMFParams   [ 25 50 100 100 ]
36. OutMFParams  [ 6 7 10 10 ]
37. OutMFParams  [ 3 5 6 7 ]
38. OutMFParams  [ 0 0 3 5 ]
39. Rule Antecedent [ 1 1 ]
40. Rule Antecedent [ 1 2 ]
41. Rule Antecedent [ 2 1 ]
42. Rule Antecedent [ 2 2 ]
43. Rule Consequent 1
44. Rule Consequent 2
45. Rule Consequent 2
```

```

46. Rule Consequent    3
47. Rule Weight        1
48. Rule Weight        1
49. Rule Weight        1
50. Rule Weight        1
51. Rule Connection    1
52. Rule Connection    1
53. Rule Connection    1
54. Rule Connection    1

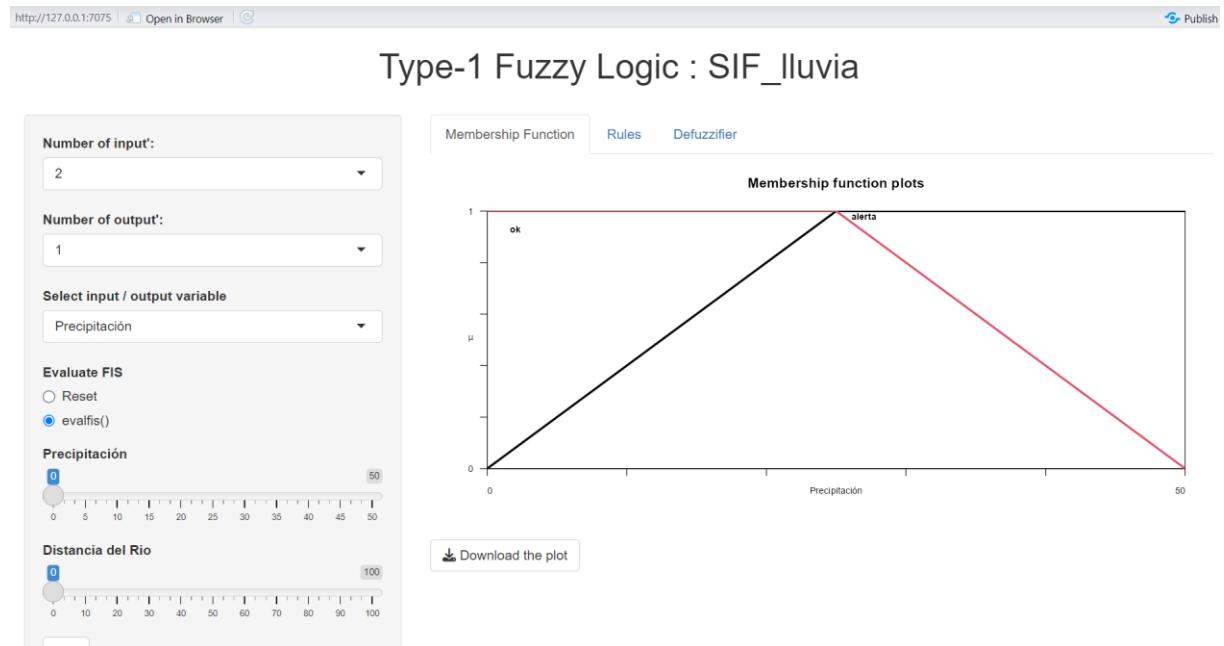
```

```

#--- GUI
showGUI(SIF_lluvia)

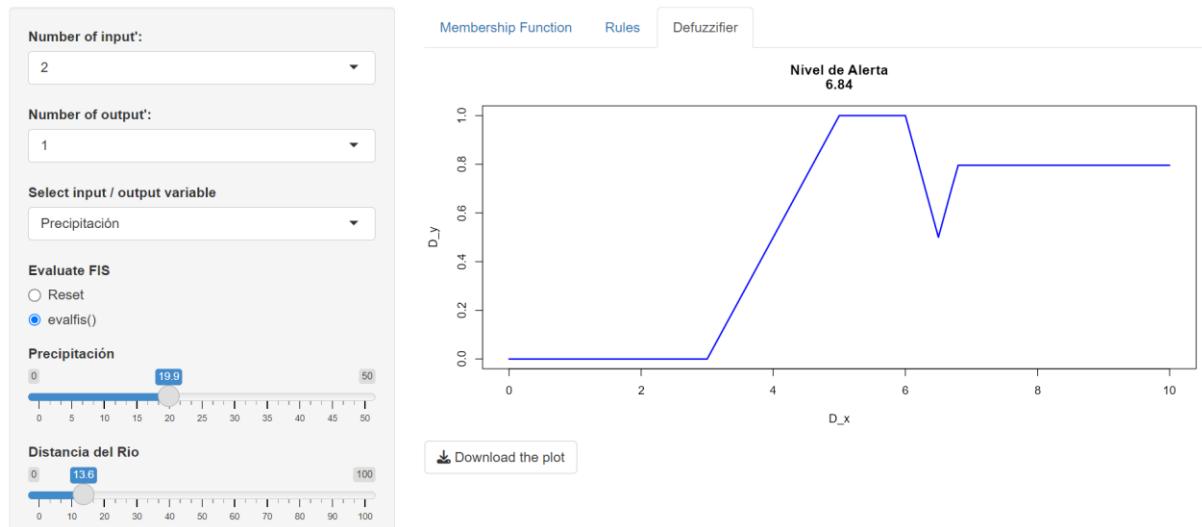
```

Al utilizar el comando showGUI nos presenta el sistema de lógica difusa que hemos construido en el navegador.

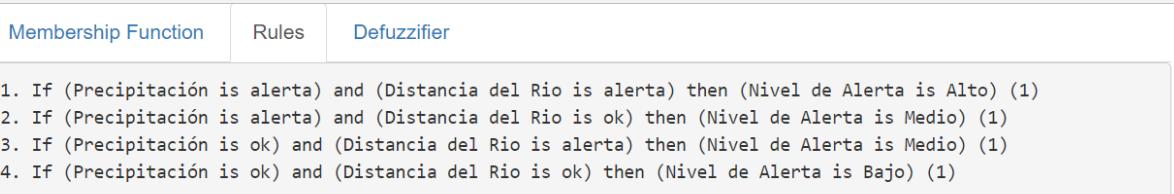


Podemos ver los resultados variando los valores de precipitación y distancia al río.

Type-1 Fuzzy Logic : SIF_lluvia



Si vamos al menú Rules podemos **visualizar** las reglas.



6.14.2. Algoritmo Fuzzy C-Means para agrupamiento

En muchas situaciones cotidianas ocurre el caso que un dato está lo suficientemente cerca de dos clusters, de tal manera que es difícil etiquetarlo en uno o en otro. Esto se debe a la relativa frecuencia con la cual un dato particular presenta características pertenecientes a clusters distintos y como consecuencia no es fácilmente clasificado; fuzzy c-means (FCM) es un algoritmo que se desarrolló con el objetivo de solucionar tales inconvenientes.

El algoritmo FCM asigna a cada dato un valor de pertenencia dentro de cada cluster y por consiguiente un dato específico puede pertenecer parcialmente a más de un cluster. A diferencia del algoritmo c-means clásico, que trabaja con una partición dura, FCM realiza una partición suave del conjunto de datos. En tal partición los datos pertenecen en algún grado a todos los clusters. El siguiente código muestra cómo hacer en R un agrupamiento difuso. Se parte del dataset iris.

```

data(iris)
x=iris[,-5]
pairs(x, col=iris[,5])

# Tres clústeres con los valores predeterminados de los argumentos
res.fcm <- ppclust::fcm(x, centers=3)

```

```

# Inicializar la matriz del prototipo usando el algoritmo K-means++
v <- inaparc::kmpp(x, k=3)$v

# Inicializar la matriz de grados de membresía
u <- inaparc::imembrand(nrow(x), k=3)$u

# Ejecutar FCM con los prototipos y membresías iniciales
fcm.res <- ppclust::fcm(x, centers=v, memberships=u, m=2)

# Muestre los grados de membresía difusos para los 5 primeros objetos
head(fcm.res$u, 5)

```

```

Cluster 1    Cluster 2    Cluster 3
1 0.9966236  0.001072034 0.002304380
2 0.9758525  0.007497947 0.016649509
3 0.9798259  0.006414579 0.013759500
4 0.9674274  0.010107523 0.022465031
5 0.9944704  0.001767935 0.003761709

```

6.14.3 Lógica difusa en Python

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import skfuzzy as fuzz
from sklearn import datasets
iris = datasets.load_iris()
X=iris.data
y = iris.target

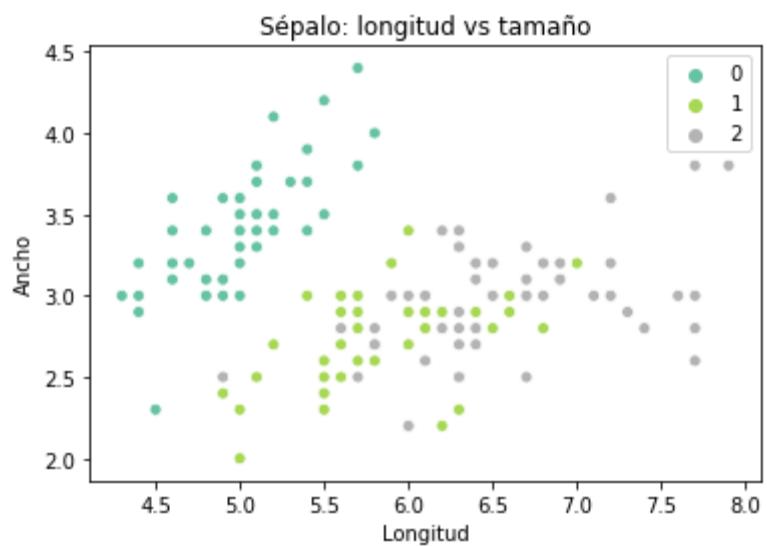
# Gráfico de dispersión Representación de los cluster
g=sns.scatterplot(X[:, 0], X[:, 1], hue=y, palette="Set2")
plt.title('Sépalo: longitud vs tamaño'), plt.xlabel('Longitud'), plt.ylabel('Ancho')
plt.show()

```

```

# Nota: se explican el ID de los data de IRIS cargada desde scikit-learn
# Setosa: 0
# Versicolor: 1
# Virgínica: 2

```



Este paquete skfuzzy (scikit-fuzzy) implementa muchas herramientas y funciones útiles para computación y proyectos de lógica difusa. En la siguiente dirección se definen todas sus funciones: <https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.defuzz>

Podemos obtener el nombre de todas las funciones de las que dispone esta librería con el comando dir:

dir(fuzz)

```
'addval',
'arglcut',
'cartadd',
'cartprod',
'centroid',
'classic_relation',
'cluster',
'cmeans',
'cmeans_predict',
'continuous_to_discrete',
'contrast',
'control',
'data_dir',
'dcentroid',
'defocus_local_means',
'defuzz',
'defuzzify',
'divval',
'doctest',
'doctest_verbose',
'dsigmf',
'dsw_add',
'dsw_div',
'dsw_mult',
'dsw_sub',
'filters',
'fire1d',
'fire2d',
'fuzzy_add',
'fuzzy_and',
'fuzzy_compare',
'fuzzy_div',
'fuzzy_min',
'fuzzy_mult',
'fuzzy_not',
'fuzzy_or',
'fuzzy_similarity',
'fuzzy_sub',
'fuzzymath',
'gauss2mf',
'gaussmf',
'gbellmf',
'image',
'inner_product',
'interp10',
'interp_membership',
'interp_universe',
'intervals',
'lambda_cut',
'lambda_cut_boundaries',
'lambda_cut_series',
'maxmin_composition',
'maxprod_composition',
'membership',
'modus_ponens',
'multval',
'nmse',
'outer_product',
'pad',
'partial_dmf',
'piecemf',
'pimf',
'pkg_dir',
'psigmf',
'relation_min',
'relation_product',
'scaleval',
'sigmf',
'sigmoid',
'smf',
'subval',
'test',
'test_verbose',
'trapmf',
'trimf',
'view_as_blocks',
'view_as_windows',
'zmf']
```

Algoritmo Fuzzy C-Means

```
# Los datos de entrada se necesitan en un array 2D -> para cambiar el tamaño de los datos se usa
la función "reshape"
```

```
X2=np.reshape(X.T, (X.shape[1],X.shape[0])) # shape[1] -> número de variables (4), shape[0] ->
número de instancias (150)
```

```
# Aplicación del modelo -> cmeans devuelve una tupla
```

```
cntr, u, u0, d, jm, p, fpc, =fuzz.cluster.cmeans(X2, c=3, m=2, error=0.005, maxiter=1000, init=None, seed=111)
```

Nota: cntr es el centro de los clusters, u es el grado de membresía, u0 la matriz inicial de membresía, etc.

```
# Centro de los grupos
```

```
pd.DataFrame(cntr, columns=['Longitud Sépalo', 'Ancho Sépalo', 'Longitud Pétalo', 'Ancho Pétalo'], index=['Cluster 1', 'Cluster 2', 'Cluster 3'])
```

	Longitud Sépalo	Ancho Sépalo	Longitud Pétalo	Ancho Pétalo
Cluster 1	5.003965	3.414116	1.482771	0.253526
Cluster 2	5.888382	2.760871	4.363147	1.396900
Cluster 3	6.774346	3.052188	5.645952	2.053226

```
# Grados de membresía -> probabilidad de pertenencia al cluster
```

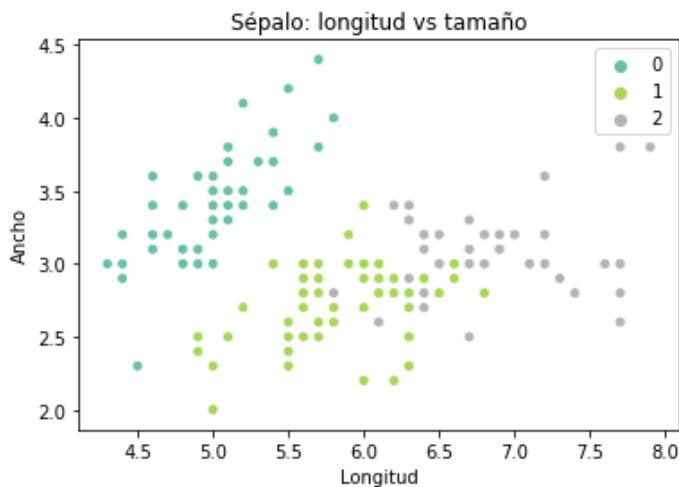
```
prob_membership=pd.DataFrame(u.T, columns=['Cluster 1', 'Cluster 2', 'Cluster 3'])
```

```
prob_membership.head()
```

	Cluster 1	Cluster 2	Cluster 3	
0		0.996624	0.002304	0.001072
1		0.975838	0.016660	0.007502
2		0.979816	0.013767	0.006417
3		0.967408	0.022479	0.010112
4		0.994470	0.003762	0.001768

```
# Gráfico de dispersión - Representación de los cluster creados
```

```
g=sns.scatterplot(X[:, 0], X[:, 1], hue=np.argmax(u.T, axis=1), palette="Set2") #argmax trabaja con arrays  
plt.title('Sépalo: longitud vs tamaño'), plt.xlabel('Longitud'), plt.ylabel('Ancho')  
plt.show()
```



Ejemplo Ejercicio

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl
```

Se definen las variables de entrada (antecedentes) y la variable de salida (consecuente)

```
comida=ctrl.Antecedent(np.arange(0, 11, 1), 'comida')
servicio=ctrl.Antecedent(np.arange(0, 11, 1), 'servicio')
propina=ctrl.Consequent(np.arange(0, 21, 1), 'propina')
```

Para ver la escala de los valores de entrada y salida -> comida.universe

dir(comida) para ver el resto de elementos del objeto

Se definen las funciones de membresía para las variables de entrada (antecedentes)

Como cada variable de entrada dispone de 3 categorías se definen 6 funciones de membresía
(2x3)

Se utilizan funciones de membresía del tipo triangular

Parámetros de la función (máximo,)

para la comida

```
comida['pobre']=fuzz.trimf(comida.universe, [0, 0, 5])
comida['aceptable']=fuzz.trimf(comida.universe, [0, 5, 10])
comida['deliciosa']=fuzz.trimf(comida.universe, [5, 10, 10])
```

para el servicio

```
servicio['malo']=fuzz.trimf(servicio.universe, [0, 0, 5])
servicio['normal']=fuzz.trimf(servicio.universe, [0, 5, 10])
servicio['agradable']=fuzz.trimf(servicio.universe, [5, 10, 10])
#para la propina
propina['baja']=fuzz.trimf(propina.universe, [0, 0, 5])
propina['media']=fuzz.trimf(propina.universe, [0, 10, 20])
propina['alta']=fuzz.trimf(propina.universe, [10, 20, 20])
```

Definición del Sistema de Reglas y Sistema de Control

Reglas

```
regla1=ctrl.Rule(comida['pobre'] | servicio['malo'], propina['baja'])
regla2=ctrl.Rule(servicio['normal'], propina['media'])
regla3=ctrl.Rule(comida['deliciosa'] | servicio['agradable'], propina['alta'])
```

Sistema de control

```
union_reglas=[regla1, regla2, regla3]
control_reglas=ctrl.ControlSystem(union_reglas)
```

```
# dir(control_reglas) para ver los elementos del objeto
```

```
# Puesta en práctica del Sistema de Control elaborado
```

```
sc_propina=ctrl.ControlSystemSimulation(control_reglas)
```

```
# Datos inputs - puntuaciones sobre la calidad de la comida y del servicio
```

```
sc_propina.input['comida']=6
sc_propina.input['servicio']=8.5
sc_propina.compute() # ejecución del sistema de control
```

```
# dir(sc_propina) para ver los elementos del objeto
```

```
# Resultado final (representación gráfica y solución numérica)
```

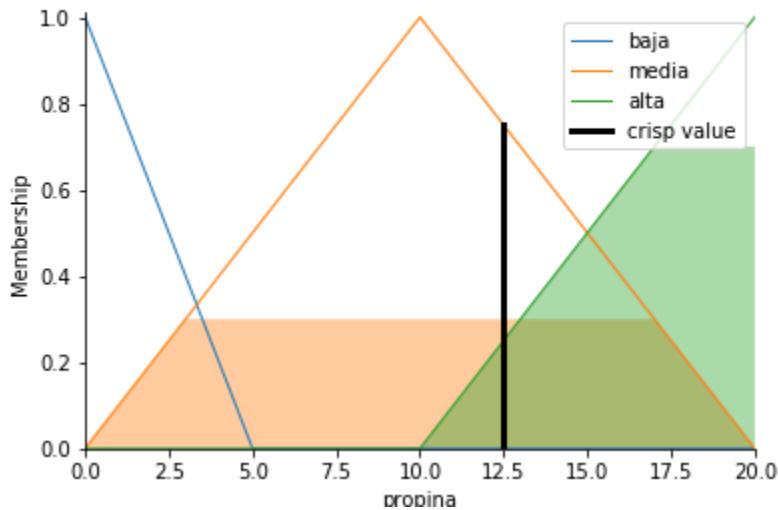
```
# gráfico explicativo
```

```
propina.view(sim=sc_propina)
plt.legend(loc='upper right')
```

```
# Resultado numérico
```

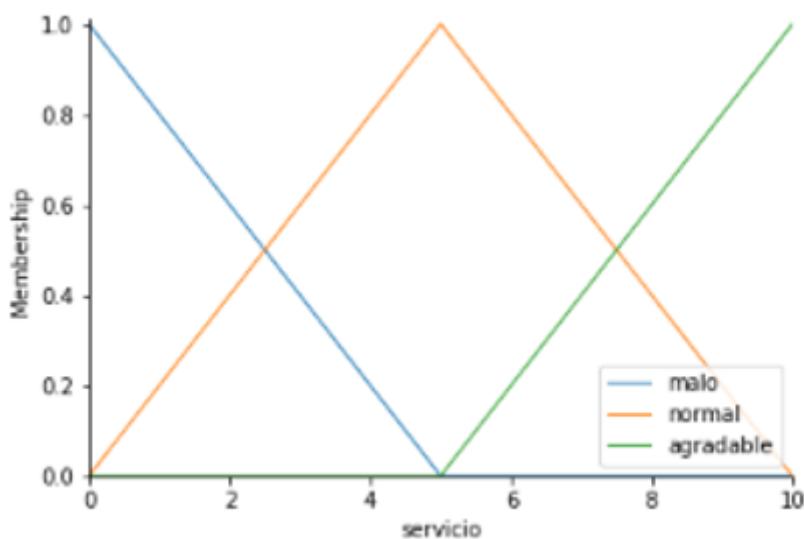
```
cantidad=sc_propina.output['propina']
print('Cantidad de propina:', round(cantidad,2))
```

Cantidad de propina: 12.49



```
# Representación de las funciones de membresía y regla (ejemplo)
```

```
servicio.view()
plt.legend(loc='lower right')
regla1.view()
plt.show()
```



7. Deep Learning

7.1. Introducción.

El deep learning o aprendizaje profundo es un concepto amplio, un término de moda que atiende a una realidad que cada día está más presente entre nosotros, y que no tiene una única definición que se pueda considerar veraz, aunque podemos generalizar la definición afirmando que surge de la idea de imitar el cerebro a partir de la utilización de hardware y software, para crear inteligencia artificial. Este intento de imitar al cerebro se ha plasmado en lo que se conoce como redes neuronales artificiales (RNA) que están dotadas de una capacidad de abstracción jerárquica: representan los datos de entrada en varios “niveles”, a través de arquitecturas en varias capas, cada capa aprende patrones más complejos según la profundidad de ésta, para conseguir información útil para el aprendizaje.

Podemos afirmar que el principio general del funcionamiento de una arquitectura profunda es guiar el entrenamiento de las capas intermedias utilizando aprendizaje no supervisado.

Algunos de los prototipos de arquitecturas y/o algoritmos desarrollados se basan en otras arquitecturas “más simples”, modificándolas y obteniendo una arquitectura “más profunda”, lo que quiere decir que en sus capas y neuronas se separan las características de un conjunto de datos de entrada de una forma jerarquizada.

Otra definición dada del aprendizaje profundo es que es un subconjunto de los algoritmos usados en machine learning, y se caracterizan por tener una arquitectura en capas donde cada capa aprende patrones más complejos según la profundidad de ésta. En este sentido, existen ya muchos prototipos de redes neuronales que tienen esta característica, por ejemplo, las ya muy conocidas redes convolucionales o las redes recurrentes. El Perceptron Multicapa de una sola capa explicado en el módulo 5 no es considerado un método de aprendizaje profundo.

El Deep Learning es una técnica reciente donde aún se sigue investigando y desarrollándose y que surgió a partir del año 2006, año en el que se considera fue retomada su investigación. El principal problema por el que se había ralentizado su progreso es que el entrenamiento basado en el método del gradiente para redes neuronales profundas supervisadas, según muchos investigadores, se estancaba en lo que se llama mínimo local aparente, lo que significa que una

red con más capas ocultas, es decir una red neuronal más profunda, obtiene peores resultados que una red con sólo una capa oculta.

Los investigadores abordaron este tema planteado a través de diferentes ópticas y se descubre que se obtienen resultados más precisos si cada capa de la red es entrenada a través de un algoritmo de preentrenamiento no supervisado (el método del gradiente requiere que sea supervisado) Para el entrenamiento de las capas, diversos autores sugieren utilizar autoencoders o máquinas de Boltzmann y, una vez se hayan preentrenado las diferentes capas, se puede aplicar un criterio supervisado.

En estos últimos años se ha producido un rápido crecimiento en la cantidad de arquitecturas y/o algoritmos de entrenamiento en una RNA, incluso, variantes de ellos, las cuales siguen el concepto planteado anteriormente.

7.2. Revisión de las Redes Neuronales

Vamos a hacer una revisión de las redes neuronales para posteriormente poder abordar los diferentes tipos de redes neuronales que se utilizan en Deep Learning. Algunos de los avances más recientes en varios de los diferentes componentes que forman parte de las redes neuronales están recopilados en (Gu et al. 2017)

Las redes neuronales artificiales tienen sus orígenes en el Perceptrón, que fue el modelo creado por Frank Rosenblatt en 1957 y basado en los trabajos que previamente habían realizado Warren McCullon (neurofisiólogo) y Walter Pitts (matemático).

El Perceptrón está construido por una neurona artificial cuyas entradas y salida pueden ser datos numéricos, no como pasaba con la neurona de McCulloch y Pitts (eran sólo datos lógicos). Las neuronas pueden tener pesos y además se le aplica una función de activación Sigmoid (a diferencia de la usada anteriormente al Paso binario).

En esta neurona nos encontramos que se realizan los siguientes cálculos:

$$z = \sum_{i=1}^n w_i x_i + b_i \quad [116]$$

$$\hat{y} = \delta(z) \quad [117]$$

donde x_i representan los datos numéricos de entrada, w_i son los pesos, b_i es el sesgo (bias), δ es la función de activación y finalmente \hat{y} es el dato de salida.

El modelo de perceptrón es el más simple, en el que hay una sola capa oculta con una única neurona.

El siguiente paso nos lleva al Perceptrón Multicapa donde ya pasamos a tener más de una capa oculta, y además podemos tener múltiples neuronas en cada capa oculta.

Cuando todas las neuronas de una capa están interconectadas con todas las de la siguiente capa estamos ante una red neuronal densamente conectada. A lo largo de las siguientes secciones nos encontraremos con redes en las que no todas las neuronas de una capa se conectan con todas de la siguiente.

Veamos como describiríamos ahora los resultados de las capas

$$z_j^{(l)} = \sum_{i=1}^{n_j} w_{ij}^{(l)} a_i^{(l-1)} + b_i^{(l)} \quad [118]$$

$$a_j^{(l)} = \delta^{(l)}(z_j^{(l)}) \quad [119]$$

donde $a_i^{(l-1)}$ representan los datos de la neurona i en la capa $l - 1$ (siendo $a_i^{(0)} = x_i$ los valores de entrada), $w_{ij}^{(l)}$ son los pesos en la capa l , $b_i^{(l)}$ es el sesgo (bias) en la capa l , $\delta^{(l)}$ es la función de activación en la capa l (puede que cada capa tenga una función de activación diferente), n_j es el número de neurona de la capa anterior que conectan con la j y finalmente $a_j^{(l)}$ es el dato de salida de la capa l . Es decir, en cada capa para calcular el nuevo valor necesitamos usar los valores de la capa anterior.

Aplicaciones de las Redes Neuronales

Cada día las redes neuronales están más presentes en diferentes campos y ayudan a resolver una gran variedad de problemas. Podríamos pensar que de forma más básica una red neuronal nos puede ayudar a resolver problemas de regresión y clasificación, es decir, podríamos considerarlo como otro modelo más de los existentes que a partir de unos datos de entrada somos capaces de obtener o un dato numérico (o varios) para hacer una regresión (calcular el precio de una vivienda en función de diferentes valores de la misma) o que somos capaces de conseguir que en función de los datos de entrada nos deje clasificada una muestra (decidir si conceder o no una hipoteca en función de diferentes datos del cliente).

Si los datos de entrada son imágenes podríamos estar usando las redes neuronales como una forma de identificar esa imagen:

- Identificando que tipo de animal es
- Identificando que señal de tráfico es
- Identificando que tipo de fruta es
- Identificando que una imagen es de exterior o interior de una casa
- Identificando que es una cara de una persona
- Identificando que una imagen radiográfica represente un tumor maligno
- Identificando que haya texto en una imagen

Luego podríamos pasar a revolver problemas más complejos combinando las capacidades anteriores:

- Detectar los diferentes objetos y personas que se encuentran en una imagen
- Etiquetado de escenas (aula con alumnos, partido de fútbol, etc...)

Después podríamos dar el paso al video que lo podríamos considerar como una secuencia de imágenes:

- Contar el número de personas que entran y salen de una habitación
- Reconocer que es una carretera
- Identificar las señales de tráfico
- Detectar si alguien lleva un arma
- Seguimiento de objetos
- Detección de estado/actitud de una persona
- Reconocimiento de acciones (interpretar lenguaje de signos, interpretar lenguaje de banderas)
- Vehículos inteligentes

Si los datos de entrada son **secuencias de texto**

- Sistemas de traducción
- Chatbots (resolución de preguntas a usuarios)
- Conversión de texto a audio

Si los datos de entrada son **audios**

- Sistemas de traducción
- Altavoces inteligentes
- Conversión de audio a texto

A continuación, pasamos a revisar diferentes elementos de las redes neuronales que suelen ser comunes a todos los tipos de redes neuronales.

Datos

Cuando se trabaja con redes neuronales necesitamos representar los valores de las **variables de entrada** en forma **numérica**. En una red neuronal todos los datos son siempre numéricos. Esto significa que todas aquellas variables que sean categóricas necesitamos convertirlas en numéricas.

Además, es muy conveniente normalizar los datos para poder trabajar con valores entre 0 y 1, que van a ayudar a que sea más fácil que se pueda converger a la solución.

Es importante que los datos seán números en coma flotante, sobre todo si se van a trabajar con **GPUs** (Graphics Process Units), ya que permitirán hacer un mejor uso de los multiples cores que les permiten operar en coma flotante de forma paralela. Actualmente, hay toda una serie de mejoras en las GPUs que permite aumentar el rendimiento de las redes neuronales como son el uso de operaciones en **FP16** (Floating Point de 16 bits en lugar de 32) de forma que pueden hacer dos operaciones de forma simultánea (el formato estándar es FP32) y además con la reducción de memoria (punto muy importante) al meter en los 32 bits 2 datos en lugar de sólo uno. También se han añadido técnicas de **Mixed Precision** (Narang et al. [2018](#)), **los Tensor Cores** (para las gráficas de NVIDIA) son otra de las mejoras que se han ido incorporando a la GPUs y que permiten acelerar los procesos tanto de entrenamiento como de predicción con las redes neuronales.

El primer objetivo será **convertir** las variables categóricas en **variables numéricas**, de forma que el AE pueda trabajar con ellas.

Para realizar la conversión de categórica a numérica básicamente tenemos dos métodos para realizarlo:

- Codificación one-hot.
- Codificación entera.

La codificación one-hot consiste en crear tantas variables como categorías tenga la variable, de forma que se asigna el valor 1 si tiene esa categoría y el 0 si no la tiene.

La codificación entera lo que hace es codificar con un número cada categoría. Realmente esta asignación no tiene ninguna interpretación numérica ya que en general las categorías no tienen porque representar un orden al que asociarlas.

Normalmente se trabaja con **codificación one-hot** para representar los datos categóricos de forma que será necesario preprocesar los datos de partida para realizar esta conversión, creando tantas variables como categorías haya por cada variable.

Si nosotros tenemos nuestra muestra de datos de que tiene n variables $x = \{x_1, x_2, x_3, \dots, x_{n-2}, x_{n-1}, x_n\}$ de forma que $x_n - 2$, $x_n - 1$ y x_n son variables categóricas que tienen k, l, m número de categorías respectivamente, tendremos finalmente las siguientes variables sólo numéricas:

$$x = \{x_1, x_2, x_3, \dots, x_{(n-2)_1}, \dots x_{(n-2)_k}, x_{(n-1)_1}, \dots, x_{(n-1)_l}, x_{n_1}, \dots x_{n_m}\}$$

De esta forma, se aumentarán el número de variables con las que vamos a trabajar en función de las categorías que tengan las variables categóricas.

Normalmente nos encontramos que en una red neuronal las **variables de salida** son:

- un número (**regresión**)
- una serie de números (**regresión múltiple**)
- un dato binario (**clasificación binaria**)
- una serie de datos binarios que representa una categoría de varias (**clasificación múltiple**)

Arquitectura de red

Para la construcción de una red neuronal necesitamos definir la arquitectura de esa red. Esta arquitectura, si estamos pensando en una **red neuronal densamente conectada**, estará definida por la cantidad de **capas ocultas** y el número de **neuronas** que tenemos en cada capa. Más adelante veremos que dependiendo del tipo de red neuronal podrá haber otro tipo de elementos en estas capas.

Función de coste y pérdida

Otro de los elementos clave que tenemos que tener en cuenta a la hora de usar nuestra red neuronal son las **funciones de pérdida y funciones de coste (objetivo)**.

La **función de pérdida** va a ser la función que nos dice cómo de diferente es el resultado del dato que nosotros queríamos conseguir respecto al dato original. Normalmente se suelen usar diferentes tipos de funciones de pérdida en función del tipo de resultado con el que se vaya a trabajar.

La **función de coste** es la función que vamos a tener que **optimizar** para conseguir el mínimo valor posible, y que recoge el valor de la función de pérdida para toda la muestra.

Tanto las funciones de pérdida como las funciones de coste, son funciones que devuelven valores de \mathbb{R} .

Si tenemos un problema de **regresión** en el que tenemos que predecir un valor o varios valores numéricos, algunas de las funciones a usar son:

- **Error medio cuadrático (L_2^2)**

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = ||\hat{y} - y||^2 = \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad [120]$$

donde \hat{y} y y son vectores de tamaño n , y es el valor real e \hat{y} es el valor predicho

- **Error medio absoluto (L_1)**

$$\mathcal{L}_{\text{MAE}}(y, \hat{y}) = |\hat{y} - y| = \sum_{i=0}^n |\hat{y}_i - y_i| \quad [121]$$

donde \hat{y} y y son vectores de tamaño n , y es el valor real e \hat{y} es el valor predicho

Para los problemas de **clasificación**:

- **Binary Crossentropy** (Sólo hay dos clases)

$$\mathcal{L}_{\text{CRE}}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \quad [122]$$

y es el valor real e \hat{y} es el valor predicho

- **Categorical Crosentropy** (Múltiples clases representadas como one-hot)

$$\mathcal{L}_{\text{CAE}}(y_c, \hat{y}_c) = -\sum_{c=1}^k y_c \log(\hat{y}_c) \quad [123]$$

y_c es el valor real para la clase c e \hat{y}_c es el valor predicho para la clase c

- **Sparse Categorical Crossentropy** (Múltiples clases representadas como un entero)

$$\mathcal{L}_{\text{SCAE}}(y_c, \hat{y}_c) = -\sum_{c=1}^k y_c \log(\hat{y}_c) \quad [124]$$

y_c es el valor real para la clase c e \hat{y}_c es el valor predicho para la clase c

- **Kullback-Leibler Divergence**

Esta función se usa para calcular la diferencia entre dos distribuciones de probabilidad y se usa por ejemplo en algunas redes como **Variational Autoencoders** (Doersch [2016](#)) o

Modelos GAN (Generative Adversarial Networks)

$$\mathcal{D}_{\text{KL}}(p \parallel q) = -H(p(x)) - E_p[\log q(x)] \quad [125]$$

$$= \sum_x p(x) \log p(x) - \sum_x p(x) \log q(x) = \sum_x p(x) \log \frac{p(x)}{q(x)} \quad [126]$$

$$\mathcal{L}_{\text{vae}}(y, \hat{y}) = E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - \mathcal{D}_{\text{KL}}(q_\phi(z|x) \parallel p(z)) \quad [127]$$

- **Hinge Loss**

$$\mathcal{L}_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y * \hat{y}) \quad [128]$$

Las correspondientes **funciones de coste** que se usarían, estarían asociadas a todas las **muestras** que se estén entrenando o sus correpondientes batch, así como posibles términos asociados a la regularización para evitar el sobreajuste del entrenamiento. Es decir, la función de pérdida se calcula para cada muestra, y la función de coste es la media de todas las muestras.

Por ejemplo, para el **Error medio cuadrático (L_2)** tendríamos el siguiente valor:

$$\mathcal{J}_{\text{MSE}}(y, \hat{y}) = \frac{1}{m} \sum_{j=1}^m \mathcal{L}_{\text{MSE}}(y_j, \hat{y}_j) = \frac{1}{m} \sum_{j=1}^m \| \hat{y}_j - y_j \|^2 = \frac{1}{n} \sum_{j=1}^m \sum_{i=1}^n (\hat{y}_{ji} - y_{ji})^2 \quad [129]$$

Optimizador

El **Descenso del gradiente** es la versión más básica de los algoritmos que permiten el aprendizaje en la red neuronal haciendo el proceso de **backpropagation** (propagación hacia atrás). A continuación veremos una breve explicación del algoritmo así como algunas variantes del mismo recogidas en (Ruder 2017)

Recordamos que el descenso del gradiente nos permitirá actualizar los parámetros de la red neuronal cada vez que demos una pasada hacia delante con todos los datos de entrada, volviendo con una pasada hacia atrás.

$$W_t = W_{t-1} - \alpha \nabla_w \mathcal{J}(w) \quad [130]$$

donde \mathcal{J} es la **función de coste**, α es el parámetro de **ratio de aprendizaje** que permite definir como de grandes se quiere que sean los pasos en el aprendizaje.

Cuando lo que hacemos es actualizar los parámetros para cada pasada hacia delante de una sola muestra, estaremos ante lo que llamamos **Stochastic Gradient Descent (SGD)**. En este proceso convergerá en menos iteraciones, aunque puede tener alta varianza en los parámetros.

$$W_t = W_{t-1} - \alpha \nabla_w \mathcal{J}(w, x(i), y(i)) \quad [131]$$

donde $x(i)$ e $y(i)$ son los valores en la pasada de la muestra i .

Podemos buscar un punto intermedio que sería cuando trabajamos por lotes y cogemos un bloque de datos de la muestra, les aplicamos la pasada hacia delante y aprendemos los parámetros para ese bloque. En este caso lo llamaremos **Mini-batch Gradient Descent**

$$W_t = W_{t-1} - \alpha \nabla_w \mathcal{J}(w, B(i)) \quad [132]$$

donde $B(i)$ son los valores de ese batch i .

En general a estos métodos nos referiremos a ellos como **SGD**.

Sobre este algoritmo base se han hecho ciertas mejoras como:

Learning rate decay

Podemos definir un valor de decenso del ratio de aprendizaje, de forma que normalmente al inicio de las iteraciones de la red neuronal los pasos serán más grandes, pero conforme nos acercamos a la solución optima deberemos dar pasos más pequeños para ajustarnos mejor.

$$W_t = w_{t-1} - \alpha_t \nabla_w J(w_{t-1}) \quad [133]$$

donde ahora α_t se irá reduciendo en función del valor del **decay**

Momentum

El **momentum** se introdujo para suavizar la convergencia y reducir la alta varianza de **SGD**.

$$V_t = \gamma v_{t-1} + \alpha \nabla_w J(w_{t-1}, x, y) \quad [134]$$

$$W_t = w_{t-1} - v_t \quad [135]$$

donde v_t es lo que se llama el **vector velocidad** con la dirección correcta.

NAG (Nesterov Accelerated Gradient)

Ahora daremos un paso más con el **NAG**, calculando la función de coste junto con el vector velocidad.

$$V_t = \gamma v_{t-1} + \alpha \nabla_w J(w_{t-1} - \gamma v_{t-1}, x, y) \quad [136]$$

$$W_t = w_{t-1} - v_t \quad [137]$$

donde ahora vemos que la función de coste se calcula usando los parámetros de w_{t-1} sumado a γv_{t-1}

Veamos algunos algoritmos de optimización más que, aunque provienen del **SGD**, se consideran independientes a la hora de usarlos y no como parámetros extras del **SGD**.

Adagrad (Adaptive Gradient)

Esta variante del algoritmo lo que hace es adaptar el ratio de aprendizaje para cada uno de los pesos en lugar de que sea global para todos.

$$W_{t,i} = w_{t-1,i} - \frac{\alpha}{\sqrt{G_{t-1,i,i} + \epsilon}} \nabla_{w_{t-1}} J(w_{t-1}, x, y) \quad [138]$$

donde tenemos que $G_t \in \mathbb{R}^{d \times d}$ es una matriz diagonal donde cada elemento i, i es la suma de los cuadrados de los gradientes en el paso $t - 1$, y ϵ es un término de suavizado para evitar divisiones por 0.

RMSEProp (Root Mean Square Propagation)

En este caso tenemos una variación del Adagrad en el que intenta reducir su agresividad reduciendo monotonamente el ratio de aprendizaje. En lugar de usar el gradiente acumulado desde el principio de la ejecución, se restringe a una ventana de tamaño fijo para los últimos n gradientes calculando su media. Así calcularemos primero la media en ejecución de los cuadros de los gradientes como:

$$E[g^2]_{t-1} = \gamma E[g^2]_{t-2} + (1 - \gamma) g_{t-1}^2 \quad [139]$$

y luego ya pasaremos a usar este valor en la actualización

$$w_{t,i} = w_{t-1,i} - \frac{\alpha}{\sqrt{E[g^2]_{t-1} + \epsilon}} \nabla_{w_{t-1}} J(w_{t-1}, x, y) \quad [140]$$

AdaDelta

Aunque se desarrollaron de forma simultánea el **AdaDelta** y el **RMSProp** son muy parecidos en su primer paso inicial, llegando el de AdaDelta un poco más lejos en su desarrollo.

$$E[g^2]_{t-1} = \gamma E[g^2]_{t-2} + (1 - \gamma) g_{t-1}^2 \quad [141]$$

y luego ya pasaremos a usar este valor en la actualización

$$w_{t,i} = w_{t-1,i} - \frac{\alpha}{\sqrt{E[g^2]_{t-1} + \epsilon}} \nabla_{w_{t-1}} J(w_{t-1}, x, y) \quad [142]$$

$$\Delta w_t = - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad [143]$$

Adam (Adaptive Moment Estimation)

$$G_t = \nabla_{w_t} J(w_t) \quad [144]$$

$$M_{t-1} = \beta_1 m_{t-2} + (1 - \beta_1) g_{t-1} \quad [145]$$

$$V_{t-1} = \beta_2 v_{t-2} + (1 - \beta_2) g_{t-1}^2 \quad [146]$$

donde m_{t-1} y v_{t-1} son estimaciones del primer y segundo momento de los gradientes respectivamente, y β_1 y β_2 parámetros a asignar.

$$\hat{M}_{t-1} = \frac{m_{t-1}}{1-\beta_1^{t-1}} \quad [147]$$

$$\hat{V}_{t-1} = \frac{v_{t-1}}{1-\beta_2^{t-1}} \quad [148]$$

$$W_t = w_{t-1} - \frac{\alpha}{\sqrt{\hat{V}_{t-1} + \epsilon}} \hat{M}_{t-1} \quad [149]$$

Adamax

$$G_t = \nabla_{w_t} J(w_t) \quad [150]$$

$$M_{t-1} = \beta_1 m_{t-2} + (1 - \beta_1) g_{t-1} \quad [151]$$

$$V_{t-1} = \beta_2 v_{t-2} + (1 - \beta_2) g_{t-1}^2 \quad [152]$$

$$U_{t-1} = \max(\beta_2 \cdot v_{t-1}, |g_t|) \quad [153]$$

donde m_{t-1} y v_{t-1} son estimaciones del primer y segundo momento de los gradientes respectivamente, y β_1 y β_2 parámetros a asignar.

$$\hat{M}_{t-1} = \frac{m_{t-1}}{1-\beta_1^{t-1}} \quad [154]$$

$$W_t = w_{t-1} - \frac{\alpha}{u_{t-1}} \hat{M}_{t-1} \quad [155]$$

Nadam (Nesterov-accelerated Adaptive Moment Estimation)

Combina Adam y NAG.

$$G_t = \nabla_{w_t} J(w_t) \quad [156]$$

$$M_{t-1} = \gamma m_{t-2} + \alpha g_{t-1} \quad [157]$$

$$W_t = w_{t-1} - m_{t-1} \quad [158]$$

Función de activación

Las funciones de activación dentro de una red neuronal son uno de los elementos clave en el diseño de la misma. Cada tipo de función de activación podrá ayudar a la convergencia de forma más o menos rápida en función del tipo de problema que se plantee.

En un **AE** las funciones de **activación** en las capas ocultas van a conseguir establecer las restricciones **no lineales** al pasar de una capa a la siguiente, normalmente se evita usar la función de activación lineal en las capas intermedias ya que queremos conseguir transformaciones no lineales.

A continuación, exponemos las principales **funciones de activación** que mejores resultados dan en las **capas ocultas**:

- **Paso binario** (Usado por los primeros modelos de neuronas)

$$F(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \quad [159]$$

- **Identidad**

$$F(x) = x \quad [160]$$

- **Sigmoid (Logística)**

$$F(x) = \frac{1}{1+e^{-x}} \quad [161]$$

- **Tangente Hiperbólica (Tanh)**

$$F(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad [162]$$

- **Softmax**

$$F(x_i) = \frac{e^{x_i}}{\sum_{j=0}^k e^{x_j}} \quad [163]$$

donde $i = 0, \dots, k$ con x_i cada uno de los elementos del vector

- **ReLU (Rectified Linear Unit)**

$$F(x) = \max(0, x) \quad [164]$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

- **LReLU (Leaky Rectified Linear Unit)**

$$F(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad [165]$$

- **PReLU (Parametric Rectified Linear Unit)**

$$F(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad [166]$$

- **RReLU (Randomized Rectified Linear Unit)**

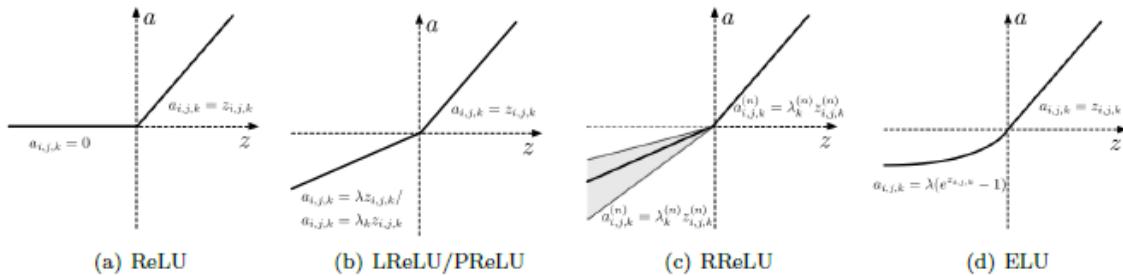
$$F(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad [167]$$

*La diferencia entre LReLU, PReLU y RReLU es que en LReLU el parámetro es uno que se asigna fijo, en el caso de PReLU el parámetro también se aprende durante el entrenamiento y finalmente en RReLU es un parámetro con valores entre 0 y 1, que se obtiene de un muestreo en una distribución normal. Se puede profundizar en este grupo de funciones de activación en ([Xu et al. 2015](#))

- **ELU (Exponential Linear Unit)**

$$F(\alpha, x) = \begin{cases} \alpha(e^{x-1}) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad [168]$$

FIGURA nº 64: COMPARACIÓN ENTRE LAS FUNCIONES ReLU, LReLU/PReLU, RReLU y ELU



FUENTE: Jiuxiang, G. et al (2019)

Función de activación en salida

En la capa de salida tenemos que tener en cuenta cual es el tipo de datos final que queremos

obtener, y en función de eso elegiremos cual es la **función de activación de salida** que usaremos.

Normalmente las funciones de activación que se usarán en la **última capa** serán:

- **Lineal** con una unidad, para regresión de un solo dato numérico

$$F(x) = x \quad [169]$$

donde x es un valor escalar.

- **Lineal** con multiples unidades, para regresión de varios datos numéricos

$$F(x) = x$$

[170]

donde x es un vector.

- **Sigmoid** para clasificación binaria

$$F(x) = \frac{1}{1+e^{-x}} \quad [171]$$

- **Softmax** para clasificación múltiple

$$F(x_i) = \frac{e^{x_i}}{\sum_{j=0}^k e^{x_j}} \quad [172]$$

donde $i = 0, \dots, k$ con x_i cada uno de los elementos del vector

Regularización

Las técnicas de regularización nos permiten conseguir mejorar los problemas que tengamos por sobreajuste en el entrenamiento de nuestra red neuronal.

A continuación, vemos algunas de las técnicas de regularización existentes en la actualidad:

- **Norma LP**

Básicamente estos métodos tratan de hacer que los pesos de las neuronas tengan valores muy pequeños consiguiendo una distribución de pesos más regular. Esto lo consiguen al añadir a la función de pérdida un coste asociado a tener pesos grandes en las neuronas. Este peso se puede construir o bien con la **norma L1 (proporcional al valor absoluto)** o con la **norma L2 (proporcional al cuadrado de los coeficientes de los pesos)**. En general se define la **norma LP**

$$E(w, y, \hat{y}) = \mathcal{L}(w, y, \hat{y}) + \lambda R(w) \quad [173]$$

$$R(w) = \sum_j \|w_j\|_p^p \quad [174]$$

Para los casos más habituales tendríamos la norma **L1** y **L2**.

$$R(w) = \sum_j \|w_j\|^2 \quad [175]$$

$$R(w) = \sum_j |w_j| \quad [176]$$

- **Dropout**

Una de las técnicas de regularización que más se están usando actualmente es la llamada **Dropout**, su proceso es muy sencillo y consiste en que en cada iteración de forma aleatoria se dejan de usar un porcentaje de las neuronas de esa capa, de esta forma es más difícil conseguir un sobreajuste porque las neuronas no son capaces de memorizar parte de los datos de entrada.

- **Dropconnect**

El **Dropconnect** es otra técnica que va un poco más allá del concepto de **Dropout** y en lugar de usar en cada capa de forma aleatoria una serie de neuronas, lo que se hace es que de forma aleatoria se ponen los pesos de la capa a cero. Es decir, lo que hacemos es que hay ciertos enlaces de alguna neurona de entrada con alguna de salida que no se activan.

Inicialización de pesos

Cuando empieza el entrenamiento de una red neuronal y tiene que realizar la primera pasada hacia delante de los datos, necesitamos que la red neuronal ya tenga asignados algún valor a los pesos.

Se pueden hacer inicializaciones del tipo:

- **Ceros** Todos los pesos se inicializan a 0.
- **Unos** Todos los pesos se inicializan a 1.
- **Distribución normal** Los pesos se inicializan con una distribución normal, normalmente con media 0 y una desviación alrededor de 0,05. Es decir, valores bastante cercanos al cero.
- **Distribución normal truncada** Los pesos se inicializan con una distribución normal, normalmente con media 0 y una desviación alrededor de 0,05 y además se truncan con un máximo del doble de la desviación. Los valores aun son más cercanos a cero.
- **Distribución uniforme** Los pesos se inicializan con una distribución uniforme, normalmente entre el 0 y el 1.
- **Glorot Normal (También llamada Xavier normal)** Los pesos se inicializan partiendo de una distribución normal truncada en la que la desviación es $\sqrt{\frac{2}{fanin+fanout}}$ donde *fanin* es el número de unidades de entrada y *fanout* es el número de unidades de salida. Ver (Glorot and Bengio [2010](#))

- **Glorot Uniforme (También llamada Xavier uniforme)** Los pesos se inicializan partiendo de una distribución uniforme donde los límites son $[-limit, +limit]$ donde $limit = \sqrt{\frac{6}{fanin+fanout}}$ y $fanin$ es el número de unidades de entrada y $fanout$ es el número de unidades de salida.
Ver ([Glorot and Bengio 2010](#))

Batch normalization

Hemos comentado que cuando entrenamos una red neuronal los datos de entrada deben ser todos de tipo numérico y además los normalizamos para tener valores “cercanos a cero”, teniendo una media de 0 y varianza de 1, consiguiendo uniformizar todas las variables y conseguir que la red pueda converger más fácilmente.

Cuando los datos entran a la red neuronal y se comienza a operar con ellos, se convierten en nuevos valores que han perdido esa propiedad de normalización. Lo que hacemos con la normalización por lotes (batch normalization) (Ioffe and Szegedy 2015) es que añadimos un paso extra para normalizar las salidas de las funciones de activación. Lo normal es que se aplicara la normalización con la media y la varianza de todo el bloque de entrenamiento en ese paso, pero normalmente estaremos trabajando por lotes y se calculará la media y varianza con ese lote de datos.

7.3. Principales arquitecturas de Deep Learning

Actualmente existen muchos tipos de estructuras de redes neuronales artificiales dado que logran resultados extraordinarios en muchos campos del conocimiento. Los primeros éxitos en el aprendizaje profundo se lograron a través de las investigaciones y trabajos de Geoffre Hinton (2006) que introduce las Redes de Creencia Profunda en cada capa de la red de una Máquina de Boltzmann Restringida (RBM) para la asignación inicial de los pesos sinápticos.

Hace tiempo que se está trabajando con arquitecturas como los Autoencoders, Hinton y Zemel (1994), las RBMs de Hinton y Sejnowski (1986) y las DBNs (Deep Belief Networks), Hinton et al. (2006) y otras como las redes recurrentes y convolucionales. Estas técnicas constituyen en sí mismas arquitecturas de redes neuronales, aunque también algunas de ellas, como se ha afirmado en la introducción, se están empleando para inicializar los pesos de arquitecturas profundas de redes neuronales supervisadas con conexiones hacia adelante.

Las principales arquitecturas de deep learning se resumen en la siguiente figura.

FIGURA 65. MODELOS DE REDES NEURONALES SEGÚN TIPO DE APRENDIZAJE

Tipo de aprendizaje	Modelo de Red Neuronal	Utilización
Supervisado	<i>Redes Neuronales Densamente Conectadas</i>	<i>Regresión y Clasificación</i>
	<i>Redes Convolucionales</i>	<i>Visión por computación</i>
	<i>Redes Recurrentes</i>	<i>Tratamiento de secuencias</i>
No supervisado	<i>Mapas Autoorganizativos</i>	<i>Detección de características</i>
	<i>Máquinas de Boltzmann</i>	<i>Sistema de recomendaciones</i>
	<i>Autoencoders</i>	<i>Sistema de recomendaciones</i>

Vamos a describir de forma somera las principales arquitecturas dado que posteriormente se desarrollan más ampliamente en este epígrafe o se tratan en documento adjunto que se acompaña en este módulo

Convolutional Neural Network

Tal vez los modelos más utilizados actualmente en el campo del Deep Learning sean las redes neuronales convolucionales, denominadas en inglés Convolutional Neural Networks (CNN). El objetivo de las redes CNN es aprender características de orden superior utilizando la operación de convolución. Estas estructuras de redes neuronales son especialmente eficaces para clasificar y segmentar imágenes, en general, son notablemente eficaces en tareas de visión artificial.

Las CNN son una modificación del perceptrón multicapa explicado en un módulo anterior. Este modelo es muy similar al trabajo que ejecuta el cerebro humano: las neuronas se corresponden a campos receptivos similares a como lo realizan las neuronas en la corteza visual de nuestro cerebro.

Esta arquitectura ya se utilizó en 1990 donde la empresa AT & T las aplicó para crear un modelo de lectura de cheques, desarrollándose posteriormente muchos sistemas OCR basados en CNN.

Las redes de convolución tienen una estructura de varias capas: las capas de Convolución que transforman los datos de entrada a través de una operación matemática llamada Convolución y la capa de pooling, que trata de sintetizar y condensar la información de la capa de convolución. Finalmente, se transforman los datos para aplicar una red densamente conectada que nos ofrece el resultado final en relación con el objetivo que se busca.

Estas redes neuronales artificiales se desarrollaron al abrigo de los concursos denominados ILSVRC (Large Scale Visual Recognition Challenge) donde aparecieron las principales aportaciones efectuadas en las redes convolucionales y que hoy podemos utilizar todos los investigadores. Algunos de las estructuras más novedosas y que son modelos ya preentrenados, con estructuras de capas más numerosas y que podemos integrar en nuestras aplicaciones, se denominan: LeNet-5, AlexNet, VGG, GoogLeNet y Resnet.

Autoencoder

Los Autoencoders (AE) son uno de los tipos de redes neuronales que caen dentro del ámbito del Deep Learning, en la que nos encontramos con un modelo de aprendizaje no supervisado. Ya se empezó a hablar de AE en la década de los 80 (Bourlard and Kamp 1988), aunque es en estos últimos años donde más se está trabajando con ellos.

La arquitectura de un AE es una Red Neuronal Artificial (ANN por sus siglas en inglés) que se encuentra dividida en dos partes, encoder y decoder (Charte et al. 2018), (Goodfellow, Bengio, and Courville 2016). El encoder va a ser la parte de la ANN que va codificar o comprimir los datos de entrada, y el decoder será el encargado de regenerar de nuevo los datos en la salida. Esta estructura de codificación y decodificación le llevará a tener una estructura simétrica. El AE es entrenado para ser capaz de reconstruir los datos de entrada en la capa de salida de la ANN, implementando una serie de restricciones (la reducción de elementos en las capas ocultas del encoder) que van a evitar que simplemente se copie la entrada en la salida.

Algunas de sus principales aplicaciones sobre las que se está investigando son:

- Reducción de dimensiones / Compresión de datos
- Búsqueda de imágenes
- Detección de Anomalías
- Eliminación de ruido

Redes recurrentes

En la actualidad las redes neuronales recurrentes (Recurrent Neural Networks) han logrado un puesto destacado en machine learning. Estas redes que no disponen de una estructura de capas, sino que permiten conexiones arbitrarias entre todas las neuronas, incluso creando ciclos. En esta arquitectura se permiten conexiones recurrentes lo que aumenta el número de pesos o de parámetros ajustables de la red, lo que incrementa la capacidad de representación, pero también la complejidad del aprendizaje. Las peculiaridades de esta red permiten incorporar a la red el

concepto de temporalidad, y también que la red tenga memoria, porque los números que introducimos en un momento dado en las neuronas de entrada son transformados, y continúan circulando por la red. Existen diferentes planteamientos de redes recurrentes, por ejemplo, son muy populares por sus aplicaciones las redes de Elmann y de Jordan.

En los últimos años se han popularizado las redes recurrentes denominadas Long-Short Term Memory (LSTM) que son una extensión de las redes recurrentes y su característica principal es que amplían su memoria para registrar experiencias que han ocurrido hace mucho tiempo. Normalmente contienen tres puertas que determinan si se permite o no una nueva entrada o se elimina la información que llega dado que se considera no importante. Son análogas a una función sigmoide lo que implica que van de 0 a 1, lo que permite incorporarlas al proceso de backpropagation.

También se encuentran entre las redes recurrentes, las denominadas GRU (Gated Recurrent Unit) que aparecieron en 2014 y simplifican a las LSTM: son computacionalmente menos costosas y más eficientes.

Boltzmann Machine y Restricted Boltzmann Machine

El aprendizaje de la denominada máquina de Boltzmann (BM) se realiza a través de un algoritmo estocástico que proviene de ideas basadas en la mecánica estadística. Este prototipo de red neuronal tiene una característica distintiva y es que el uso de conexiones sinápticas entre las neuronas es simétrico.

Las neuronas son de dos tipos: visibles y ocultas. Las neuronas visibles son las que interactúan y proveen una interface entre la red y el ambiente en el que operan, mientras que las neuronas actúan libremente sin interacciones con el entorno. Esta máquina dispone de dos modos de operación. El primero es la condición de anclaje donde las neuronas están fijas por los estímulos específicos que impone el ambiente. El otro modo es la condición de libertad, donde tanto las neuronas ocultas como las visibles actúan libremente sin condiciones impuestas por el medio ambiente. Las máquinas restringidas de Boltzmann (RBM) solamente toman en cuenta aquellos modelos en los que no existen conexiones del tipo visible-visible y oculta-oculta. Estas redes también asumen que los datos de entrenamiento son independientes y están idénticamente distribuidos.

Una forma de estimar los parámetros de un modelo estocástico es calculando la máxima verosimilitud. Para ello, se hace uso de los Markov Random Fiels (MRF), ya que al encontrar los parámetros que maximizan los datos de entrenamiento bajo una distribución MRF, equivale a encontrar los parámetros θ que maximizan la verosimilitud de los datos de entrenamiento, Fischer e Igel (2012). Maximizar dicha verosimilitud es el objetivo que persigue el algoritmo de entrenamiento de una RBM. A pesar de utilizar la distribución MRF, computacionalmente hablando se llega a ecuaciones inviables de implementar. Para evitar el problema anterior, las esperanzas que se obtienen de MRF pueden ser aproximadas por muestras extraídas de distribuciones basadas en las técnicas de Markov Chain Monte Carlo Techniques (MCMC). Las técnicas de MCMC utilizan un algoritmo denominado muestreo de Gibbs con el que obtenemos una secuencia de observaciones o muestras que se aproximan a partir de una distribución de verosimilitud de múltiples variables aleatorias. La idea básica del muestreo de Gibbs es actualizar cada variable posteriormente en base a su distribución condicional dado el estado de las otras variables.

Deep Belief Network

Una red Deep Belief Network tal como demostró Hinton se puede considerar como un “apilamiento de redes restringidas de Boltzmann”. Tiene una estructura jerárquica que como sabemos es una de las características del deep learning. Como en el anterior modelo, esta red también es un modelo en grafo estocástico, que aprende a extraer una representación jerárquica profunda de los datos de entrenamiento. Cada capa de la RBM extrae un nivel de abstracción de características de los datos de entrenamiento, cada vez más significativo; pero para ello, la capa siguiente necesita la información de la capa anterior lo que implica el uso de las variables latentes.

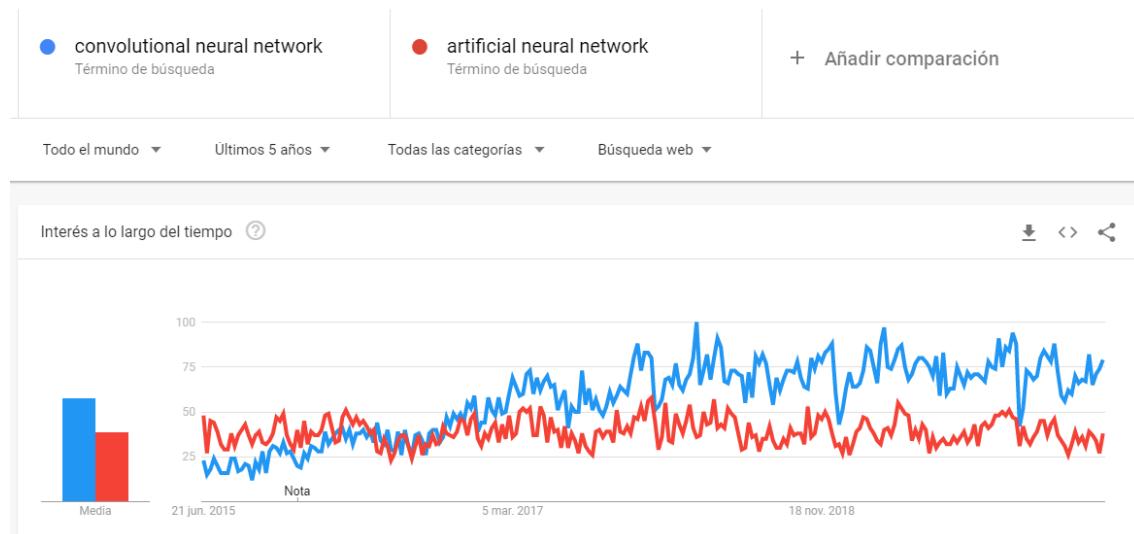
Estos modelos caracterizan la distribución conjunta h_k entre el vector de observaciones x y las capas ocultas, donde $x=h_0$, es una distribución condicional para las unidades visibles limitadas sobre las unidades ocultas que pertenecen a la RBM en el nivel k , y es la distribución conjunta oculta visible en la red RBM del nivel superior o de salida. El entrenamiento de esta red puede ser híbrido, empezando por un entrenamiento no supervisado para después aplicar un entrenamiento supervisado para un mejor y más óptimo ajuste, aunque pueden aplicarse diferentes tipos de entrenamiento, Bengio et al. (2007) y Salakhutdinov (2014) Para realizar un entrenamiento no supervisado se aplica a las redes de creencia profunda con Redes restringidas de Boltzmann el método de bloque constructor que fue presentado por Hinton (2006) y por Bengio (2007)

7.4. Redes Neuronales Convolucionales

7.4.1. Introducción

Esta arquitectura de redes de neuronas convolucionales, CNN, Convolutional Neural Networks es en la actualidad el campo de investigación más fecundo dentro de las redes neuronales artificiales de Deep learning y donde los investigadores, empresas e instituciones están dedicando más recursos e investigación. Para apoyar esta aseveración, en google trend se observa que el término convolutional neural network en relación con el concepto de artificial neural network crece y está por encima desde el año 2016. Es en este último lustro donde el Deep learning ha tomado una importancia considerable.

FIGURA 66. BÚSQUEDA DE TÉRMINOS DE REDES NEURONALES EN GOOGLE TREND



Fuente: Google Trend

En este modelo de redes convolucionales las neuronas se corresponden a campos receptivos similares a las neuronas en la corteza visual de un cerebro humano. Este tipo de redes se han mostrado muy efectivas para tareas de detección y categorización de objetos y en la clasificación y segmentación de imágenes. Por ejemplo, estas redes en la década de 1990 las aplicó AT & T para desarrollar un modelo para la lectura de cheques. También más tarde se desarrollaron muchos sistemas OCR basados en CNN.

En esta arquitectura cada neurona de una capa no recibe conexiones entrantes de todas las neuronas de la capa anterior, sino sólo de algunas. Esta estrategia favorece que una neurona se especialice en una región del conjunto de números (píxeles) de la capa anterior, lo que disminuye

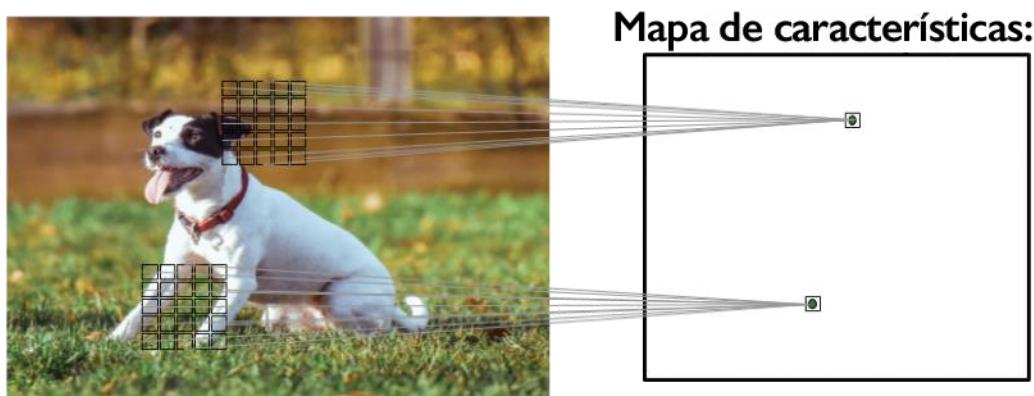
notablemente el número de pesos y de operaciones a realizar. Lo más normal es que neuronas consecutivas de una capa intermedia se especialicen en regiones solapadas de la capa anterior.

Una forma intuitiva para entender cómo trabajan estas redes neuronales es ver cómo nos representamos y vemos las imágenes. Para reconocer una cara primero tenemos que tener una imagen interna de lo que es una cara. Y a una imagen de una cara la reconocemos porque tiene nariz, boca, orejas, ojos, etc. Pero en muchas ocasiones una oreja está tapada por el pelo, es decir, los elementos de una cara se pueden ocultar de alguna manera. Antes de clasificarla, tenemos que saber la proporción y disposición y también cómo se relacionan la partes entre sí.

Para saber si las partes de la cara se encuentran en una imagen tenemos que identificar previamente líneas bordes, formas, texturas, relación de tamaño, etcétera. En una red convolucional, cada capa lo que va a ir aprendiendo son los diferentes niveles de abstracción de la imagen inicial.

Para comprender mejor el concepto anterior hemos seleccionado esta imagen de Raschka y Mirjalili (2019) donde se observa como partes del perro se transforman en neuronas del mapa de características

FIGURA 67. . CORRESPONDENCIA DE ZONAS DE LA IMAGEN Y MAPA DE CARACTERÍSTICAS



Fuente: Raschka y Mirjalili (2019)

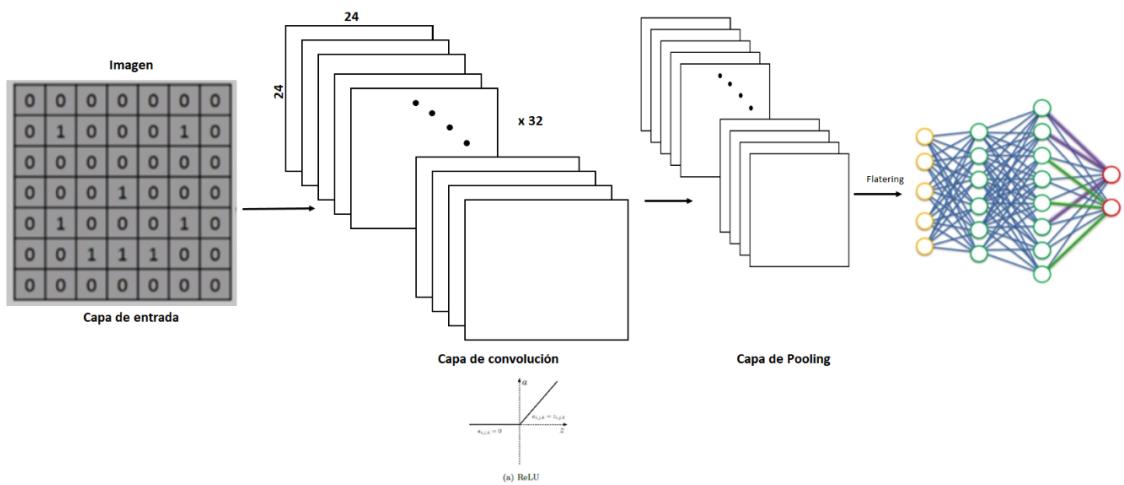
El objetivo de las redes CNN es aprender características de orden superior utilizando la operación de convolución.

Puesto que las redes neuronales convolucionales pueden aprender relaciones de entrada-salida (donde la entrada es una imagen), en la convolución, cada pixel de salida es una combinación lineal de los pixeles de entrada.

La convolución consiste en filtrar una imagen utilizando una máscara. Diferentes máscaras producen distintos resultados. Las máscaras representan las conexiones entre neuronas de capas anteriores. Estas capas aprenden progresivamente las características de orden superior de la entrada sin procesar.

Las redes neuronales convolucionales se forman usando dos tipos de capas: convolucionales y pooling. La capa de convolución transforma los datos de entrada a través de una operación matemática llamada convolución. Esta operación describe cómo fusionar dos conjuntos de información diferentes. A esta operación se le suele aplicar una función de transformación, generalmente la RELU. Después de la capa o capas de convolución se usa una capa de pooling, cuya función es resumir las respuestas de las salidas cercanas. Antes de obtener el output unimos la última capa de pooling con una red densamente conectada. Previamente se ha aplanado (Flatering) la última capa de pooling para obtener un vector de entrada a la red neural final que nos ofrecerá los resultados.

FIGURA N° 68: ARQUITECTURA DE UNA CNN



Las redes neuronales convolucionales debido a su forma de concebirse son aptas para poder aprender a clasificar todo tipo de datos donde éstos estén distribuidos de una forma continua a lo largo del mapa de entrada, y a su vez sean estadísticamente similares en cualquier lugar del mapa de entrada. Por esta razón, son especialmente eficaces para clasificar imágenes. También pueden ser aplicadas para la clasificación de series de tiempo o señales de audio.

En relación con el color y la forma de codificarse, en las redes convolucionales se realiza en tensores 3D, dos ejes para el ancho (width) y el alto (height) y el otro eje llamado de profundidad

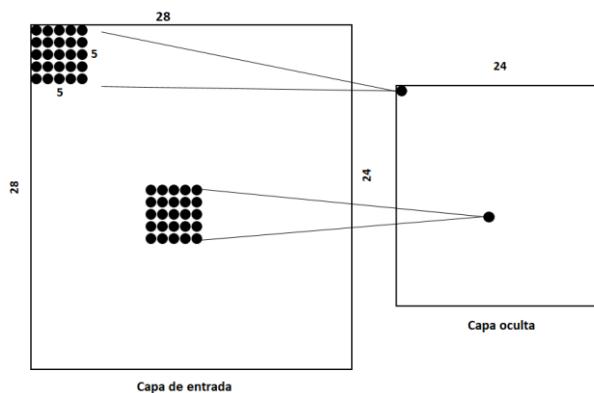
(depth) que es el canal del color con valor tres si trabajamos con imágenes de color RGB (Red, Green y Blue) rojo, verde y azul. Si disponemos de imágenes en escala de grises el valor de depth es uno. La base de datos MNIST (National Institute of Standards and Technology database) con la que trabajaremos en este epígrafe contiene imágenes de 28×28 píxeles, los valores de height y de width son ambos 28, y al ser una base de datos en blanco y negro el valor de depth es 1.

Las imágenes son matrices de píxeles que van de cero a 255 y que para la red neuronal se normalizan para que sus valores oscilen entre cero y uno.

7.4.2. Convolución

En las redes convolucionales todas las neuronas de la capa de entrada (los píxeles de las imágenes) no se conectan con todas las neuronas de la capa oculta del primer nivel como lo hacen las redes clásicas del tipo perceptrón multicapa o las redes que conocemos de forma genérica como redes densamente conectadas. Las conexiones se realizan por pequeñas zonas de la capa de entrada.

FIGURA 69. CONEXIÓN DE LAS NEURONAS DE LA CAPA DE ENTRADA CON LA CAPA OCULTA



Veamos un ejemplo para la base de datos de los dígitos del 1 a 9. Vamos a conectar cada neurona de la capa oculta con una región de 5×5 neurona, es decir, con 25 neuronas de la capa de entrada, que podemos denominarla ventana. Esta ventana va a ir recorriendo todo el espacio de entrada de 28×28 empezando por arriba y desplazándose de izquierda a derecha y de arriba abajo. Suponemos que los desplazamientos de la ventana son de un paso (un pixel) aunque este es un parámetro de la red que podemos modificar (en la programación lo llamaremos stride).

Para conectar la capa de entrada con la de salida utilizaremos una matriz de pesos (W) de tamaño 3×3 que recibe el nombre de filtro (filter) y el valor del sesgo. Para obtener el valor de cada neurona de la capa oculta realizaremos el producto escalar entre el filtro y la ventana de la capa

de entrada. Utilizamos el mismo filtro para obtener todas las neuronas de la capa oculta, es decir en todos los productos escalares siempre utilizamos la misma matriz, el mismo filtro.

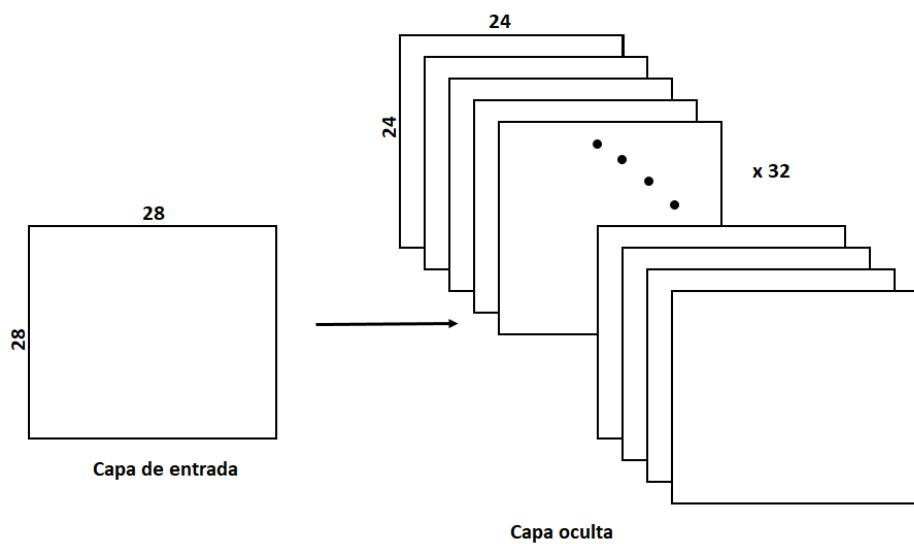
Se definen matemáticamente estos productos escalares a través de la siguiente expresión:

$$Y = X * W \rightarrow Y[i, j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i - k_1, j - k_2] W[k_1, k_2] \quad [177]$$

Matriz X					Matriz W			Resultado		
0	0	0	0	0						
0	2	2	2	1	0,3	0,0	0,2	1,6	4,2	1,6
0	4	0	3	0	0,0	0,8	0,0	4,0	3,6	3,8
0	1	1	5	0	0,6	0,0	0,4	0,8	2,6	4,0
0	0	0	0	0						

Como en este tipo de red un filtro sólo nos permite revelar una característica muy concreta de la imagen, lo que se propone es usar varios filtros simultáneamente, uno para cada característica que queramos detectar. Una forma visual de representarlo (si suponemos que queremos aplicar 32 filtros) es como se muestra a continuación:

FIGURA 70. PRIMERA CAPA DE LA RED CONVOLUCIONAL CON 32 FILTROS



Al resultado de la aplicación de los diferentes filtros se les suele aplicar la función de activación denominada RELU y que ya se comentó en la introducción.

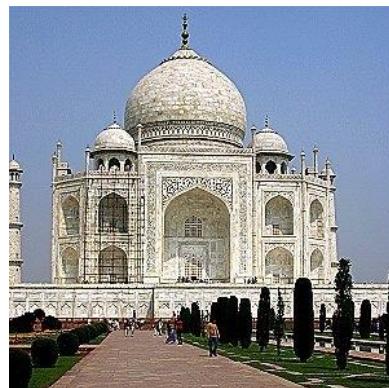
Una interesante fuente de información es la documentación del software gratuito GIMP donde expone diferentes efectos que se producen en las imágenes al aplicar diversas convoluciones.

Un ejemplo claro y didáctico lo podemos obtener de la documentación del software libre de dibujo y tratamiento de imágenes denominado GIMP (<https://docs.gimp.org/2.6/es/plug-in-convmatrix.html>). Algunos de estos efectos nos ayudan a entender la operación de los filtros en las redes convolucionales y cómo afectan a las imágenes, en concreto, el ejemplo que presenta lo realiza sobre la figura del Taj Mahal

El filtro enfocar lo que consigue es afinar los rasgos, los contornos lo que nos permite agudizar los objetos de la imagen. Toma el valor central de la matriz de cinco por cinco lo multiplica por cinco y le resta el valor de los cuatro vecinos. Al final hace una media, lo que mejora la resolución del pixel central porque elimina el ruido o perturbaciones que tiene de sus pixeles vecinos.

El filtro enfocar (Sharpen)

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0



Lo contrario al filtro enfocar lo obtenemos a través de la matriz siguiente, difuminando la imagen al ser estos píxeles mezclados o combinados con los píxeles cercanos. Promedia todos los píxeles vecinos a un pixel dado lo que implica que se obtienen bordes borrosos.

Filtro desenfocar

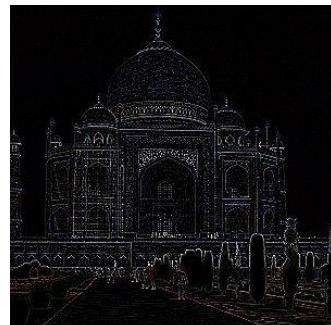
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0



Filtro Detectar bordes (Edge Detect)

Este efecto se consigue mejorando los límites o las aristas de la imagen. En cada píxel se elimina su vecino inmediatamente anterior en horizontal y en vertical. Se eliminan las similitudes vecinas y quedan los bordes resaltados. Al pixel central se le suman los cuatro píxeles vecinos y lo que queda al final es una medida de cómo de diferente es un píxel frente a sus vecinos. En el ejemplo, al hacer esto da un valor de cero de ahí que se observen tantas zonas oscuras.

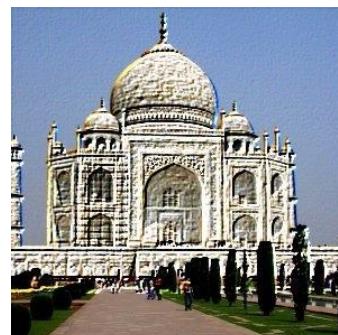
$$\begin{array}{|c|c|c|c|} \hline & & & \\ \hline & 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline & 0 & 1 & 0 \\ \hline & & & \end{array}$$



Filtro Repujado (Emboss)

En este filtro se observa que la matriz es simétrica y lo que intenta a través del diseño del filtro es mejorar los píxeles centrales y de derecha abajo restándole los anteriores. Se obtiene lo que en fotografía se conoce como un claro oscuro. Trata de mejorar las partes que tienen mayor relevancia.

$$\begin{array}{|c|c|c|c|} \hline & & & \\ \hline & -2 & -1 & 0 \\ \hline -1 & 1 & 1 \\ \hline 0 & 1 & 2 \\ \hline & & & \end{array}$$

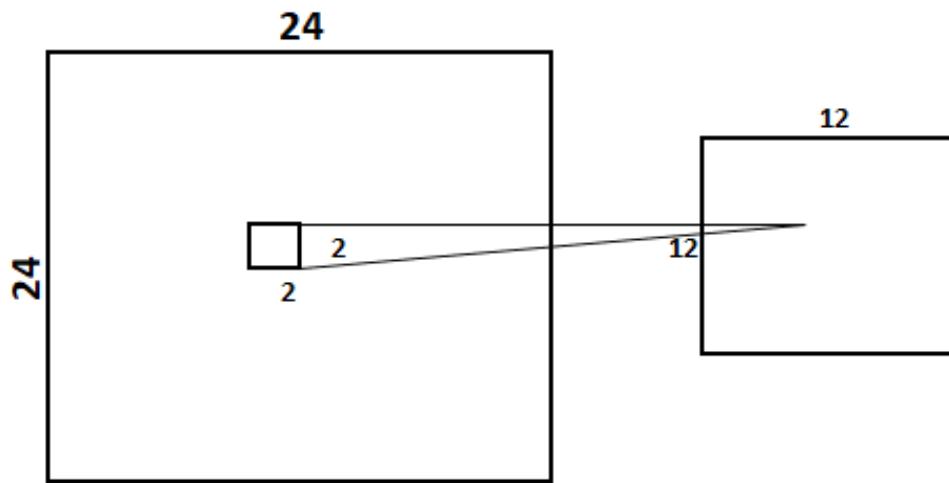


7.4.3. Pooling

Con la operación de pooling se trata de condensar la información de la capa convolucional. A este procedimiento también se le conoce como submuestreo. Es simplemente una operación en la que

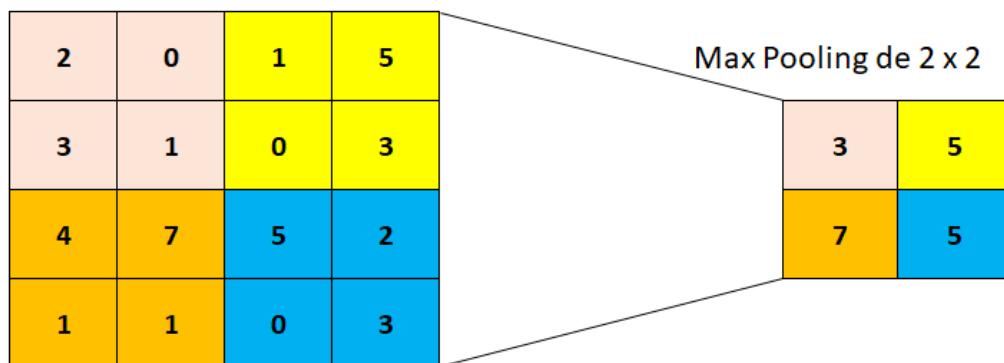
reducimos los parámetros de la red. Se aplica normalmente a través de dos operaciones: max-pooling y mean-pooling, que también es conocido como average-pooling. Tal y como se observa en la imagen siguiente, desde la capa de convolución se genera una nueva capa aplicando la operación a todas las agrupaciones, donde previamente hemos elegido el tamaño de la región; en la figura siguiente es de tamaño 2, con lo que pasamos de un espacio de 24×24 neuronas a la mitad, 12×12 en la capa de pooling.

FIGURA 71. ETAPA DE POOLING DE TAMAÑO 2×2



Vamos a estudiar el pooling suponiendo que tenemos una imagen de 5×5 píxeles y que queremos efectuar una agrupación max-pooling. Es la más utilizada, ya que obtiene buenos resultados. Observamos los valores de la matriz y se escoge el valor máximo de los cuatro bloques de matrices de dos por dos.

Max Pooling



En la agrupación **Average Pooling** la operación que se realiza es sustituir los valores de cada grupo de entrada por su valor medio. Esta transformación es menos utilizada que el max-pooling.

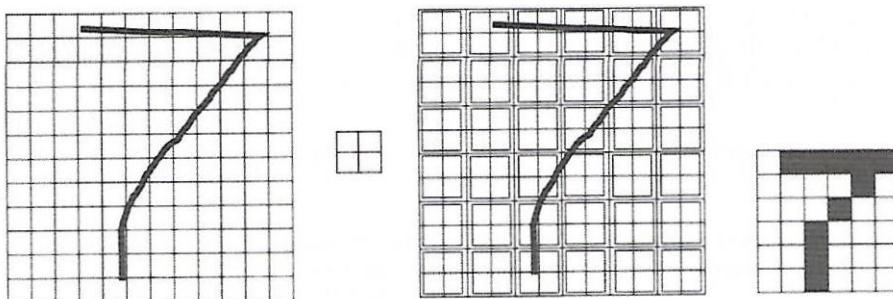
2	0	1	5
3	1	0	3
4	7	5	2
1	1	0	3

Average Pooling de 2 x 2

1,5	2,25
3,25	2,5

La transformación max-pooling presenta un tipo de invarianza local: pequeños cambios en una región local no varían el resultado final realizado con el max – pooling: se mantiene la relación espacial. Para ilustrar este concepto hemos escogido la imagen que presenta Torres (2020) donde se ilustra como partiendo de una matriz de 12 x 12 que representa al número 7, al aplicar la operación de max-pooling con una ventana de 2 x 2 se conserva la relación espacial.

FIGURA 72. MANTENIMIENTO DEL POOLING CON LA TRANSFORMACIÓN



Fuente: Torres. J. (2020)

7.4.4. Padding

Para explicar el concepto del Padding vamos a suponer que tenemos una imagen de 5 x 5 píxeles, es decir 25 neuronas en la capa de entrada, y que elegimos, para realizar la convolución, una ventana de 3 x 3. El número de neuronas de la capa oculta resultará ser de nueve. Enumeramos los píxeles de la imagen de forma natural del 1 al 25 para que resulte más sencillo de entender.

FIGURA Nº 73. OPERACIÓN DE CONVOLUCIÓN CON UNA VENTANA DE 3 x 3

Resultado del recorrido de la ventana de 3 x 3

Imagen					1 2 3	2 3 4	3 4 5	8 9 10	13 14 15	18 19 20
1	2	3	4	5	6	7	8	9	10	11
6	7	8	9	10	11	12	13	14	15	12
11	12	13	14	15	6	7	8	9	10	13
16	17	18	19	20	11	12	13	14	15	14
21	22	23	24	25	16	17	18	17	18	19
					11	12	13	12	13	14
					16	17	18	17	18	19
					21	22	23	22	23	24
								13	14	15
								18	19	20
								23	24	25

Pero si queremos obtener un tensor de salida que tenga las mismas dimensiones que la entrada podemos llenar la matriz de ceros antes de deslizar la ventana por ella. Vemos la figura siguiente donde ya se ha llenado de valores cero y obtenemos, después de deslizar la ventana de 3 x 3 de izquierda a derecha y de arriba abajo, las veinticinco matrices de la figura nº 71

FIGURA Nº 74. IMAGEN CON RELLENO DE CEROS

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	16	17	18	19	20	0
0	21	22	23	24	25	0
0	0	0	0	0	0	0

Cuando utilizamos el programa keras disponemos de dos opciones para llevar a cabo esta operación de padding: “same” y “valid”. Si utilizamos “valid” implica no hacer padding y el método “same” obliga a que la salida tenga la misma dimensión que la entrada.

FIGURA Nº 75. OPERACIÓN DE CONVOLUCIÓN CON VENTANA 3 x 3 Y PADDING

<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>6</td><td>7</td></tr></table>	0	0	0	0	1	2	0	6	7	<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	0	0	1	2	3	6	7	8	<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td>3</td><td>4</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	0	0	0	2	3	4	7	8	9	<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>8</td><td>9</td><td>10</td></tr></table>	0	0	0	3	4	5	8	9	10	<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>4</td><td>5</td><td>0</td></tr><tr><td>9</td><td>10</td><td>0</td></tr></table>	0	0	0	4	5	0	9	10	0
0	0	0																																															
0	1	2																																															
0	6	7																																															
0	0	0																																															
1	2	3																																															
6	7	8																																															
0	0	0																																															
2	3	4																																															
7	8	9																																															
0	0	0																																															
3	4	5																																															
8	9	10																																															
0	0	0																																															
4	5	0																																															
9	10	0																																															
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>6</td><td>7</td></tr><tr><td>0</td><td>11</td><td>12</td></tr></table>	0	1	2	0	6	7	0	11	12	<table border="1"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>6</td><td>7</td><td>8</td></tr><tr><td>11</td><td>12</td><td>13</td></tr></table>	1	2	3	6	7	8	11	12	13	<table border="1"><tr><td>2</td><td>3</td><td>4</td></tr><tr><td>7</td><td>8</td><td>9</td></tr><tr><td>12</td><td>13</td><td>14</td></tr></table>	2	3	4	7	8	9	12	13	14	<table border="1"><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>8</td><td>9</td><td>10</td></tr><tr><td>13</td><td>14</td><td>15</td></tr></table>	3	4	5	8	9	10	13	14	15	<table border="1"><tr><td>4</td><td>5</td><td>0</td></tr><tr><td>9</td><td>10</td><td>0</td></tr><tr><td>14</td><td>15</td><td>0</td></tr></table>	4	5	0	9	10	0	14	15	0
0	1	2																																															
0	6	7																																															
0	11	12																																															
1	2	3																																															
6	7	8																																															
11	12	13																																															
2	3	4																																															
7	8	9																																															
12	13	14																																															
3	4	5																																															
8	9	10																																															
13	14	15																																															
4	5	0																																															
9	10	0																																															
14	15	0																																															
<table border="1"><tr><td>0</td><td>6</td><td>7</td></tr><tr><td>0</td><td>11</td><td>12</td></tr><tr><td>0</td><td>16</td><td>17</td></tr></table>	0	6	7	0	11	12	0	16	17	<table border="1"><tr><td>6</td><td>7</td><td>8</td></tr><tr><td>11</td><td>12</td><td>13</td></tr><tr><td>16</td><td>17</td><td>18</td></tr></table>	6	7	8	11	12	13	16	17	18	<table border="1"><tr><td>7</td><td>8</td><td>9</td></tr><tr><td>12</td><td>13</td><td>14</td></tr><tr><td>17</td><td>18</td><td>19</td></tr></table>	7	8	9	12	13	14	17	18	19	<table border="1"><tr><td>8</td><td>9</td><td>10</td></tr><tr><td>13</td><td>14</td><td>15</td></tr><tr><td>18</td><td>19</td><td>20</td></tr></table>	8	9	10	13	14	15	18	19	20	<table border="1"><tr><td>9</td><td>10</td><td>0</td></tr><tr><td>14</td><td>15</td><td>0</td></tr><tr><td>19</td><td>20</td><td>0</td></tr></table>	9	10	0	14	15	0	19	20	0
0	6	7																																															
0	11	12																																															
0	16	17																																															
6	7	8																																															
11	12	13																																															
16	17	18																																															
7	8	9																																															
12	13	14																																															
17	18	19																																															
8	9	10																																															
13	14	15																																															
18	19	20																																															
9	10	0																																															
14	15	0																																															
19	20	0																																															
<table border="1"><tr><td>0</td><td>11</td><td>12</td></tr><tr><td>0</td><td>16</td><td>17</td></tr><tr><td>0</td><td>21</td><td>22</td></tr></table>	0	11	12	0	16	17	0	21	22	<table border="1"><tr><td>11</td><td>12</td><td>13</td></tr><tr><td>16</td><td>17</td><td>18</td></tr><tr><td>21</td><td>22</td><td>23</td></tr></table>	11	12	13	16	17	18	21	22	23	<table border="1"><tr><td>12</td><td>13</td><td>14</td></tr><tr><td>17</td><td>18</td><td>19</td></tr><tr><td>22</td><td>23</td><td>24</td></tr></table>	12	13	14	17	18	19	22	23	24	<table border="1"><tr><td>13</td><td>14</td><td>15</td></tr><tr><td>18</td><td>19</td><td>20</td></tr><tr><td>23</td><td>24</td><td>25</td></tr></table>	13	14	15	18	19	20	23	24	25	<table border="1"><tr><td>14</td><td>15</td><td>0</td></tr><tr><td>19</td><td>20</td><td>0</td></tr><tr><td>24</td><td>25</td><td>0</td></tr></table>	14	15	0	19	20	0	24	25	0
0	11	12																																															
0	16	17																																															
0	21	22																																															
11	12	13																																															
16	17	18																																															
21	22	23																																															
12	13	14																																															
17	18	19																																															
22	23	24																																															
13	14	15																																															
18	19	20																																															
23	24	25																																															
14	15	0																																															
19	20	0																																															
24	25	0																																															
<table border="1"><tr><td>0</td><td>16</td><td>17</td></tr><tr><td>0</td><td>21</td><td>22</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	0	16	17	0	21	22	0	0	0	<table border="1"><tr><td>16</td><td>17</td><td>18</td></tr><tr><td>21</td><td>22</td><td>23</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	16	17	18	21	22	23	0	0	0	<table border="1"><tr><td>17</td><td>18</td><td>19</td></tr><tr><td>22</td><td>23</td><td>24</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	17	18	19	22	23	24	0	0	0	<table border="1"><tr><td>18</td><td>19</td><td>20</td></tr><tr><td>23</td><td>24</td><td>25</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	18	19	20	23	24	25	0	0	0	<table border="1"><tr><td>19</td><td>20</td><td>0</td></tr><tr><td>24</td><td>25</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	19	20	0	24	25	0	0	0	0
0	16	17																																															
0	21	22																																															
0	0	0																																															
16	17	18																																															
21	22	23																																															
0	0	0																																															
17	18	19																																															
22	23	24																																															
0	0	0																																															
18	19	20																																															
23	24	25																																															
0	0	0																																															
19	20	0																																															
24	25	0																																															
0	0	0																																															

7.4.5. Stride

Hasta ahora, la forma de recorrer la matriz a través de la ventana se realiza desplazándola de un solo paso, pero podemos cambiar este hiperparámetro conocido como stride. Al aumentar el paso se decrementa la información que pasará a la capa posterior.

A continuación, se muestra el resultado de las cuatro matrices que obtenemos con un stride de valor 3.

FIGURA Nº 76. OPERACIÓN DE CONVOLUCIÓN CON UNA VENTANA DE 3 x 3

Resultado del recorrido de la ventana de 3 x 3 con un stride de 2

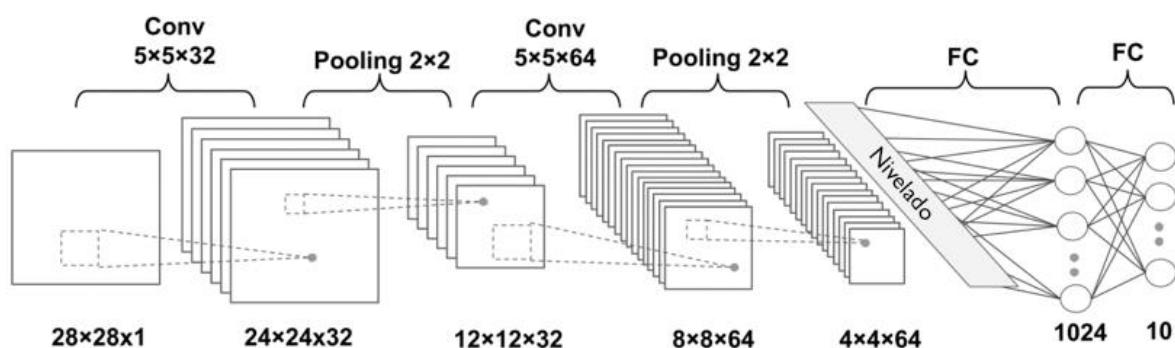
Imagen							
1	2	3	4	5	1	2	3
6	7	8	9	10	6	7	8
11	12	13	14	15	11	12	13
16	17	18	19	20	11	12	13
21	22	23	24	25	16	17	18
					21	22	23
					3	4	5
					8	9	10
					13	14	15
					11	12	13
					16	17	18
					21	22	23
					13	14	15
					18	19	20
					23	24	25

Finalmente, para resumir, una red convolucional contiene los siguientes elementos:

- **Entrada:** Son el número de pixeles de la imagen. Serán alto, ancho y profundidad. Tenemos un solo color (escala de grises) o tres: rojo, verde y azul.
- **Capa de convolución:** procesará la salida de neuronas que están conectadas en «regiones locales» de entrada (es decir pixeles cercanos), calculando el producto escalar entre sus pesos (valor de pixel) y una pequeña región a la que están conectados. En este epígrafe se presentan las imágenes con 32 filtros, pero puede realizarse con la cantidad que deseemos.
- «**Capa RELU**» Se aplicará la función de activación en los elementos de la matriz.
- **POOL (agrupar) o Submuestreo:** Se procede normalmente a una reducción en las dimensiones alto y ancho, pero se mantiene la profundidad.
- **CAPA tradicional.** Se finalizará con la red de neuronas feedforward (Perceptrón multicapa que se denomina normalmente como red densamente conectada) que vinculará con la última capa de subsampling y finalizará con la cantidad de neuronas que queremos clasificar.

En el gráfico siguiente se muestran todas las fases de una red neuronal convolucional.

FIGURA Nº 77. OPERACIÓN DE CONVOLUCIÓN COMPLETA



Fuente: Raschka y Mirjalili (2019)

7.4.6. Redes convolucionales con nombre propio

Existen en la actualidad muchas arquitecturas de redes neuronales convolucionales que ya están preparadas, probadas, disponibles e incorporadas en el software de muchos programas como Keras y Tensorflow.

Vamos a comentar algunos de estos modelos, bien por ser los primeros, o por sus excelentes resultados en concursos como el ILSVRC (Large Scale Visual Recognition Challenge). Estas estructuras merecen atención dado que son excelentes para estudiarlas e incorporarlas por su notable éxito. El ILSVRC fue un concurso celebrado de 2011 a 2016 de donde nacieron las principales aportaciones efectuadas en las redes convolucionales. Este concurso fue diseñado para estimular la innovación en el campo de la visión computacional. Actualmente se desarrollan este tipo de concursos a través de la plataforma web: <https://www.kaggle.com/>

Para ver más prototipos de redes convolucionales y los últimos avances y consejos sobre las redes convolucionales se puede consultar el siguiente artículo “Recent Advances in Convolutional Neural Networks” de Jiuxiang. G. et al. (2019)

Los cinco modelos más destacados hasta el año 2017 son los siguientes: LeNet-5, Alexnet, GoogLeNet, VGG y Restnet.

1. **LeNet-5.** Este modelo de Yann LeCun de los años 90 consiguió excelentes resultados en la lectura de códigos postales consta de imágenes de entrada de 32 x 32 píxeles seguida de dos etapas de convolución – pooling, una capa densamente conectada y una capa softmax final que nos permite conocer los números o las imágenes.
2. **AlexNet.** Fue la arquitectura estrella a partir del año 2010 en el ILSVRC y popularizada en el documento de 2012 de Alex Krizhevsky, et al. titulado “Clasificación de ImageNet con redes neuronales convolucionales profundas”.

Podemos resumir los aspectos clave de la arquitectura relevantes en los modelos modernos de la siguiente manera:

- ✓ Empleo de la función de activación ReLU después de capas convolucionales y softmax para la capa de salida.
- ✓ Uso de la agrupación máxima en lugar de la agrupación media.
- ✓ Utilización de la regularización de Dropout entre las capas totalmente conectadas.
- ✓ Patrón de capa convolucional alimentada directamente a otra capa convolucional.
- ✓ Uso del aumento de datos (Data Aumentation,)

3. **VGG.** Este prototipo fue desarrollado por un grupo de investigación de Geometría Visual en Oxford. Obtuvo el segundo puesto en la competición del año 2014 del ILSVRC. Las aportaciones principales de la investigación se pueden encontrar en el documento titulado “Redes

convolucionales muy profundas para el reconocimiento de imágenes a gran escala” desarrollado por Karen Simonyan y Andrew Zisserman.

Este modelo contribuyó a demostrar que la profundidad de la red es una componente crítica para alcanzar unos buenos resultados. Otra diferencia importante con los modelos anteriores y que actualmente es muy utilizada es el uso de un gran número de filtros y de tamaño reducido. Estas redes emplean ejemplos de dos, tres e incluso cuatro capas convolucionales apiladas antes de usar una capa de agrupación máxima. En esta arquitectura el número de filtros aumenta con la profundidad del modelo. El modelo comienza con 64 y aumenta a través de los filtros de 128, 256 y 512 al final de la parte de extracción de características del modelo.

Los investigadores evaluaron varias variantes de la arquitectura si bien en los programas sólo se hace referencia a dos de ellas que son las que aportan un mayor rendimiento y que son nombradas por las capas que tienen: VGG-16 y VGG-19.

4. **GoogLeNet. GoogLeNet fue desarrollado por investigadores de Google Research.** de Google, que con su módulo denominado de inception reduce drásticamente los parámetros de la red (10 veces menos que AlexNet) y de ella han derivado varias versiones como la Inception-v4. Esta arquitectura ganó la competición en el año 2014 y su éxito se debió a que la red era mucho más profunda (muchas más capas) y como ya se ha indicado introdujeron en el modelo las subredes llamadas inception.

Las aportaciones principales en el uso de capas convolucionales fueron propuestos en el documento de 2015 por Christian Szegedy, et al. titulado “Profundizando con las convoluciones”.

Estos autores introducen una arquitectura llamada “inicio” y un modelo específico denominado GoogLenet. El módulo inicio es un bloque de capas convolucionales paralelas con filtros de diferentes tamaños y una capa de agrupación máxima de 3×3 , cuyos resultados se concatenan.

Otra decisión de diseño fundamental en el modelo inicial fue la conexión de la salida en diferentes puntos del modelo que lograron realizar con la creación de pequeñas redes de salida desde la red principal y que fueron entrenadas para hacer una predicción. La intención era proporcionar una señal de error adicional de la tarea de clasificación en diferentes puntos del modelo profundo para abordar el problema de los gradientes de fuga.

5. **Red Residual o ResNet.** Esta arquitectura gano la competición de 2015 y fue creada por el grupo de investigación de Microsoft. Se puede ampliar la información en He, et al. en su documento de 2016 titulado “ Aprendizaje profundo residual para el reconocimiento de la imagen ”.

Esta red es extremadamente profunda con 152 capas, confirmando al pasar los años que las redes son cada vez más profundas, más capas, pero con menos parámetros que estimar.

La cuestión clave del diseño de esta red es la incorporación de la idea de bloques residuales que hacen uso de conexiones directa. Un bloque residual, según los autores, “es un patrón de dos capas convolucionales con activación ReLU donde la salida del bloque se combina con la entrada al bloque, por ejemplo, la conexión de acceso directo”

Otra clave, en este caso para el entrenamiento de la red tan profunda es lo que llamaron skip connections que implica que la señal con la que se alimenta una capa también se agregue a una capa que se encuentre más adelante.

Resumiendo, las tres principales aportaciones de este modelo son:

- Empleo de conexiones de acceso directo.
- Desarrollo y repetición de los bloques residuales.
- Modelos muy profundos (152 capas) Aunque se encuentran otros modelos que también son muy populares con 34, 50 y 101 capas.

Una buena parte de los modelos comentados se incluyen en la librería de Keras y se pueden encontrar en la siguiente dirección de internet:

<https://keras.io/api/applications/>

Según los autores del programa Keras: “Las aplicaciones Keras son modelos de aprendizaje profundo que están disponibles junto con pesos preentrenados. Estos modelos se pueden usar para predicción, extracción de características y ajustes. Los pesos se descargan automáticamente cuando se crea una instancia de un modelo. Se almacenan en ~ / .keras / models /. Tras la creación de instancias, los modelos se construirán de acuerdo con el formato de datos de imagen establecido en su archivo de configuración de Keras en ~ / .keras / keras.json. Por ejemplo, si ha configurado image_data_format = channel_last, cualquier modelo cargado desde este repositorio se construirá de acuerdo con la convención de formato de datos TensorFlow, "Altura-Ancho-Profundidad".

FIGURA Nº 78. MODELOS PREENTRENADOS EN KERAS

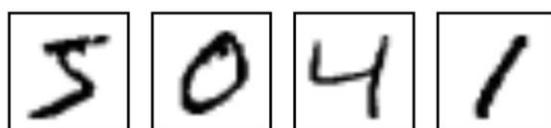
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetB200	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetB21	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetB22	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetB23	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV25	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

Fuente : <https://keras.io/api/applications/>

7.4.7. Ejemplos con Python y R.

Tanto para el código de python como de R, utilizaremos como ejemplos varias bases de datos de imágenes ampliamente utilizadas en el mundo del Deep Learning: MNIST, la base de datos de perros y gatos y CIFAR10.

MNIST (National Institute of Standards and Technology database) es un gran conjunto de datos de dígitos escritos a mano que se usa comúnmente para entrenar varios sistemas de procesamiento de imágenes. Contiene 60.000 imágenes de entrenamiento y 10.000 imágenes de prueba. Las imágenes en blanco y negro se normalizaron para encajar en un cuadrado de 28x28 píxeles en niveles de escala de grises.

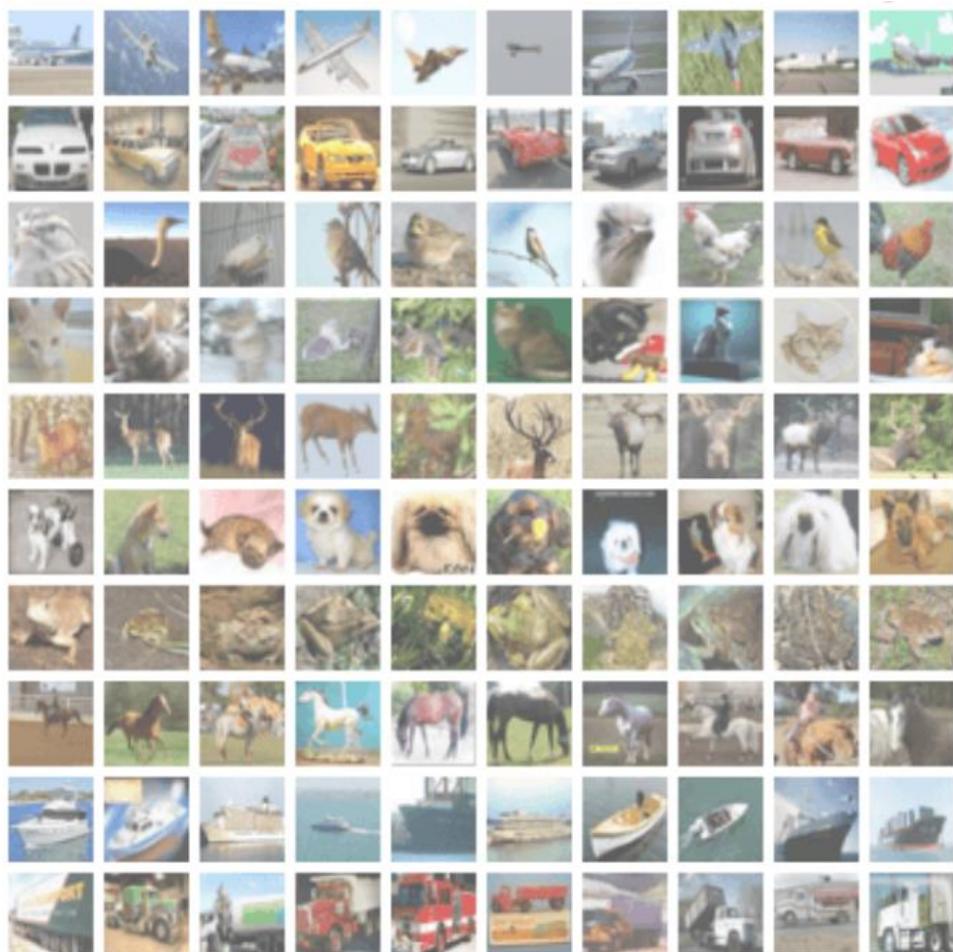


En Wikipedia se puede encontrar el cuadro siguiente que expresa diferente información sobre los algoritmos utilizados en la clasificación de estas imágenes de números así como el éxito alcanzado.

This is a table of some of the machine learning methods used on the database and their error rates, by type of classifier:

Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6 ^[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52 ^[18]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87 ^[19]
Non-linear classifier	40 PCA + quadratic classifier	None	None	3.3 ^[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56 ^[20]
Neural network	2-layer 784-800-10	None	None	1.6 ^[21]
Neural network	2-layer 784-800-10	elastic distortions	None	0.7 ^[21]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	elastic distortions	None	0.35 ^[22]
Convolutional neural network	6-layer 784-40-80-500-1000-2000-10	None	Expansion of the training data	0.31 ^[15]
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.27 ^[16]
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	elastic distortions	Width normalizations	0.23 ^[8]
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.21 ^[17]

El conjunto de datos CIFAR-10, que son las siglas del *Canadian Institute for Advanced Research* consta de 60.000 imágenes en color de 32x32 pixeles en 10 clases, con 6.000 imágenes por clase. Hay 50.000 imágenes de entrenamiento y 10.000 imágenes de prueba.



El conjunto de imágenes de gatos y perros es muy popular en la clasificación de Deep learning. Las imágenes se pueden descargar desde la página web: <https://kaggle.com/c/dogs-vs-cats/data>. Para que la fase de entrenamiento se pueda reducir considerablemente, utilizaremos un conjunto de sólo 5.000 imágenes.



Figura 10.4

7.4.7.1. Código de Python

Los ejemplos de código de Python que se presentan a continuación se han implementado con la librería Keras disponible tanto en R como en Python y con las bases de datos mencionadas anteriormente:

- En primer lugar, se realiza un ejemplo con una red neuronal densamente conectada donde se establecen diferentes capas ocultas, la función de transferencia entre capas y el método de regulación dropout.
- Posteriormente, se implementa con los mismos datos un modelo de red convolucional.

También, para la base de datos de datos de perros y gatos se realizan tres métodos con Keras:

- Una red convolucional.
- Se implementa otro modelo con Data Aumentation que mejora los resultados de la red anterior.
- Se programa otro modelo aplicando Feature Extraction con un modelo preentrenado denominado VGG16.

Es muy importante saber la versión de keras que estamos utilizando ya que al cambiar de versión es muy probable que algunos comandos que se presentan no funcionen. Para saber la versión de una librería en Python se tienen que emplear los siguientes comandos:

```
import keras  
print('La versión de Keras que se utiliza es la ',keras.__version__)
```

La versión de Keras que se utiliza es la 2.2.4

Modelo de red totalmente conectada

En primer lugar, importamos las librerías que vamos a utilizar.

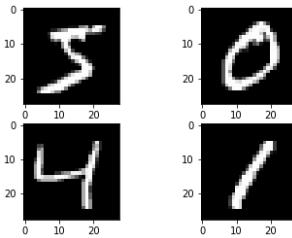
```
import numpy as np  
import matplotlib.pyplot as plt  
import tensorflow as tf  
import keras  
from keras.models import Sequential  
from keras.utils import np_utils  
from keras.layers.core import Dense, Activation, Flatten, Dropout  
from sklearn.preprocessing import normalize  
from keras.datasets import mnist  
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Realizamos un gráfico de los números

```
plt.subplot(221)  
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))  
plt.subplot(222)  
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))  
plt.subplot(223)  
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))  
plt.subplot(224)  
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
```

Mostramos el gráfico

```
plt.show()
```



Ahora importamos los datos

```
# reshape for vector input y los adaptamos par la red neuronal
N, x, y = X_train.shape
X_train = normalize(np.reshape(X_train, (N, x * y)))
N, x, y = X_test.shape
X_test = normalize(np.reshape(X_test, (N, x * y)))
# one-hot encoding
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Definimos ahora la estructura de la red neuronal

```
model = Sequential()
model.add(Dense(output_dim=750, input_dim=784))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(150))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Podemos obtener una descripción de nuestra red con el comando model.summary()

```
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 32, 24, 24)	832
max_pooling2d_1 (MaxPooling2D)	(None, 32, 12, 12)	0
dropout_1 (Dropout)	(None, 32, 12, 12)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 128)	589952
dense_2 (Dense)	(None, 10)	1290
<hr/>		
Total params: 592,074		
Trainable params: 592,074		
Non-trainable params: 0		

Ahora compilamos el modelo (model.compile) y llevamos a cabo el entrenamiento de la red (model.fit)

```
model.compile(loss='categorical_crossentropy', optimizer='Nadam', metrics=['accuracy'])  
fit = model.fit(X_train, y_train, batch_size=128, nb_epoch=10, verbose=0)
```

Imprimimos la precisión de nuestro modelo, de acuerdo con la función de pérdida especificada en model.compile.

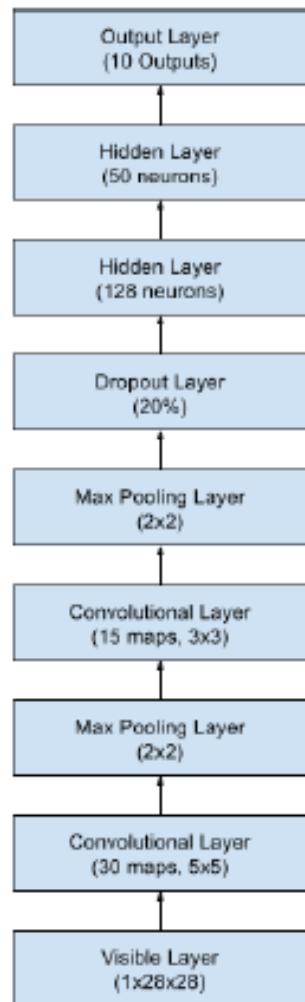
```
score = model.evaluate(X_test, y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

Test score: 0.07673678106308071

Test accuracy: 0.9824

Modelo de red convolucional

El modelo de red neuronal convolucional que vamos a desarrollar ahora tiene dos capas convolucionales con max pooling y una red neuronal de salidas con dos capas ocultas de 128 y 50 neuronas y la capa de salida con función de activación softmax.



Importamos las librerías necesarias

```
import numpy
import pandas as pd
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

Cargamos los datos

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

Definimos una función con las diferentes capas de la red convolucional

```
def larger_model():
```

```
# creamos el modelo
```

```
model = Sequential()  
model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2, 2)))  
model.add(Conv2D(15, (3, 3), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2, 2)))  
model.add(Dropout(0.2))  
model.add(Flatten())  
model.add(Dense(128, activation='relu'))  
model.add(Dense(50, activation='relu'))  
model.add(Dense(num_classes, activation='softmax'))
```

```
# Compilamos el modelo
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  
return model  
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 30, 24, 24)	780
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 30, 12, 12)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 15, 10, 10)	4065
<hr/>		
max_pooling2d_2 (MaxPooling2D)	(None, 15, 5, 5)	0
<hr/>		
dropout_5 (Dropout)	(None, 15, 5, 5)	0
<hr/>		
flatten_1 (Flatten)	(None, 375)	0
<hr/>		
dense_6 (Dense)	(None, 128)	48128
<hr/>		
dense_7 (Dense)	(None, 50)	6450
<hr/>		
dense_8 (Dense)	(None, 10)	510
<hr/>		
Total params: 59,933		
Trainable params: 59,933		
Non-trainable params: 0		

Construcción del modelo

```
model = larger_model()
```

Entrenamos el modelo

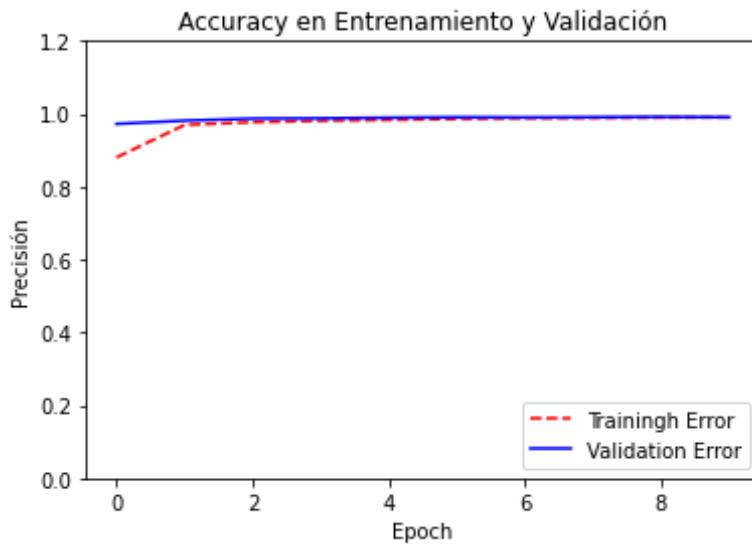
```
historia = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)
```

Definimos una función para ver gráficamente la evolución de las métricas de rendimiento durante el entrenamiento

```
hist = pd.DataFrame(historia.history)  
hist
```

```
In [24]: hist  
Out[24]:  
      val_loss  val_acc      loss      acc  
0  0.087452  0.9724  0.389875  0.880400  
1  0.057190  0.9813  0.100148  0.969617  
2  0.041589  0.9869  0.073584  0.977033  
3  0.037516  0.9876  0.060906  0.981083  
4  0.034129  0.9893  0.051887  0.983417  
5  0.027802  0.9907  0.044547  0.986100  
6  0.029121  0.9902  0.039203  0.987900  
7  0.025310  0.9908  0.035396  0.988533  
8  0.023738  0.9916  0.033032  0.989633  
9  0.026123  0.9906  0.031010  0.990050
```

```
def plot_history(history):  
    hist = pd.DataFrame(history.history)  
    hist['epoch'] = history.epoch  
  
    plt.figure()  
    plt.xlabel('Epoch')  
    plt.ylabel('Función de pérdida')  
    plt.plot(hist['epoch'], hist['loss'], 'r--',  
             label='Training Error')  
    plt.plot(hist['epoch'], hist['val_loss'], 'b',  
             label='Validation Error')  
    plt.ylim([0,0.5])  
    plt.title('Accuracy en Entrenamiento y Validación')  
    plt.legend()  
    plt.show()  
plot_history(historia)
```



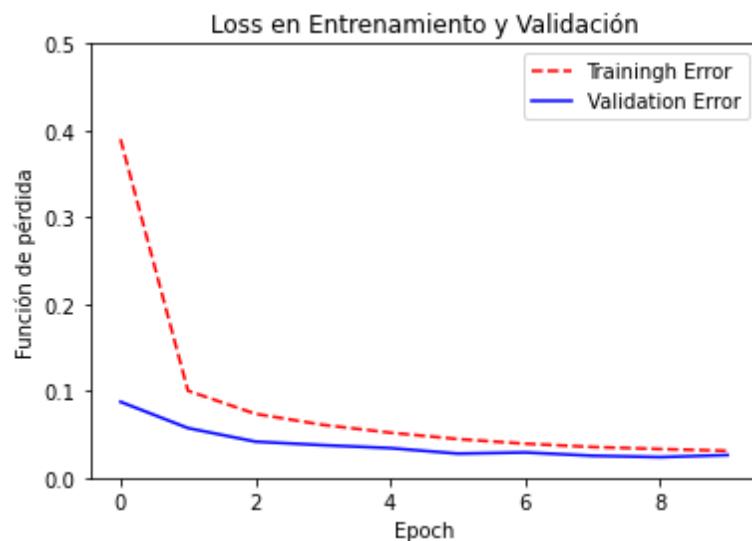
```

def plot_history(history):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Función de pérdida')
    plt.plot(hist['epoch'], hist['loss'], 'r--',
             label='Training Error')
    plt.plot(hist['epoch'], hist['val_loss'], 'b',
             label = 'Validation Error')
    plt.ylim([0,0.5])
    plt.title('Loss en Entrenamiento y Validación')
    plt.legend()
    plt.show()

plot_history(historia)

```

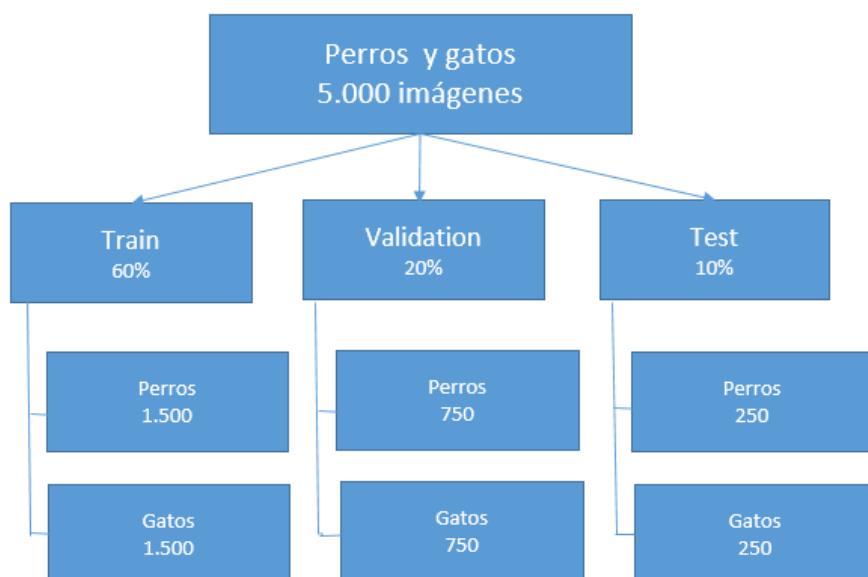


Ejemplo con imágenes de perros y gatos

Vamos a estructurar las imágenes en tres carpetas, una de entrenamiento (train) otra de validación (Validation) y otra de prueba (test) dado que si se ofrece la información tal y como la entiende keras y tensorflow nos facilita el trabajo para problemas en la vida real.

Del conjunto de imágenes inicial extraemos una muestra de imágenes para train, que tiene dos carpetas, una para gatos y otra para perros, cada una con 1000 ejemplos. A las carpetas de validación y de test, también divididas en dos, las asignamos 500 imágenes para cada una de ellas. La siguiente imagen ilustra el reparto de los diferentes conjuntos de información.

FIGURA Nº 79. ESQUEMA DE DISTRIBUCIÓN DE LA MUESTRA ENTRENAMIENTO, VALIDACION Y TEST



A continuación, vamos a exponer el código de Python:

```
import os
import tensorflow as tf
from tensorflow.compat.v1.keras import backend as K
from tensorflow import keras
config = tf.compat.v1.ConfigProto( device_count = {'GPU': 1 , 'CPU': 16} )
sess = tf.compat.v1.Session(config=config)
K.set_session(sess)

from keras import backend as K

K.tensorflow_backend._get_available_gpus()
print(tf.__version__)
base_dir = 'F:\MASTER_2020\MODULO_8_2020\perros_gatos_5000'
```

```

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directorio con las imagenes de training
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directorio con las imagenes de validation
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

# Directorio con las imagenes de test
test_cats_dir = os.path.join(test_dir, 'cats')
test_dogs_dir = os.path.join(test_dir, 'dogs')

```

```

train_cat_fnames = os.listdir( train_cats_dir )
print(train_cat_fnames[:5])

train_dog_fnames = os.listdir( train_dogs_dir )
print(train_dog_fnames[:5])

validation_cat_fnames = os.listdir( validation_cats_dir )
print(validation_cat_fnames[:5])

validation_dog_fnames = os.listdir( validation_dogs_dir )
print(validation_dog_fnames[:5])

test_cat_fnames = os.listdir( test_cats_dir )
print(test_cat_fnames[:5])

test_dog_fnames = os.listdir( test_dogs_dir )
print(test_dog_fnames[:5])

['cat.0.jpg', 'cat.1.jpg', 'cat.10.jpg', 'cat.100.jpg', 'cat.1000.jpg']
['dog.0.jpg', 'dog.1.jpg', 'dog.10.jpg', 'dog.100.jpg', 'dog.1000.jpg']
['cat.1500.jpg', 'cat.1501.jpg', 'cat.1502.jpg', 'cat.1503.jpg', 'cat.1504.jpg']
['dog.1501.jpg', 'dog.1502.jpg', 'dog.1503.jpg', 'dog.1504.jpg', 'dog.1505.jpg']
['cat.10000.jpg', 'cat.10001.jpg', 'cat.10002.jpg', 'cat.10003.jpg', 'cat.10004.jpg']
['dog.5000.jpg', 'dog.5001.jpg', 'dog.5002.jpg', 'dog.5003.jpg', 'dog.5004.jpg']

```

```

print('total training cat images :', len(os.listdir(train_cats_dir) ))

```

```
print('total training dog images :', len(os.listdir(train_dogs_dir )))

print('total validation cat images :', len(os.listdir( validation_cats_dir )))
print('total validation dog images :', len(os.listdir( validation_dogs_dir )))

print('total test cat images :', len(os.listdir( test_cats_dir )))
print('total test dog images :', len(os.listdir( test_dogs_dir )))

total training cat images : 1500
total training dog images : 1500
total validation cat images : 750
total validation dog images : 750
total test cat images : 250
total test dog images : 250
```

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
def print_pictures(dir, fnames):
```

```
# presentaremos imágenes en una configuración de 4x4
```

```
nrows = 4
ncols = 4
pic_index = 0 # Índice para iterar sobre las imágenes

fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)
pic_index+=8

next_pix = [os.path.join(dir, fname)
            for fname in fnames[ pic_index-8:pic_index]
            ]

for i, img_path in enumerate(next_pix):
    sp = plt.subplot(nrows, ncols, i + 1)
    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()
```

```

print("Figura 10.3")
print_pictures(train_cats_dir, train_cat_fnames)

print("Figura 10.4")
print_pictures(train_dogs_dir, train_dog_fnames)

print("Figura 10.5")
print_pictures(validation_cats_dir, validation_cat_fnames)

print("Figura 10.6")
print_pictures(validation_dogs_dir, validation_dog_fnames)

print("Figura 10.7")
print_pictures(test_cats_dir, test_cat_fnames)

print("Figura 10.8")
print_pictures(test_dogs_dir, test_dog_fnames)

```

Figura 10.6



Modelo de red convolucional básico

```

import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

```

```

model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(Flatten())
model.add(Dense(512, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

```

```

model.summary()

Model: "sequential"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	321177
dense_1 (Dense)	(None, 1)	513

```

Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics = ['acc'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator( rescale = 1.0/255. )
validation_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen = ImageDataGenerator( rescale = 1.0/255. )

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    target_size=(150, 150))
validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                               batch_size=20,
                                                               target_size = (150, 150))

test_generator = test_datagen.flow_from_directory(validation_dir,
                                                 batch_size=20,
                                                 target_size = (150, 150))
```

```
Found 3000 images belonging to 2 classes.
Found 1500 images belonging to 2 classes.
Found 1500 images belonging to 2 classes.
```

```
batch_size = 20

steps_per_epoch = train_generator.n // batch_size
validation_steps = validation_generator.n // batch_size

print (steps_per_epoch)
print (validation_steps)
```

```
150
75
```

```

import sys
from PIL import Image
sys.modules['Image'] = Image
from PIL import Image
print(Image.__file__)
import Image
print(Image.__file__)
history = model.fit(
    train_generator,
    steps_per_epoch= steps_per_epoch,
    epochs=100,
    validation_data=validation_generator,
    validation_steps= validation_steps,
    verbose=2)

```

```

Epoch 94/100
150/150 - 143s - loss: 9.2485e-04 - acc: 0.9993 - val_loss: 2.1554 - val_acc: 0.7660
Epoch 95/100
150/150 - 140s - loss: 0.0027 - acc: 0.9990 - val_loss: 2.2804 - val_acc: 0.7747
Epoch 96/100
150/150 - 140s - loss: 0.0046 - acc: 0.9987 - val_loss: 2.1901 - val_acc: 0.7707
Epoch 97/100
150/150 - 141s - loss: 1.5648e-04 - acc: 1.0000 - val_loss: 2.2502 - val_acc: 0.7773
Epoch 98/100
150/150 - 142s - loss: 3.4261e-04 - acc: 0.9997 - val_loss: 2.9272 - val_acc: 0.7507
Epoch 99/100
150/150 - 140s - loss: 0.0049 - acc: 0.9990 - val_loss: 2.7712 - val_acc: 0.7467
Epoch 100/100
150/150 - 140s - loss: 0.0021 - acc: 0.9993 - val_loss: 2.3029 - val_acc: 0.7620

```

```

history_dict = history.history
print(history_dict.keys())

```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```

acc      = history.history[  'acc' ]
val_acc  = history.history[ 'val_acc' ]
loss     = history.history[  'loss' ]
val_loss = history.history['val_loss']

```

```
epochs  = range(1,len(acc)+1,1) # obtener número de epochs del eje X
```

```

plt.plot ( epochs,  acc, 'r--', label='Training acc' )
plt.plot ( epochs, val_acc, 'b', label='Validation acc')

```

```

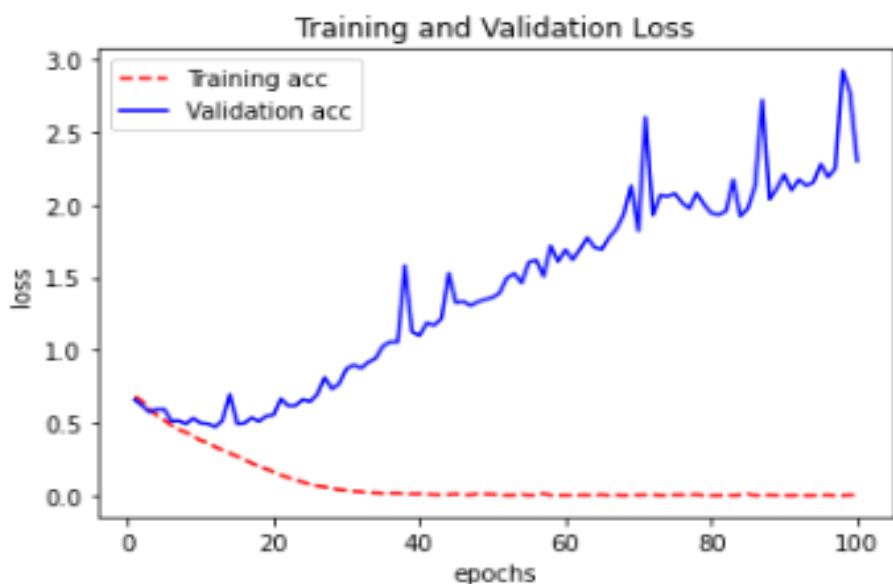
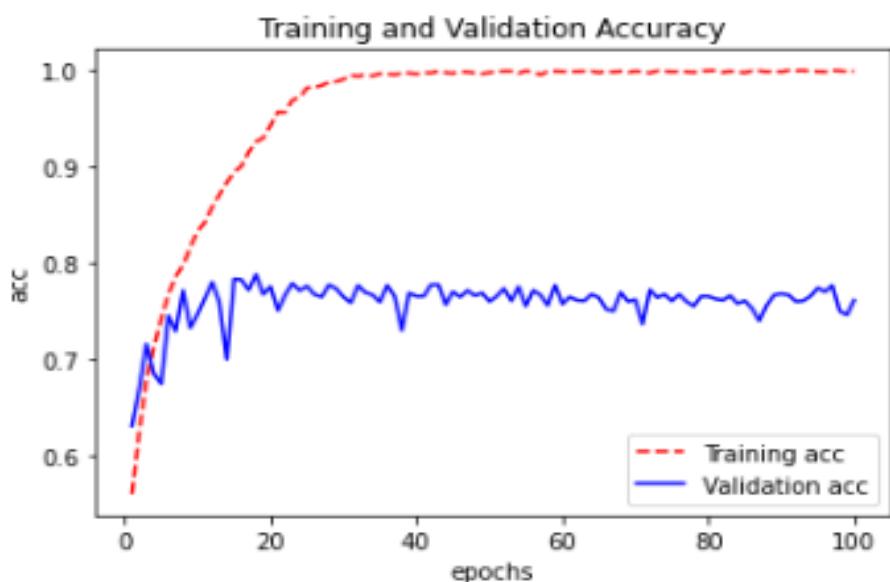
plt.title('Training and Validation Accuracy')
plt.ylabel('acc')
plt.xlabel('epochs')

plt.legend()
plt.figure()

plt.plot ( epochs,    loss, 'r--', label='Training acc' )
plt.plot ( epochs, val_loss , 'b', label='Validation acc' )
plt.title ('Training and Validation Loss' )
plt.ylabel('loss')
plt.xlabel('epochs')

plt.legend()
plt.figure()

```



```
test_lost, test_acc= model.evaluate(test_generator)
print ("Test Accuracy:", test_acc)
```

Test Accuracy: 0.762

```
history_dict = history.history
print(history_dict.keys())
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```
from tensorflow.keras.preprocessing import image
import numpy as np
```

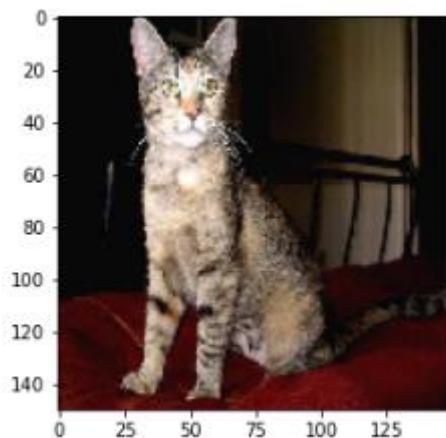
```
path='F:\\MASTER_2020\\MODULO_8_2020\\DOC_2020_DEEP_LEARNIG\\DIA_24_06_
2020\\perros_gatos_5000\\test\\cats\\cat.10000.jpg'
img=image.load_img(path, target_size=(150, 150))
```

```
x=image.img_to_array(img)
image=np.expand_dims(x, axis=0)
```

```
classes = model.predict(image)
print(classes)
```

```
plt.imshow(img)
plt.show()
```

```
if classes>0: print( "cat.10000.jpg IS A DOG")
else: print( "cat.10000.jpg IS A CAT")
```



cat.10000.jpg IS A DOG

Mejora del modelo con Data Aumentation y Transfer Learning

```
# tensorflow_version 2.x
```

```
import os
import tensorflow as tf
from tensorflow import keras
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from tensorflow.keras.preprocessing import image
print(tf.__version__)
```

```
base_dir = 'F:\\MASTER_2020\\MODULO_8_2020\\perros_gatos_5000'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directorio con las imagenes de training
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directorio con las imagenes de validation
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

# Directorio con las imagenes de test
test_cats_dir = os.path.join(test_dir, 'cats')
test_dogs_dir = os.path.join(test_dir, 'dogs')
```

```
print('total training cat images :', len(os.listdir(train_cats_dir)))
print('total training dog images :', len(os.listdir(train_dogs_dir)))

print('total validation cat images :', len(os.listdir(validation_cats_dir)))
print('total validation dog images :', len(os.listdir(validation_dogs_dir)))

print('total test cat images :', len(os.listdir(test_cats_dir)))
print('total test dog images :', len(os.listdir(test_dogs_dir)))
```

```
total training cat images : 1500
total training dog images : 1500
total validation cat images : 750
total validation dog images : 750
total test cat images : 250
total test dog images : 250
```

Modelo con *Data Augmentation*

```
from tensorflow.keras import Model

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

modelDA = Sequential()
modelDA.add(Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)))
modelDA.add(MaxPooling2D(2, 2))
modelDA.add(Conv2D(64, (3,3), activation='relu'))
modelDA.add(MaxPooling2D(2,2))
modelDA.add(Conv2D(128, (3,3), activation='relu'))
modelDA.add(MaxPooling2D(2,2))
modelDA.add(Conv2D(128, (3,3), activation='relu'))
modelDA.add(MaxPooling2D(2,2))
modelDA.add(Flatten())
modelDA.add(Dense(512, activation='relu'))
modelDA.add(Dense(1, activation='sigmoid'))

from tensorflow.keras.optimizers import RMSprop

modelDA.compile(loss='binary_crossentropy',
                 optimizer=RMSprop(lr=1e-4),
                 metrics=['acc'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen = ImageDataGenerator( rescale = 1.0/255. )
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                             batch_size=20, class_mode = 'binary', target_size = (150, 150))
```

```
test_generator = test_datagen.flow_from_directory(validation_dir, batch_size=20, class_mode = 'binary', target_size = (150, 150))
```

```
Found 3000 images belonging to 2 classes.  
Found 1500 images belonging to 2 classes.  
Found 1500 images belonging to 2 classes.
```

```
batch_size = 20  
steps_per_epoch = train_generator.n // batch_size  
validation_steps = validation_generator.n // batch_size  
  
historyDA = modelDA.fit(  
    train_generator,  
    steps_per_epoch= steps_per_epoch,  
    epochs= 100,  
    validation_data= validation_generator,  
    validation_steps= validation_steps,  
    verbose=2)
```

```
Epoch 90/100  
150/150 - 74s - loss: 0.3363 - acc: 0.8500 - val_loss: 0.4660 - val_acc: 0.8213  
Epoch 91/100  
150/150 - 74s - loss: 0.3341 - acc: 0.8540 - val_loss: 0.4159 - val_acc: 0.8320  
Epoch 92/100  
150/150 - 75s - loss: 0.3253 - acc: 0.8597 - val_loss: 0.3768 - val_acc: 0.8467  
Epoch 93/100  
150/150 - 74s - loss: 0.3236 - acc: 0.8550 - val_loss: 0.4314 - val_acc: 0.8340  
Epoch 94/100  
150/150 - 74s - loss: 0.3117 - acc: 0.8667 - val_loss: 0.5574 - val_acc: 0.8047  
Epoch 95/100  
150/150 - 74s - loss: 0.3166 - acc: 0.8583 - val_loss: 0.4543 - val_acc: 0.8127  
Epoch 96/100  
150/150 - 74s - loss: 0.3321 - acc: 0.8523 - val_loss: 0.4377 - val_acc: 0.8113  
Epoch 97/100  
150/150 - 74s - loss: 0.3215 - acc: 0.8703 - val_loss: 0.4041 - val_acc: 0.8347  
Epoch 98/100  
150/150 - 74s - loss: 0.3102 - acc: 0.8643 - val_loss: 0.8154 - val_acc: 0.7567  
Epoch 99/100  
150/150 - 74s - loss: 0.3195 - acc: 0.8637 - val_loss: 0.3886 - val_acc: 0.8487  
Epoch 100/100  
150/150 - 74s - loss: 0.3135 - acc: 0.8673 - val_loss: 0.5230 - val_acc: 0.8007
```

```
print (steps_per_epoch)  
print (validation_steps)  
test_lost, test_acc= modelDA.evaluate(test_generator)  
print ("Test Accuracy:", test_acc)
```

```
150  
75  
Test Accuracy: 0.8006667
```

```
import matplotlib.pyplot as plt  
acc = historyDA.history[ 'acc' ]  
val_acc = historyDA.history[ 'val_acc' ]  
loss = historyDA.history[ 'loss' ]  
val_loss = historyDA.history['val_loss' ]
```

```

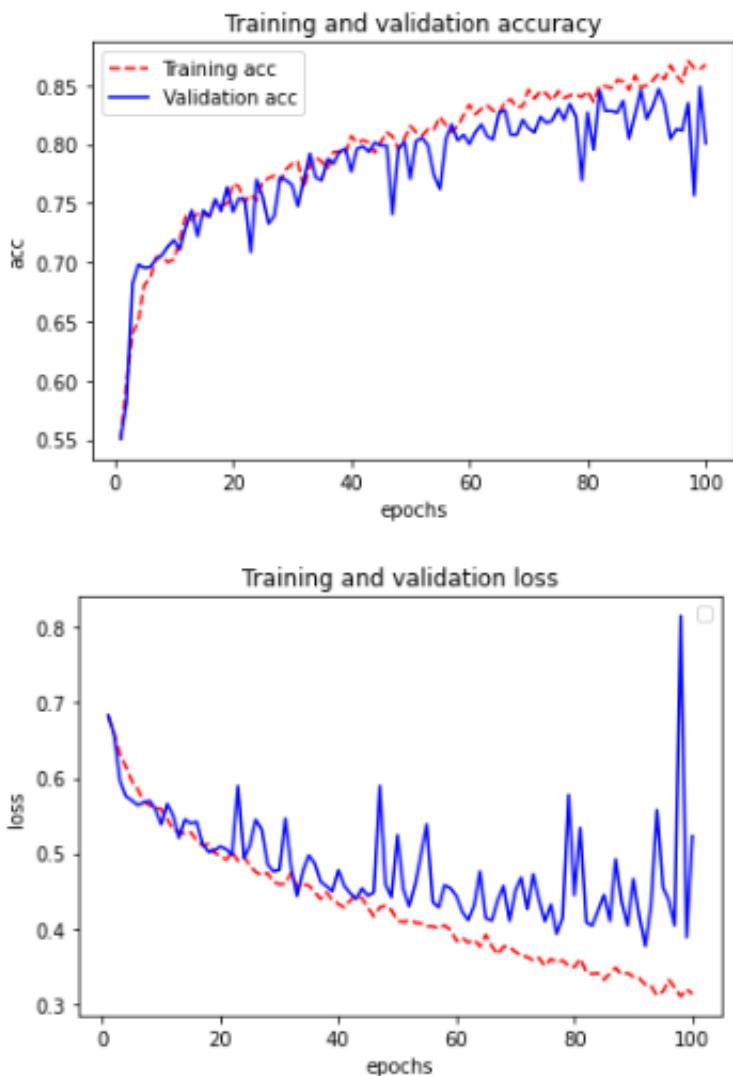
epochs = range(1,len(acc)+1,1) # obtener número de epochs

plt.plot ( epochs, acc, 'r--', label='Training acc' )
plt.plot ( epochs, val_acc, 'b', label='Validation acc')
plt.title ('Training and validation accuracy')
plt.ylabel('acc')
plt.xlabel('epochs')
plt.legend()
plt.figure()

plt.plot ( epochs, loss, 'r--' )
plt.plot ( epochs, val_loss , 'b' )
plt.title ('Training and validation loss' )
plt.ylabel('loss')
plt.xlabel('epochs')

plt.legend()
plt.figure()

```



Modelo con Feature Extraction (Modelo preentrenado)

```
from tensorflow.keras.applications import VGG16  
  
pre_trained_model = VGG16(input_shape = (150, 150, 3),  
                           include_top = False,  
                           weights = 'imagenet')
```

```
pre_trained_model.summary()
```

```
for layer in pre_trained_model.layers:  
    layer.trainable = False
```

```
pre_trained_model.summary()
```

```
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688

Trainable params: 0

Non-trainable params: 14,714,688

La diferencia es que ahora no hay que entrenar los parámetros

```
from tensorflow.keras import Model

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
modelFE = Sequential()
modelFE.add(pre_trained_model)
modelFE.add(Flatten())

modelFE.add(Dense(256, activation='relu'))
modelFE.add(Dense(1, activation='sigmoid'))
modelFE.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_5 (Flatten)	(None, 8192)	0
dense_10 (Dense)	(None, 256)	2097408
dense_11 (Dense)	(None, 1)	257

Total params: 16,812,353

Trainable params: 2,097,665

Non-trainable params: 14,714,688

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

```
validation_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen = ImageDataGenerator( rescale = 1.0/255. )
```

```
train_generator = train_datagen.flow_from_directory(train_dir,
batch_size=20,
class_mode='binary',
target_size=(150, 150))
```

```
validation_generator = validation_datagen.flow_from_directory(validation_dir,
batch_size=20,
class_mode = 'binary',
target_size = (150, 150))
```

```
test_generator = test_datagen.flow_from_directory(validation_dir,
batch_size=20,
class_mode = 'binary',
```

target_size = (150, 150))

```
Found 3000 images belonging to 2 classes.
Found 1500 images belonging to 2 classes.
Found 1500 images belonging to 2 classes.
```

```

from tensorflow.keras.optimizers import RMSprop

modelFE.compile(loss='binary_crossentropy',
                 optimizer=RMSprop(lr=1e-4),
                 metrics=['acc'])

batch_size = 20
steps_per_epoch = train_generator.n // batch_size
validation_steps = validation_generator.n // batch_size

```

```

historyFE = modelFE.fit(
    train_generator,
    validation_data = validation_generator,
    steps_per_epoch = steps_per_epoch,
    epochs = 100,
    validation_steps = validation_steps,
    verbose = 2)

```

```

Epoch 90/100
150/150 - 262s - loss: 0.1852 - acc: 0.9253 - val_loss: 0.2858 - val_acc: 0.9133
Epoch 91/100
150/150 - 268s - loss: 0.1793 - acc: 0.9277 - val_loss: 0.2893 - val_acc: 0.9173
Epoch 92/100
150/150 - 264s - loss: 0.1841 - acc: 0.9267 - val_loss: 0.2743 - val_acc: 0.9147
Epoch 93/100
150/150 - 263s - loss: 0.1863 - acc: 0.9270 - val_loss: 0.2826 - val_acc: 0.9147
Epoch 94/100
150/150 - 263s - loss: 0.1692 - acc: 0.9327 - val_loss: 0.2901 - val_acc: 0.9160
Epoch 95/100
150/150 - 262s - loss: 0.1940 - acc: 0.9267 - val_loss: 0.2842 - val_acc: 0.9147
Epoch 96/100
150/150 - 262s - loss: 0.1850 - acc: 0.9283 - val_loss: 0.2623 - val_acc: 0.9193
Epoch 97/100
150/150 - 262s - loss: 0.1709 - acc: 0.9373 - val_loss: 0.3079 - val_acc: 0.9127
Epoch 98/100
150/150 - 262s - loss: 0.1661 - acc: 0.9390 - val_loss: 0.2583 - val_acc: 0.9253
Epoch 99/100
150/150 - 262s - loss: 0.1868 - acc: 0.9257 - val_loss: 0.2614 - val_acc: 0.9253
Epoch 100/100
150/150 - 262s - loss: 0.1826 - acc: 0.9287 - val_loss: 0.3055 - val_acc: 0.9073

```

```

test_lost, test_acc= modelFE.evaluate(test_generator)

print ("Test Accuracy:", test_acc)

```

Test Accuracy: 0.9073333

```

import matplotlib.pyplot as plt

acc      = historyFE.history[      'acc'  ]
val_acc  = historyFE.history[ 'val_acc' ]
loss     = historyFE.history[      'loss'  ]
val_loss = historyFE.history['val_loss']

epochs   = range(1,len(acc)+1,1) # obtener número de epochs

```

```

plt.plot(epochs, acc, 'r--', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('acc')
#plt.ylim(0,1)
plt.xlabel('epochs')

plt.legend()
plt.figure()

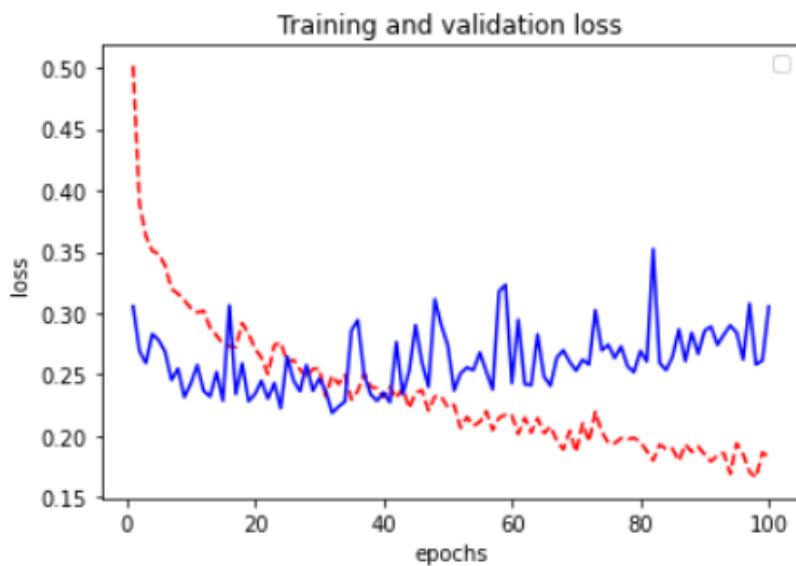
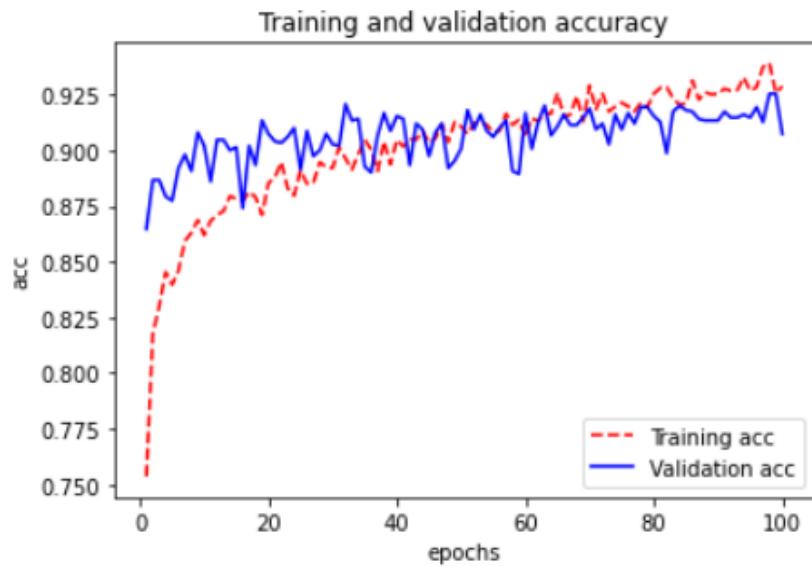
```

```

plt.plot(epochs, loss, 'r--')
plt.plot(epochs, val_loss, 'b')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epochs')
#plt.ylim(0,1)

plt.legend()
plt.figure()

```



ModelFT : Modelo con *Fine Tuning*

```
from tensorflow.keras.applications import VGG16
pre_trained_model = VGG16(input_shape = (150, 150, 3),
                           include_top = False,
                           weights = 'imagenet')
pre_trained_model.trainable = True
set_trainable = False
for layer in pre_trained_model.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
pre_trained_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_4 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0

block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 7,079,424		
Non-trainable params: 7,635,264		

```

modelFT = Sequential()
modelFT.add(pre_trained_model)
modelFT.add(Flatten())
modelFT.add(Dense(256, activation='relu'))
modelFT.add(Dense(1, activation='sigmoid'))

```

```
modelFT.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_3 (Flatten)	(None, 8192)	0
dense_6 (Dense)	(None, 256)	2097408
dense_7 (Dense)	(None, 1)	257
=====		
Total params: 16,812,353		
Trainable params: 9,177,089		
Non-trainable params: 7,635,264		

```

from tensorflow.keras.optimizers import RMSprop
modelFT.compile(loss='binary_crossentropy',
                 optimizer=RMSprop(lr=1e-4),
                 metrics=['acc'])

from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator( rescale = 1.0/255. )

test_datagen = ImageDataGenerator( rescale = 1.0/255. )

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                               batch_size=20,
                                                               class_mode = 'binary',
                                                               target_size = (150, 150))

test_generator = test_datagen.flow_from_directory(validation_dir,
                                                 batch_size=20,
                                                 class_mode = 'binary',
                                                 target_size = (150, 150))

```

Found 3000 images belonging to 2 classes.
 Found 1500 images belonging to 2 classes.
 Found 1500 images belonging to 2 classes.

```

batch_size = 20
steps_per_epoch = train_generator.n // batch_size
validation_steps = validation_generator.n // batch_size
historyFT = modelFT.fit(
    train_generator,
    validation_data = validation_generator,
    steps_per_epoch = steps_per_epoch,
    epochs = 100,
    validation_steps = validation_steps,
    verbose = 2)

```

```

Epoch 90/100
150/150 - 319s - loss: 0.0539 - acc: 0.9907 - val_loss: 1.9008 - val_acc: 0.9267
Epoch 91/100
150/150 - 319s - loss: 0.0412 - acc: 0.9897 - val_loss: 1.5064 - val_acc: 0.9420
Epoch 92/100
150/150 - 319s - loss: 0.0389 - acc: 0.9897 - val_loss: 1.4866 - val_acc: 0.9433
Epoch 93/100
150/150 - 319s - loss: 0.0617 - acc: 0.9867 - val_loss: 1.2377 - val_acc: 0.9493
Epoch 94/100
150/150 - 319s - loss: 0.0765 - acc: 0.9890 - val_loss: 1.2892 - val_acc: 0.9433
Epoch 95/100
150/150 - 330s - loss: 0.0405 - acc: 0.9920 - val_loss: 1.7324 - val_acc: 0.9427
Epoch 96/100
150/150 - 330s - loss: 0.0833 - acc: 0.9883 - val_loss: 0.5520 - val_acc: 0.9467
Epoch 97/100
150/150 - 319s - loss: 0.0360 - acc: 0.9893 - val_loss: 1.0298 - val_acc: 0.9473
Epoch 98/100
150/150 - 319s - loss: 0.0694 - acc: 0.9860 - val_loss: 0.8731 - val_acc: 0.9513
Epoch 99/100
150/150 - 321s - loss: 0.0474 - acc: 0.9893 - val_loss: 1.0240 - val_acc: 0.9493
Epoch 100/100
150/150 - 319s - loss: 0.0335 - acc: 0.9913 - val_loss: 1.3415 - val_acc: 0.9500

```

```

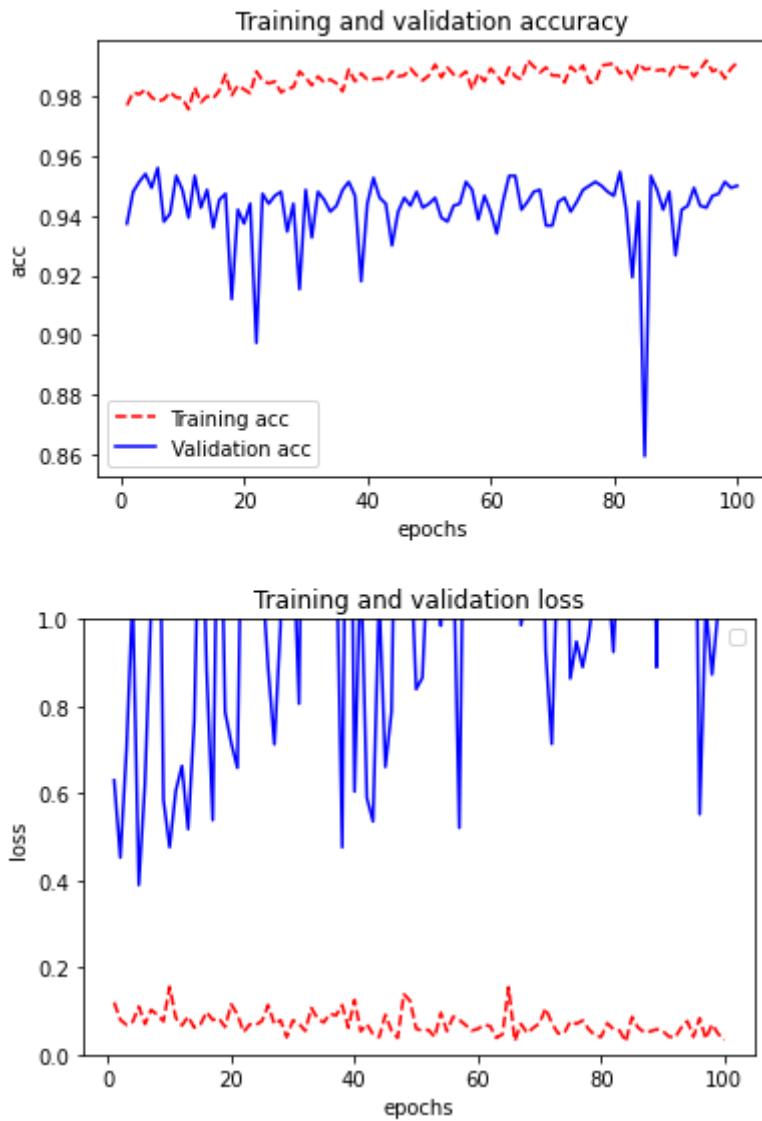
import matplotlib.pyplot as plt

acc      = historyFT.history[ 'acc' ]
val_acc = historyFT.history[ 'val_acc' ]
loss     = historyFT.history[ 'loss' ]
val_loss = historyFT.history['val_loss']

epochs   = range(1,len(acc)+1,1) # obtener número de epochs
plt.plot ( epochs, acc, 'r--', label='Training acc' )
plt.plot ( epochs, val_acc, 'b', label='Validation acc' )
plt.title ('Training and validation accuracy')
plt.ylabel('acc')
# plt.ylim(0,1)
plt.xlabel('epochs')
plt.legend()
plt.figure()
plt.plot ( epochs, loss, 'r--' )

```

```
plt.plot ( epochs, val_loss , 'b' )
plt.title ('Training and validation loss' )
plt.ylabel('loss')
plt.xlabel('epochs')
plt.ylim(0,1)
plt.legend()
plt.figure()
```



```
test_lost, test_acc= modelFT.evaluate(test_generator)
print ("Test Accuracy:", test_acc)
```

Test Accuracy: 0.95

```

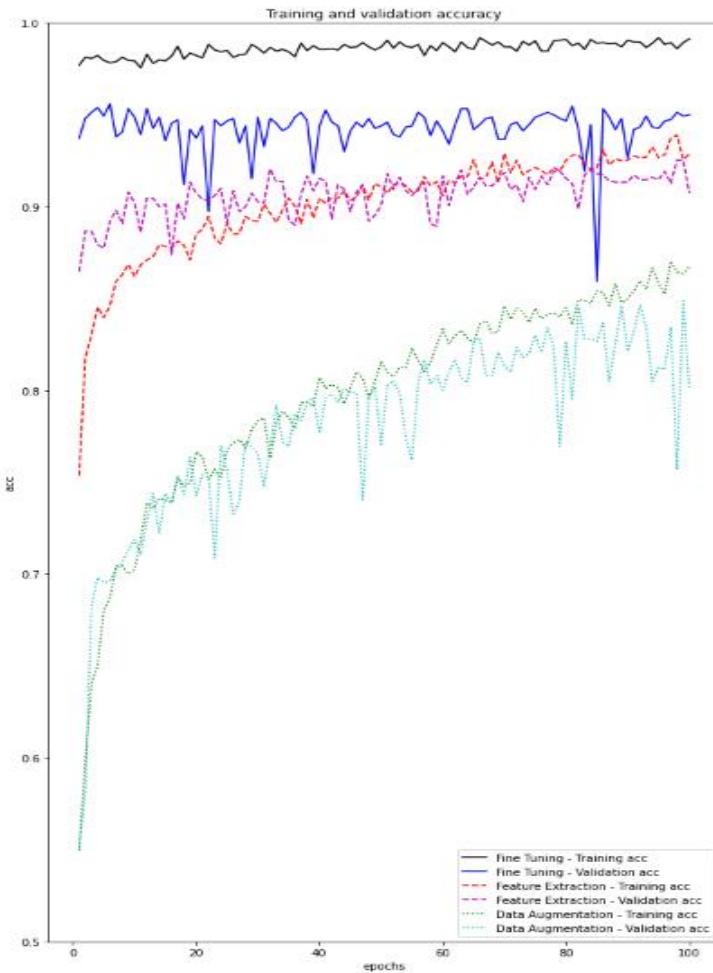
accDA    = historyDA.history[    'acc' ]
val_accDA = historyDA.history[ 'val_acc' ]

accFE    = historyFE.history[    'acc' ]
val_accFE = historyFE.history[ 'val_acc' ]
accFT    = historyFT.history[    'acc' ]
val_accFT = historyFT.history[ 'val_acc' ]
epochs   = range(1,len(accDA)+1,1)

plt.figure(figsize=(10,18))
plt.plot ( epochs,    accFT, 'k', label='Fine Tuning - Training acc ' )
plt.plot ( epochs, val_accFT, 'b', label='Fine Tuning - Validation acc ')
plt.plot ( epochs,    accFE, 'r--', label='Feature Extraction - Training acc' )
plt.plot ( epochs, val_accFE, 'm--', label='Feature Extraction - Validation acc')

plt.plot ( epochs,    accDA, 'g:', label='Data Augmentation - Training acc' )
plt.plot ( epochs, val_accDA, 'c:', label='Data Augmentation - Validation acc')
plt.title ('Training and validation accuracy')
plt.ylabel('acc')
plt.ylim(0.5,1)
plt.xlabel('epochs')
plt.legend()
plt.figure()

```



```

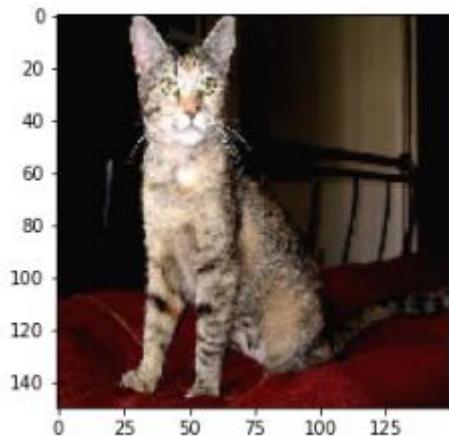
import numpy as np
from tensorflow.keras.preprocessing import image

path='F:\\MASTER_2020\\MODULO_8_2020\\perros_gatos_5000\\test\\cats\\cat.10000.jpg'
img=image.load_img(path, target_size=(150, 150))

x=image.img_to_array(img)
image=np.expand_dims(x, axis=0)
classes = modelFT.predict(image)
print(classes)

plt.imshow(img)
plt.show()
if classes>0: print("cat.10000.jpg IS A DOG")
else: print( "cat.10000.jpg IS A CAT")

```



cat.10000.jpg IS A CAT

7.4.7.2. Código de R

En esta parte de código de R sólo desarrollamos dos ejemplos sencillos. El primero es una red neuronal densamente conectada con datos de MNIST y, el segundo, es un ejemplo de red convolucional y Data Augmentation con la base de datos CIFAR10.

Estos ejemplos más desarrollados y arquitecturas más complejas se encuentran en la documentación anexa de Deep learning que acompañamos a este módulo.

Ejemplo de red neuronal densamente conectada con la base de datos MNIST

```
# ??keras
library(keras)
# install_keras()

mnist<-dataset_mnist()
x_train<-mnist$train$x
y_train<-mnist$train$y
x_test<-mnist$test$x
y_test<-mnist$test$y

# reshape
x_train<-array_reshape(x_train, c(nrow(x_train), 784))
x_test<-array_reshape(x_test, c(nrow(x_test), 784))

# rescale
x_train<-x_train/ 255
x_test<-x_test/ 255
y_train<-to_categorical(y_train, 10)
y_test<-to_categorical(y_test, 10)

model<-keras_model_sequential()
model%>%
layer_dense(units= 256, activation= 'relu', input_shape= c(784)) %>%
layer_dropout(rate= 0.4) %>%
layer_dense(units= 128, activation= 'relu') %>%
layer_dropout(rate= 0.3) %>%
layer_dense(units= 10, activation= 'softmax')
summary(model)

model%>% compile(
loss= 'categorical_crossentropy',
optimizer= optimizer_rmsprop(),
metrics= c('accuracy')
)

history <-model%>% fit(x_train, y_train, epochs= 30, batch_size= 128, validation_split= 0.2)
plot(history)

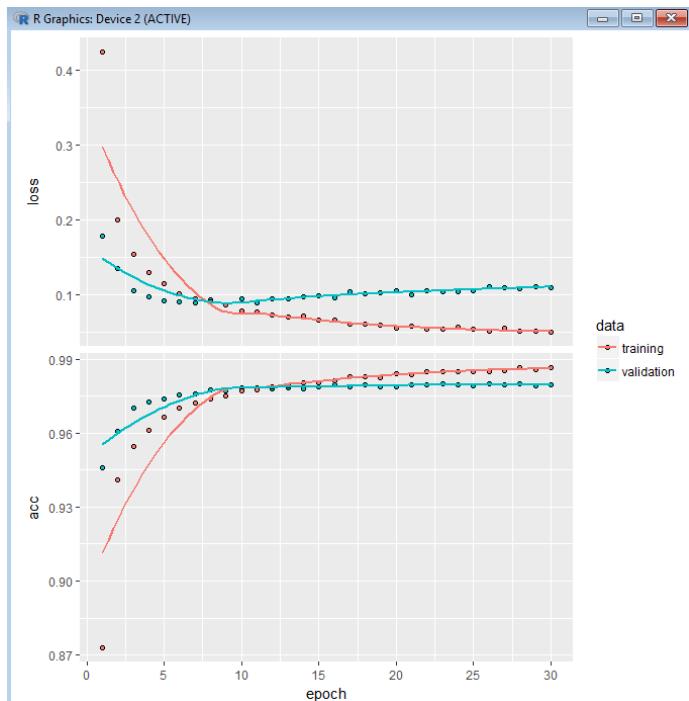
model%>% evaluate(x_test, y_test)
model%>% predict_classes(x_test)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [=====] - 3s 62us/step - loss: 0.4246 - acc: 0.8728 - val_loss: 0.1789 - val_acc: 0.9458
Epoch 2/30
48000/48000 [=====] - 2s 48us/step - loss: 0.2000 - acc: 0.9410 - val_loss: 0.1351 - val_acc: 0.9607
Epoch 3/30
48000/48000 [=====] - 2s 47us/step - loss: 0.1546 - acc: 0.9546 - val_loss: 0.1060 - val_acc: 0.9702
Epoch 4/30
48000/48000 [=====] - 2s 48us/step - loss: 0.1299 - acc: 0.9612 - val_loss: 0.0977 - val_acc: 0.9725
Epoch 5/30
48000/48000 [=====] - 2s 49us/step - loss: 0.1148 - acc: 0.9665 - val_loss: 0.0924 - val_acc: 0.9738
Epoch 6/30
48000/48000 [=====] - 2s 49us/step - loss: 0.1015 - acc: 0.9702 - val_loss: 0.0917 - val_acc: 0.9753
Epoch 7/30
48000/48000 [=====] - 2s 48us/step - loss: 0.0957 - acc: 0.9722 - val_loss: 0.0902 - val_acc: 0.9759
Epoch 8/30
48000/48000 [=====] - 2s 48us/step - loss: 0.0917 - acc: 0.9739 - val_loss: 0.0943 - val_acc: 0.9776
Epoch 9/30
48000/48000 [=====] - 2s 49us/step - loss: 0.0872 - acc: 0.9751 - val_loss: 0.0874 - val_acc: 0.9772
Epoch 10/30
48000/48000 [=====] - 2s 49us/step - loss: 0.0789 - acc: 0.9770 - val_loss: 0.0947 - val_acc: 0.9784
```

```
> model %>% predict_classes(x_test)
 [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3 5 1
 [48] 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0 7 0 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3
 [95] 1 4 1 7 6 9 6 0 5 4 9 9 2 1 9 4 8 7 3 9 7 4 4 9 2 5 4 7 6 7 9 0 5 8 5 6 6 5 7 8 1 0 1 6 4 6
[142] 7 3 1 7 1 8 2 0 2 9 9 5 5 1 5 6 0 3 4 4 6 5 4 6 5 4 5 1 4 4 7 2 3 2 7 1 8 1 8 1 8 5 0 8 9 2 5
[189] 0 1 1 1 0 9 0 3 1 6 4 2 3 6 1 1 1 3 9 5 2 9 4 5 9 3 9 0 3 6 5 5 7 2 2 7 1 2 8 4 1 7 3 3 8 8 7
[236] 9 2 2 4 1 5 9 8 7 2 3 0 2 4 2 4 1 9 5 7 7 2 8 2 0 8 5 7 7 9 1 8 1 8 0 3 0 1 9 9 4 1 8 2 1 2 9
[283] 7 5 9 2 6 4 1 5 8 2 9 2 0 4 0 0 2 8 4 7 1 2 4 0 2 7 4 3 3 0 0 3 1 9 6 5 2 5 8 7 9 3 0 4 2 0 7
[330] 1 1 2 1 5 3 3 9 7 8 6 3 6 1 3 8 1 0 5 1 3 1 5 5 6 1 8 5 1 7 9 4 6 2 2 5 0 6 5 6 3 7 2 0 8 8 5
[377] 4 1 1 4 0 7 3 7 6 1 6 2 1 9 2 8 6 1 9 5 2 5 4 4 2 8 3 8 2 4 5 0 3 1 7 7 5 7 9 7 1 9 2 1 4 2 9
[424] 2 0 4 9 1 4 8 1 8 4 5 9 8 8 3 7 6 0 0 3 0 2 0 6 9 9 3 3 3 2 3 9 1 2 6 8 0 5 6 6 6 6 3 8 8 2 7 5
[471] 8 9 6 1 8 4 1 2 5 9 1 9 7 5 4 0 8 9 9 1 0 5 2 3 7 2 9 4 0 6 3 9 5 2 1 3 1 3 6 5 7 4 2 2 6 3 2
[519] 5 4 8 0 7 1 3 0 3 8 3 1 0 3 4 4 5 4 2 1 8 2 5 4 8 8 4 0 0 2 3 2 7 1 0 8 7 4 4 7 0 6 0 0 0 0 0
```

```
> plot(history)
```

```
> model %>% evaluate(x_test, y_test)
```



\$acc

[1] 0.9806

Ejemplo de R con red convolucional y Data Aumentation con la base de datos CIFAR

```
library(keras)
```

```
# Parámetros de la red
```

```
batch_size <- 32  
epochs <- 100  
data_augmentation <- TRUE
```

```
# Preparación de los datos
```

```
cifar10 <- dataset_cifar10()
```

```
# Escalado y preparación de las imágenes
```

```
x_train <- cifar10$train$x/255  
x_test <- cifar10$test$x/255  
y_train <- to_categorical(cifar10$train$y, num_classes = 10)  
y_test <- to_categorical(cifar10$test$y, num_classes = 10)
```

```
# Estructura del modelo
```

```
#model <- keras_model_sequential()  
model %>%
```

```
# Red convolucional de dos capas con imágenes de 32 x 32 píxeles
```

```
# Primera capa
```

```
layer_conv_2d(  
  filter = 32, kernel_size = c(3,3), padding = "same",  
  input_shape = c(32, 32, 3)  
) %>%  
layer_activation("relu") %>%
```

```
# Segunda capa oculta
```

```
layer_conv_2d(filter = 32, kernel_size = c(3,3)) %>%  
layer_activation("relu") %>%
```

```
# Uso de max pooling
```

```
layer_max_pooling_2d(pool_size = c(2,2)) %>%  
layer_dropout(0.25) %>%
```

```
# Una capa adicional más
```

```
layer_conv_2d(filter = 32, kernel_size = c(3,3), padding = "same") %>%  
layer_activation("relu") %>%  
layer_conv_2d(filter = 32, kernel_size = c(3,3)) %>%  
layer_activation("relu") %>%
```

```
# Max Pooling
```

```
layer_max_pooling_2d(pool_size = c(2,2)) %>%  
layer_dropout(0.25) %>%  
layer_flatten() %>%  
layer_dense(512) %>%  
layer_activation("relu") %>%  
layer_dropout(0.5) %>%  
layer_dense(10) %>%  
layer_activation("softmax")  
opt <- optimizer_rmsprop(lr = 0.0001, decay = 1e-6)  
model %>% compile(  
  loss = "categorical_crossentropy",  
  optimizer = opt,  
  metrics = "accuracy"  
)
```

```
# Entrenamiento de la red
```

```
if(!data_augmentation){  
  model %>% fit(  
    x_train, y_train,  
    batch_size = batch_size,  
    epochs = epochs,  
    validation_data = list(x_test, y_test),
```

```

shuffle = TRUE
)
} else {
  datagen <- image_data_generator(
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip = TRUE
)
  datagen %>% fit_image_data_generator(x_train)
  model %>% fit_generator(
    flow_images_from_data(x_train, y_train, datagen, batch_size = batch_size),
    steps_per_epoch = as.integer(50000/batch_size),
    epochs = epochs,
    validation_data = list(x_test, y_test)
)
}

```

```

Epoch 95/100
10000/10000 [=====] - 15s/lms/sample - loss: 0.9596 - acc: 0.6697
1563/1563 [=====] - 243s 156ms/step - loss: 1.0510 - acc: 0.6359 - val_loss: 0.9595 - val_acc: 0.6697
Epoch 96/100
10000/10000 [=====] - 15s/lms/sample - loss: 1.1087 - acc: 0.6219
1563/1563 [=====] - 244s 156ms/step - loss: 1.0458 - acc: 0.6401 - val_loss: 1.1087 - val_acc: 0.6219
Epoch 97/100
10000/10000 [=====] - 15s/lms/sample - loss: 0.9726 - acc: 0.6707
1563/1563 [=====] - 244s 156ms/step - loss: 1.0547 - acc: 0.6375 - val_loss: 0.9728 - val_acc: 0.6707
Epoch 98/100
10000/10000 [=====] - 15s/lms/sample - loss: 0.8799 - acc: 0.6998
1563/1563 [=====] - 244s 156ms/step - loss: 1.0555 - acc: 0.6363 - val_loss: 0.8800 - val_acc: 0.6998
Epoch 99/100
10000/10000 [=====] - 15s/lms/sample - loss: 0.9140 - acc: 0.6890
1563/1563 [=====] - 246s 157ms/step - loss: 1.0606 - acc: 0.6350 - val_loss: 0.9143 - val_acc: 0.6890
Epoch 100/100
10000/10000 [=====] - 15s/lms/sample - loss: 1.1925 - acc: 0.6225
1563/1563 [=====] - 244s 156ms/step - loss: 1.0587 - acc: 0.6358 - val_loss: 1.1924 - val_acc: 0.6225

```

7.4.8. Ejemplo de clasificación de textos con Conv1D en Python

Las redes convolucionales son actualmente utilizadas para diferentes propósitos: tratamiento de imágenes (visión por computador, extracción de características, segmentación, etc.) generación y clasificación de texto (o audio), predicción de series temporales, etc. En este caso, veremos en detalle un ejemplo de clasificación de texto.

Se presenta a continuación una aplicación práctica de clasificación de texto multiclasificación a partir de redes Convolucionales de una dimensión. Para ello, se utiliza una bbdd referida a las reclamaciones de los usuarios ante una entidad bancaria en función del tipo de producto.

En primer lugar, se importan las librerías a utilizar y se lee el fichero:

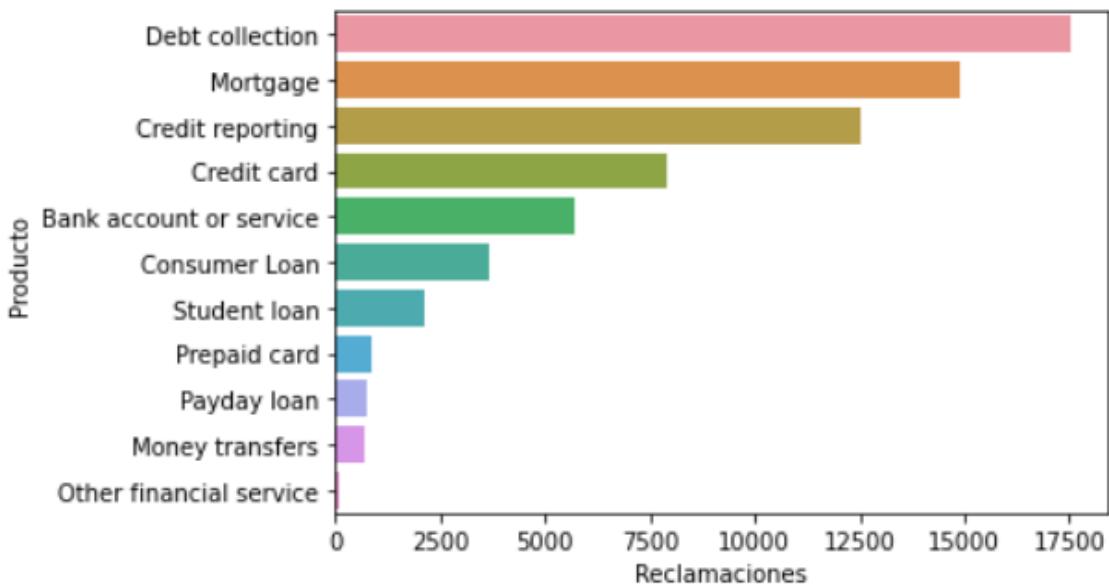
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import warnings
warnings.filterwarnings('ignore')
```

El fichero de trabajo contiene una serie de reclamaciones que no vienen acompañadas con su texto asociado. Se considera que lo más adecuado es excluir tales instancias del dataset de partida.

```
datos = pd.read_csv('C:/DEEP LEARNING/consumer_complaints.csv')
datos = datos[['product', 'consumer_complaint_narrative']] # variables de interés
datos =
datos.dropna(subset=['product', 'consumer_complaint_narrative']).reset_index
(drop=True) # registros con texto no informado son eliminados de la muestra
print('Tamaño de los datos:', datos.shape)
Tamaño de los datos: (66806, 2)
sns.countplot(y='product', data=datos, order =
datos['product'].value_counts().index)
plt.xlabel('Reclamaciones'), plt.ylabel('Producto')
plt.show()
```



Como puede verse, se parte de once tipos de productos diferentes; si bien, para varios de ellos el número de reclamaciones no es considerado significativo por el área legal de la entidad. Por ello,

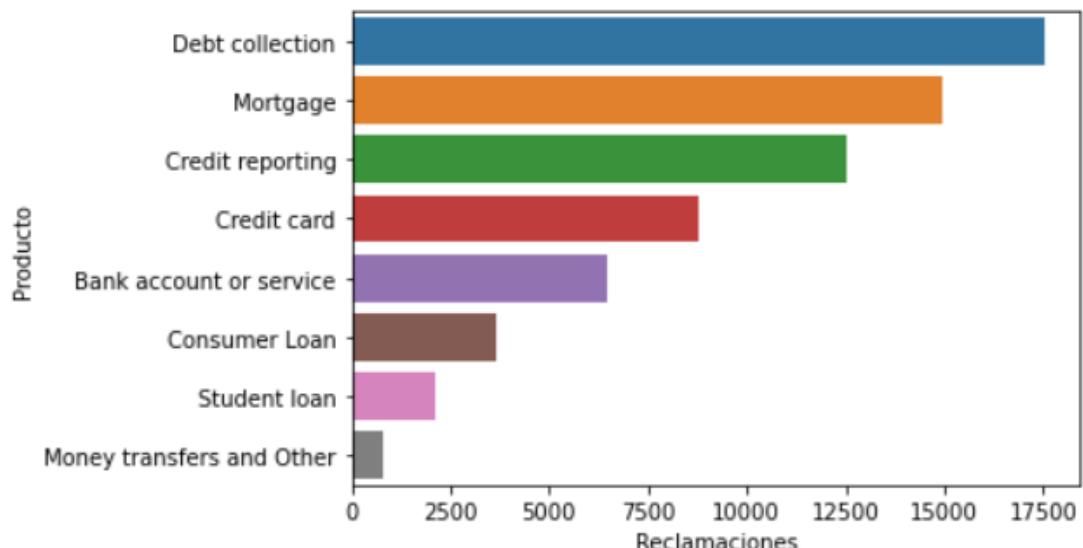
y en base a la similitud de los productos, se agrupan las cuatro categorías con un menor número reclamaciones en

- Prepaid card: se incluye en la categoría de "Credit card"
- Payday loan: se incluye en la categoría "Bank account or service"
- Money transfers y Other financial service: forman un grupo conjunto denominado "Money transfers and Other financial service"

```
# agrupaciones
datos['product'] = np.where(datos['product']=='Payday loan', 'Bank account or service', datos['product']) # préstamos
datos['product'] = np.where(datos['product']=='Prepaid card', 'Credit card', datos['product']) # créditos

tipo_producto = ['Money transfers', 'Other financial service'] # otros servicios financieros
datos['product'] = np.where(datos['product'].isin(tipo_producto), 'Money transfers and Other', datos['product'])

sns.countplot(y='product', data=datos, order =
datos['product'].value_counts().index)
plt.xlabel('Reclamaciones'), plt.ylabel('Producto')
plt.show()
```



De esta forma, el número de grupos ha sido distribuido de una forma más equitativa. A modo de ejemplo, se muestra una de las reclamaciones:

```
def plot_reclamaciones(df, elemento):
    df = df.loc[elemento].to_list()
    return df
print('Producto:', plot_reclamaciones(datos, 100)[0])
print('Reclamacion:', plot_reclamaciones(datos, 100)[1])
```

Producto: Consumer Loan

Reclamacion: I contacted Drivetime on several different occasions regarding a car loan I had with them account # XXXX i do not have the number for the other loan that was removed also from the credit bureau with a positive credit rating that was closed and paid in full. The company insisted on avoiding an agreement entered into by myself and XXXX of their company representative in a corporate position they have the agreed upon resolution and can provide your agency with a copy of the same because i do not have the capability to do so with my computer. Drivetime during the month of XXXX XXXX removed my credit reporting information from XXXX credit bureaus reporting agencies i believe XXXX and XXXX XXXX I discovered in or about XXXX XXXX that my positive credit rating s were no longer in the XXXX and XXXX credit bureaus and that another positive account for another car loan was no longer reporting in my credit file. these removals adversely affected my credit ratings and as such my ability to purchase something with a lower interest rate. I have never missed a payment to the Drivetime company on my car loans inclusive of XXXX XXXX XXXX XXXX different auto loans and for them to destroy my credit ratings is despicable. I spoke with XXXX today (XXXX/XXXX/XXXX) regarding those issues and his response to me was that Drivetime is now in the process of updating all consumers credit files and all of the consumers credit files were removed from the credit bureaus and is being updated however he does not know how the process is being performed who's records are being updated corrected and what process or timeframes this process will take and furthermore does the process involves all consumers credit reports or are they updating reports for existing customers again which would not apply to me because my loans have been paid off and are closed however they did the same thing to my credit report therefore the process would leave me with no credit ratings for accounts that i have paid off in good faith and never missed a payment to do so with my computer. they kept trying to keep my credit rating up and for those reasons i am submitting a complaint to the consumer financial protection bureau based on the ruling implemented in terms of enforcement against the drivetime XXXX for their violations of consumer rights dated XXXX XXXX, XXXX which is what they did to me in XXXX XXXX XXXX and prior to those dates in XXXX.

Como puede verse, la reclamación 101 del dataset está asociada a un préstamo al consumo. Sin embargo, lo verdaderamente interesante del texto de ejemplo es la necesidad de realizar un preprocesado a los textos puesto que algunos símbolos, caracteres o, incluso palabras, no son relevantes para que la red sea capaz de interpretar el contenido del mismo. Por tanto, se lleva a cabo lo siguiente:

- Conversión del texto a minúsculas
- Exclusión del texto el contenido cifrado (XXXX)
- Eliminación de caracteres extraños
- Para poder hacer este preprocesado de textos se hace uso del paquete **re** de Python.

```
def preprocesado(reclamacion):  
    reclamacion = reclamacion.lower() # texto en minúsculas  
    reclamacion = reclamacion.replace('x','') # cambio X por espacio  
    reclamacion = re.compile('[(/){}][\\]|@;]').sub('', reclamacion) #  
    simbolos extraños (1)  
    reclamacion = re.compile('[^0-9a-z #+_]').sub('', reclamacion) #  
    simbolos extraños (2)  
    return reclamacion  
  
datos['consumer_complaint_narrative'] =  
datos['consumer_complaint_narrative'].apply(preprocesado) # aplicación de  
la función
```

Se presenta de nuevo el ejemplo anterior para ver el resultado del procesamiento de textos realizado.

```
print('Producto:', plot_reclamaciones(datos, 100)[0])  
print('Reclamacion:', plot_reclamaciones(datos, 100)[1])
```

Producto: Consumer Loan

Reclamacion: i contacted drivetime on several different occasions regarding a car loan i had with them account # i do not have the number for the other loan that was removed also from the credit bureau with a positive credit rating that was closed and paid in full the company insisted on avoiding an agreement entered into by myself and of their company representative in a corporate position they have the agreed upon resolution and can provide your agency with a copy of the same because i do not have the capability to do so with my computer drivetime during the month of re moved my credit reporting information from credit bureaus reporting agencies i believe and i discovered in or about that my positive credit ratings were no longer in the and credit bureaus and that another positive account for another car loan was no longer reporting in my credit file these removals adversely affected my credit ratings and as such my ability to purchase something with a lower interest rate i have never missed a payment to the drivetime company on my car loans inclusive of different auto loans and for them to destroy my credit ratings is despicable i spoke with today regarding those issues and his response to me was that drivetime is now in the process of updating all consumers credit files and all of the consumers credit files were removed from the credit bureaus and is being updated however he does not know how the process is being performed who's records are being updated corrected and what process or timeframes this process will take and furthermore does the process involves all consumers credit reports or are they updating reports for existing customers again which would not apply to me because my loans have been paid off and are closed however they did the same thing to my credit report therefore the process would leave me with no credit ratings for accounts that i have paid off in good faith and never missed a payment to do so with my computer they kept trying to keep my credit rating up and for those reasons i am submitting a complaint to the consumer financial protection bureau based on the ruling implemented in terms of enforcement against the drivetime for their violations of consumer rights dated which is what they did to me in and prior to those dates in

Ahora, como cualquier proyecto de ciencia de datos, se procede con la partición de los datos (entrenamiento y test - 80 y 20%, respectivamente) y se convierte cada categoría del target como variable dummy.

```

seed=123
tf.random.set_seed(seed)
np.random.seed(seed)

X_texto = datos['consumer_complaint_narrative']
Y_label = pd.get_dummies(datos['product']).values # las categorías son
convertidas a variable dummy

X_train_text, X_test_text, Y_train, Y_test =
train_test_split(X_texto,Y_label, test_size = 0.2, random_state = seed)
print('Entrenamiento:', X_train_text.shape)
print('Test:', X_train_text.shape)
Entrenamiento: (53444,
Test: (53444,

```

Antes de definir la arquitectura de la red, se lleva a cabo la conversión del texto a variables numéricas que es el input que puede leer la red. Para ello, se realiza:

- La vectorización del texto asociado a las reclamaciones.
- El truncamiento y relleno de las secuencias de entrada para igualar la longitud en la modelización.

```

MAX_NB_WORDS = 25000 # frecuencia de palabras
MAX_SEQUENCE_LENGTH = 200 # número de palabras en cada reclamacion
EMBEDDING_DIM = 150 # dimensión del embedding

tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=MAX_NB_WORDS,
filters='!"#$%&()*+,-./:;=>?@[\]^`{|}~', lower=True)
tokenizer.fit_on_texts(X_train_text.values)
word_index = tokenizer.word_index
print('Tokens:', len(word_index))

X_train = tokenizer.texts_to_sequences(X_train_text.values)
X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train,
 maxlen=MAX_SEQUENCE_LENGTH)
print('Datos de entrada:', X_train.shape)
Tokens: 57245
Datos de entrada: (53444, 200)

```

Por último, se crea la red neuronal siguiendo el método funcional. La red tiene las siguientes capas:

- Entrada: de 200 neuronas pues corresponde con la longitud de las secuencias
- Embedding: de dimensión 200 y toma como input el número máximo de palabras (25.000)
- Convolucional: de 64 neuronas
- MaxPooling:
- Densa: de 32 neuronas y con función de activación "relu"
- Salida: capa densa con 8 neuronas (número de categorías del target) y función de activación "softmax"

```

# capa de entrada
inputs = tf.keras.Input(shape=(X_train.shape[1],))
embedding = tf.keras.layers.Embedding(input_dim=MAX_NB_WORDS,
                                      output_dim=EMBEDDING_DIM)(inputs)
capa_conv = tf.keras.layers.Conv1D(filters=64,
                                   kernel_size=3,
                                   padding='valid',
                                   activation='relu')(embedding)
max_pooling = tf.keras.layers.GlobalMaxPooling1D()(capa_conv)
capa_densa = tf.keras.layers.Dense(units=32,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01))(max_pooling)
out = tf.keras.layers.Dense(units=Y_train.shape[1],
                            activation='softmax')(capa_densa)
modelo = tf.keras.Model(inputs=inputs, outputs=out)

```

El summary nos muestra el número de parámetros por capa y el número de parámetros total. Puede verse que el alto número de parámetros viene, principalmente, por la capa de Embedding.

```
modelo.summary()
```

```

Model: "model_2"

Layer (type)          Output Shape         Param #
=====
input_3 (InputLayer)  [(None, 200)]        0
embedding_2 (Embedding) (None, 200, 150)    3750000
conv1d_2 (Conv1D)      (None, 198, 64)      28864
global_max_pooling1d_2 (GlobalMaxPooling1D) (None, 64)    0
dense_4 (Dense)        (None, 32)           2080
dense_5 (Dense)        (None, 8)            264
=====
Total params: 3,781,208
Trainable params: 3,781,208
Non-trainable params: 0

```

La métrica utilizada para evaluar el desempeño es el accuracy y, como es un problema de clasificación multiclas, como función de pérdida categorical_crossentropy. Por su parte, se emplea Adam para la utilización del algoritmo de propagación del error hacia atrás (parámetros por defecto).

```
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Para el proceso de entrenamiento de la red destacar:

- Un máximo de 10 épocas y actualización de los pesos cada 128 muestras
- Reserva del 20% del dataset para ser usado como validación

- Uso de parada temprana para recoger el mejor modelo posible en el proceso iterativo

```

epochs = 10
batch_size = 128

history = modelo.fit(X_train, Y_train,
                      epochs=epochs,
                      batch_size=batch_size,
                      validation_split=0.2,
                      callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                      patience=2)])

```

```

Epoch 1/10
335/335 [=====] - 51s 151ms/step - loss: 1.6626 - accuracy: 0.5294 - val_loss: 0.7403 - val_accuracy: 0.8004
Epoch 2/10
335/335 [=====] - 51s 153ms/step - loss: 0.6644 - accuracy: 0.8204 - val_loss: 0.6176 - val_accuracy: 0.8281
Epoch 3/10
335/335 [=====] - 52s 155ms/step - loss: 0.5052 - accuracy: 0.8615 - val_loss: 0.5723 - val_accuracy: 0.8360
Epoch 4/10
335/335 [=====] - 51s 152ms/step - loss: 0.4040 - accuracy: 0.8928 - val_loss: 0.5763 - val_accuracy: 0.8376
Epoch 5/10
335/335 [=====] - 51s 153ms/step - loss: 0.3247 - accuracy: 0.9172 - val_loss: 0.5813 - val_accuracy: 0.8387

```

Una vez realizado el entrenamiento, se visualiza el proceso para conocer su convergencia

```

# construcción de un data.frame
df_train=pd.DataFrame(history.history)
df_train['epochs']=history.epoch

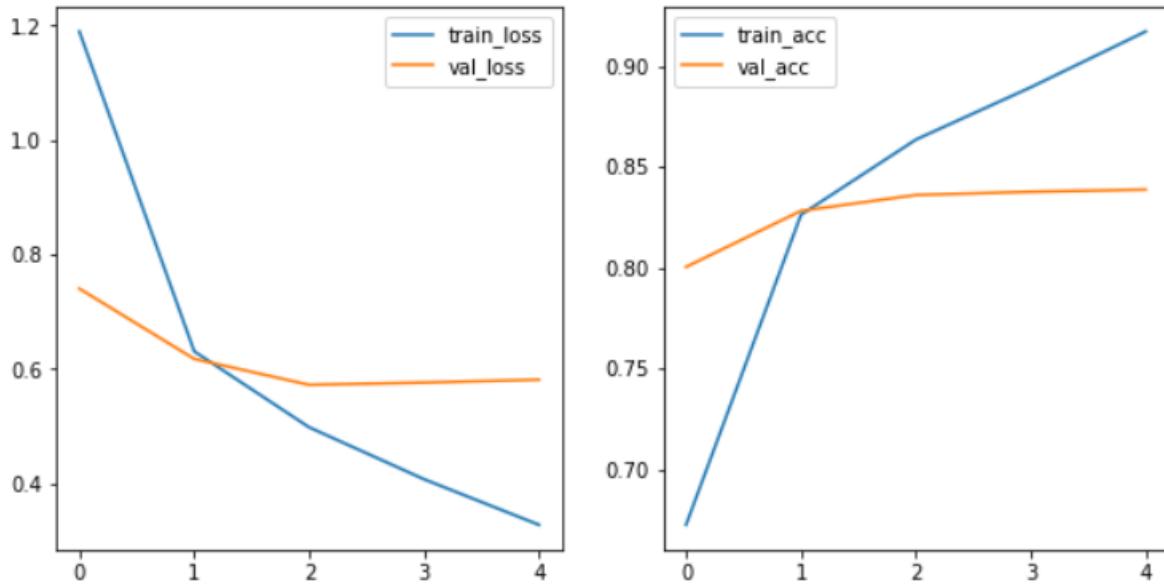
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

ax1.plot(df_train['epochs'], df_train['loss'], label='train_loss')
ax1.plot(df_train['epochs'], df_train['val_loss'], label='val_loss')

ax2.plot(df_train['epochs'], df_train['accuracy'], label='train_acc')
ax2.plot(df_train['epochs'], df_train['val_accuracy'], label='val_acc')

ax1.legend()
ax2.legend()
plt.show()

```



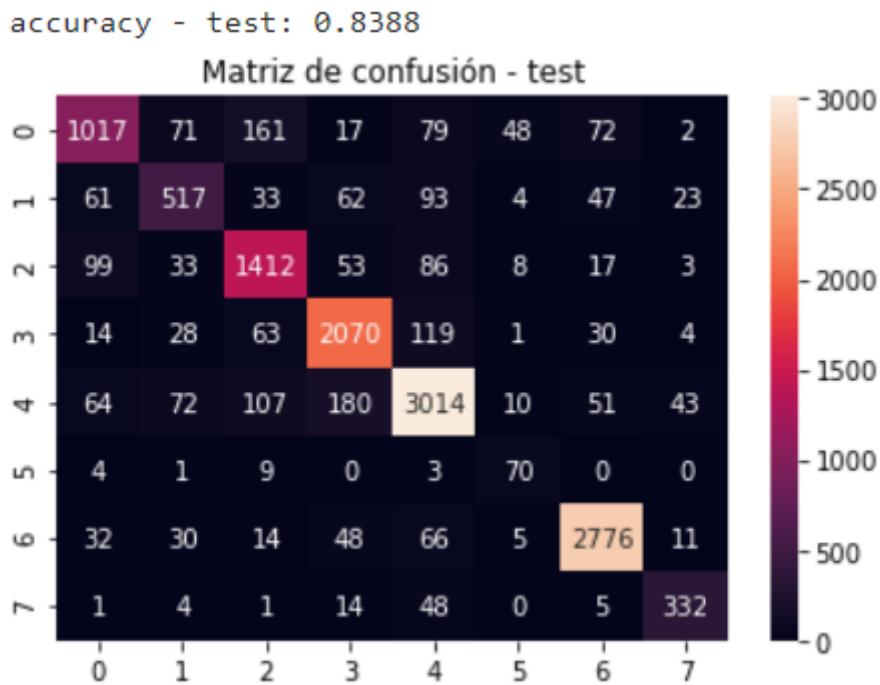
Finalmente, se estima la bondad de ajuste con la muestra de test. Para ello, como esta muestra hace referencia a la puesta en producción del modelo, es necesario crear las secuencias de este nuevo dataset en función de la tokenización del modelo.

```
X_test = tokenizer.texts_to_sequences(X_test_text)
X_test      = tf.keras.preprocessing.sequence.pad_sequences(X_test,
maxlen=MAX_SEQUENCE_LENGTH)
```

Y ahora ya sí, se realizan las predicciones y se evaluar el performance del modelo creado en la muestra de test.

```
# uso de argmax para pasar de probabilidad a estimación final
Y_test_pred = np.argmax(modelo.predict(X_test), axis=1) # predicción de la etiqueta
Y_test_label = np.argmax(Y_test, axis=1) # obtención de las etiquetas sin dummy
print('accuracy - test:', np.round(accuracy_score(Y_test_label,
Y_test_pred),5))

sns.heatmap(confusion_matrix(Y_test_pred, Y_test_label), annot = True,
fmt='%.0f') # matriz de confusión
plt.title('Matriz de confusión - test')
plt.show()
```



7.5. Redes Neuronales Recurrentes

7.5.1. Introducción

Las peculiaridades de esta red permiten incorporar a la red el concepto de temporalidad, y también que la red tenga memoria, porque la información que introducimos en un momento dado en las neuronas de entrada es transformada, y continúa circulando por la red.

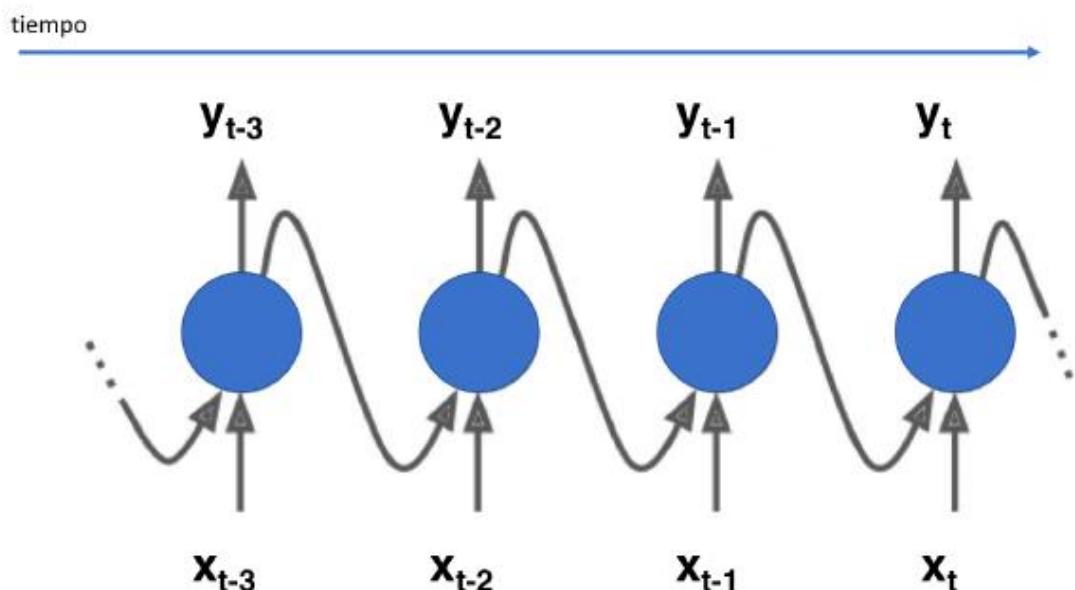
Las redes neuronales recurrentes modelan secuencias. Estas secuencias son únicas y diferenciables respecto a otro tipo de datos: el orden es importante y los elementos de estas secuencias no son independientes unos de otros, tal y como ocurría en los otros planteamientos de redes neuronales como el perceptrón multicapa o las redes convolucionales que no son capaces de recordar información pasada y por lo tanto procesar nuevos eventos.

Las redes de neuronas recurrentes (*Recurrent Neural Networks*) no disponen de una estructura de capas, sino que permiten conexiones arbitrarias entre todas las neuronas, incluso creando ciclos.

El permitir esta arquitectura de conexiones recurrentes, conlleva un aumento en el número de pasos o de parámetros ajustables de la red, lo que incrementa la capacidad de representación, pero también la complejidad del aprendizaje.

Una representación gráfica se muestra a continuación.

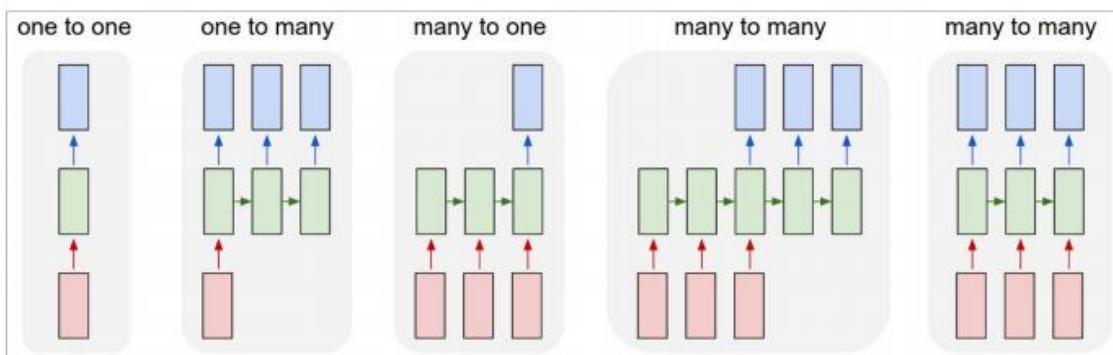
FIGURA N° 80. ESQUEMA DE RED NEURONAL RECURRENTE



La secuencia la representamos por X_{t-3} , X_{t-2} , X_{t-1} , X_t . En este caso el sub índice indica el orden de las instancias. El flujo de la información en paso temporales adyacentes en la capa oculta va a permitir a la red disponer de un recuerdo de lo ocurrido en su pasado.

En el artículo de Karpathy (2015) "The unreasonable effectiveness of recurrent neural networks" (La eficacia razonable de las redes neuronales recurrentes) se describen cuatro diferentes opciones que se presentan según la secuencia de los datos de entrada y/o de salida.

FIGURA N° 81. OPCIONES DE RED NEURONAL RECURRENTE SEGÚN ENTRADAS Y SALIDAS



Fuente: Raschka. S y Mirjalili (2019)

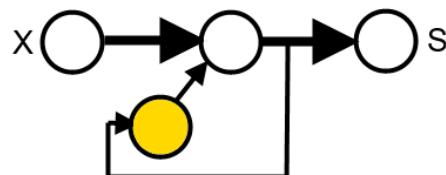
- ✓ **One to many:** En esta opción los datos entran en una forma estándar pero la salida es una secuencia. Ejemplos de este formato puede ser la catalogación de videos donde la entrada es una película y la salida una frase.
- ✓ **Many to one:** La entrada está en forma de secuencia y la salida tiene un tamaño fijo. En esta categoría entra el análisis de sentimientos
- ✓ **Many to many:** Las salidas y las entradas son secuencias que pueden estar sincronizadas (clasificación de videos) o retrasadas (traducción de un idioma a otro)

7.5.2. Redes clásicas: Elman y Jordan

Existen diferentes planteamientos de redes recurrentes. Las primeras redes recurrentes consideradas clásicas y que todavía gozan de cierta popularidad por sus aplicaciones son las redes de Elmann y de Jordan

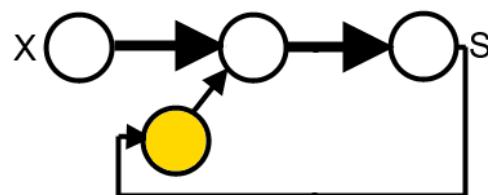
En las redes de Elman, las entradas de estas neuronas, se toman desde las salidas de las neuronas de una de las capas ocultas, y sus salidas se conectan de nuevo en las entradas de esta misma capa, lo que proporciona una especie de memoria sobre el estado anterior de dicha capa. El esquema es como en la Figura 78, donde X es la entrada, S la salida y el nodo amarillo es la neurona de la capa de contexto.

FIGURA N° 82: RED NEURONAL DAE ELMAN



En las redes de Jordan, la diferencia está en que la entrada de las neuronas de la capa de contexto se toma desde la salida de la red.

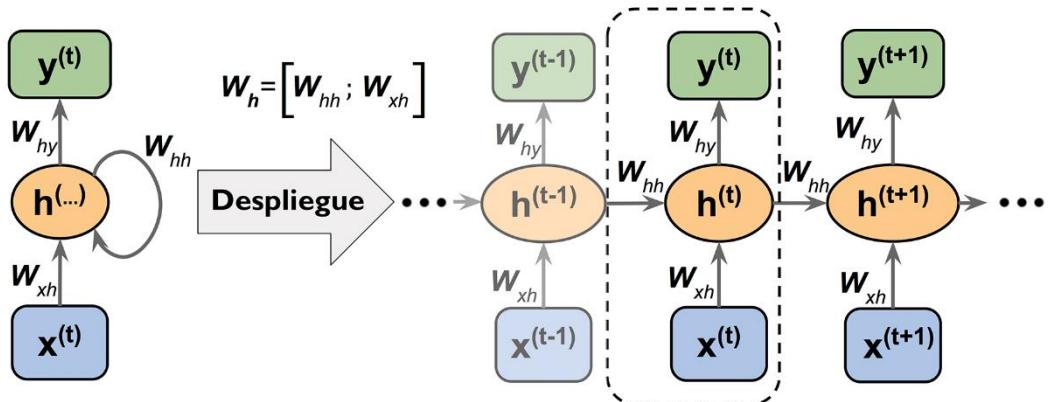
FIGURA N° 83: RED NEURONAL DE JORDAN



7.5.3. Red. LSTM (Long Short-Term Memory)

El despliegue de una red neuronal recurrente para tres momentos del tiempo puede observarse en el siguiente diagrama.

FIGURA N° 84. RED NEURONAL LONG SHORT-TERM MEMORY



Fuente: Raschka. S y Mirjalili (2019)

Vamos a calcular las activaciones de las capas ocultas y de la capa de salida en esta red neuronal recurrente. Recordar que los pesos se comparten a través del eje temporal. Las diferentes matrices que aparecen asociadas a esta red son:

- W_{xh} que representa la matriz ponderada entre la entrada $x^{(t)}$ y la capa oculta h .
- W_{hh} es la matriz que está asociada al borde recurrente.
- W_{hy} son los valores de la matriz ponderada entre la capa oculta y la capa de salida.

Para la capa oculta, la entrada z_h (preativación) se tiene que calcular a través de la suma de las multiplicaciones de las matrices ponderadas con los respectivos vectores y añadir el sesgo:

$$z_h^{(t)} = w_{xh}x^{(t)} + w_{hh}h^{t-1} + b_h \quad [178]$$

Las unidades ocultas con su función de activación se calculan por:

$$h^t = c(z_h^t) = \phi_h(w_{xh}x^{(t)} + w_{hh}h^{t-1} + b_h) \quad [179]$$

Finalmente, la activación de las unidades de salida se calcula de la siguiente forma:

$$y^{(t)} = \phi_y(w_{hy}h^{(t)} + b_y) \quad [180]$$

Un tema muy importante relacionado con el gradiente a la hora de entrenar este tipo de redes y, en general, a cualquier tipo de red, son los problemas que se conocen como Exploding Gradients,

si el algoritmo asigna una importancia exageradamente elevada a los pesos sin mucha razón y Vanishing Gradients que es el caso contrario, los valores del gradiente son extremadamente bajos.

Los gradientes en las redes recurrentes pueden no ser fáciles de calcular, pero al tener una referencia temporal la función de pérdida que tenemos que minimizar es la suma de todos los momentos desde $t = 1$ hasta $t = T$:

$$L = \sum_{i=1}^T L^{(t)} \quad [181]$$

Dado que esta función de perdida depende de las unidades ocultas de todas las etapas anteriores, el gradiente se tiene que calcular a través de la siguiente expresión:

$$\frac{\partial L^{(t)}}{\partial w_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^t \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial w_{hh}} \right) \quad [182]$$

Donde $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ es una multiplicación de los pasos temporales adyacentes y tiene la siguiente expresión:

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^t \frac{\partial h^{(i)}}{\partial h^{(i-1)}} \quad [183]$$

Este término tiene $t-k$ multiplicaciones lo que significa que si los pesos w son muy pequeños o muy grandes el resultado de la multiplicación se hará muy cercano a cero (Vanishing Gradients) o será excesivamente grande (Exploding Gradients)

Una de las formas de resolver estos problemas en las redes recurrentes es incorporando a la red el concepto de gate unit (puertas) que incorporan las redes LSTM.

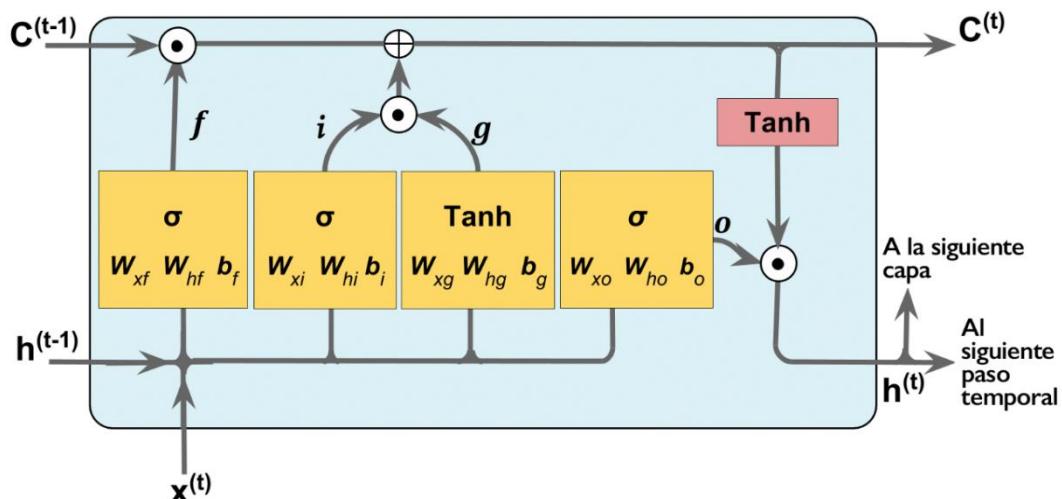
Estas redes neuronales se utilizan habitualmente en el tratamiento de textos de lenguajes naturales, en el reconocimiento de escritura a mano, en el reconocimiento de voz, de gestos, en la captura de imágenes, y en otros campos similares de conocimiento.

La LSTM (Long Short-Term Memory) se desvía de las arquitecturas de redes neuronales típicas basadas en neuronas e incluye el concepto de una celda de memoria. La celda de memoria puede retener su valor durante un periodo de tiempo corto o largo como una función de sus entradas, esto va permitir a la celda recordar lo que es importante y no solamente el último valor que calculó.

La celda de memoria LSTM contiene tres puertas que controlan la forma de como la información fluye dentro o fuera de la celda: puerta de entrada (input gate), puerta de olvidar (forget gate) y puerta de salida (out gate)

La puerta de entrada controla cuando la información nueva puede entrar en la memoria. La puerta del olvido controla cuando se olvida una parte de la información, lo que permite a la celda recordar datos nuevos. Por último, la puerta de salida controla cuando se utiliza en el resultado de la celda la información que está contenida en la celda. Las celdas pueden contener ponderaciones, que nos sirven para controlar a cada puerta. El algoritmo de capacitación, normalmente BPTT (Back Propagation Through Time), optimiza esas ponderaciones basándose en el error de salida de red resultante. El BPTT se basa en el gradiente de una función de costo que esté relacionada directamente con los pesos de toda la red.

FIGURA N° 85: ESTRUCTURA DE UNA CELDA DE LA RED LSTM



Fuente: Raschka. S y Mirjalili (2019)

El desarrollo en forma matemática de las puertas de la celda de LSTM se expresa así:

La puerta del olvido (f_t) que posibilita a la celda de memoria restablecer el estado de la celda sin crecer de forma indefinida y, además, decide aquella información que debe suprimirse o pasar. Esta puerta fue añadida años después del modelo original (Gers et al, 2000)

$$f_t = \sigma(w_{xf}x^{(t)} + w_{hf}h^{(t-1)} + b_f) \quad [184]$$

La puerta i_t y el nodo de entrada g_t se encargan de actualizar el estado de la celda

$$i_t = \sigma(\mathbf{w}_{xi}x^{(t)}\mathbf{w}_{hi}h^{(t-1)} + \mathbf{b}_i) \quad [185]$$

$$g_t = \tanh (\mathbf{w}_{xg}x^{(t)}\mathbf{w}_{hg}h^{(t-1)} + \mathbf{b}_g) \quad [186]$$

El estado de la celda en el momento t se calcula así.

$$C^t = (C^{t-1} \otimes f_i) \oplus (i_t \otimes g_t) \quad [187]$$

La puerta de salida (O_t) es la que decide cómo actualizar los valores de las unidades ocultas:

$$O_t = \sigma(W_{xo}x^{(t)}) + (W_{ho}h^{(t-1)} + b_o) \quad [188]$$

Las unidades ocultas en el paso temporal actual se calculan de esta forma:

$$h^t = O_t \otimes \tanh(C^t) \quad [189]$$

7. 5.4. Red GRU

El GRU (Gated Recurrent Unit) constituye un enfoque más reciente que las LSTM y fue propuesto en el año 2014 por Cho et al. Estas redes tienen una arquitectura más sencilla, lo que implica que es más eficiente computacionalmente y el rendimiento es comparable con las LSTM

Esta arquitectura incluye dos puertas: una puerta de actualización y una puerta de reajuste. La puerta de actualización indica cuánto del contenido de las celdas anteriores hay que mantener. La puerta de reajuste define cómo incorporar la nueva entrada con los contenidos anteriores de la celda. Un GRU puede modelar un RNN estándar simplemente estableciendo la puerta de reajuste a 1 y la puerta de actualización a 0.

Para ampliar información de la GRU se puede consultar el documento escrito por Junyoung Chung et al. (2014) titulado “Evaluación empírica de redes neuronales recurrentes cerradas en modelo secuencial”

Los autores de la investigación evaluaron empíricamente redes neuronales recurrentes (RNN) con tres modelos ampliamente utilizados de unidades recurrentes: una unidad de tanh tradicional, una unidad de memoria a largo plazo (LSTM) y una unidad recurrente cerrada (GRU). La evaluación se centró en la tarea de secuencia de modelado en una serie de conjuntos de datos, incluidos datos de música polifónicos y datos de señales de voz sin formato. La evaluación demostró claramente la superioridad de las unidades cerradas; tanto la unidad LSTM como GRU, sobre la

unidad tradicional de tanh. Sin embargo, afirman los autores que no pudieron llegar a una conclusión concreta sobre cuál de las dos unidades de compuerta fue mejor.

Las redes recurrentes son actualmente utilizadas para diferentes propósitos: análisis de sentimientos, generación y clasificación de texto (o audio), predicción de series temporales, etc. En este caso, veremos en detalle un ejemplo de clasificación de texto.

Se presenta a continuación una aplicación práctica de clasificación de texto multiclasa a partir de redes LSTM. Para ello, se utiliza una bbdd referida a las reclamaciones de los usuarios ante una entidad bancaria en función del tipo de producto.

Ejemplo de clasificación de textos con LSTM en R

Las redes recurrentes son actualmente utilizadas para diferentes propósitos: análisis de sentimientos, generación y clasificación de texto (o audio), predicción de series temporales, etc. En este caso, veremos en detalle un ejemplo de clasificación de texto.

Se presenta a continuación una aplicación práctica de clasificación de texto multiclasa a partir de redes LSTM. Para ello, se utiliza una base de datos referida a las reclamaciones de los usuarios ante una entidad bancaria en función del tipo de producto.

```
library(data.table)
library(tidymodels)
library(formattable)
library(skimr)
library(janitor)
library(yardstick)
library(tensorflow)
# install_tensorflow(version = "2.0.0")
library(keras)
library(fastDummies)
library(ramify)
library(gridExtra)

# semilla reproducción datos
# use_session_with_seed(123)
set.seed(123)
```

El fichero de trabajo contiene una serie de reclamaciones que no vienen acompañadas con su texto asociado. Se considera que lo más adecuado es excluir tales instancias del dataset de partida.

```
datos <- fread("D:/03. Deep Learning en accion - UNED/Ejercicios/3.
Recurrente/consumer_complaints.csv")

# selección de las variables de interés y eliminación de blanks
datos <- datos %>% select(product, consumer_complaint_narrative) %>%
```

```

filter(consumer_complaint_narrative != '')

paste('Tamaño de los datos (instancias, columnas):', dim(datos)[1],
dim(datos)[2])

```

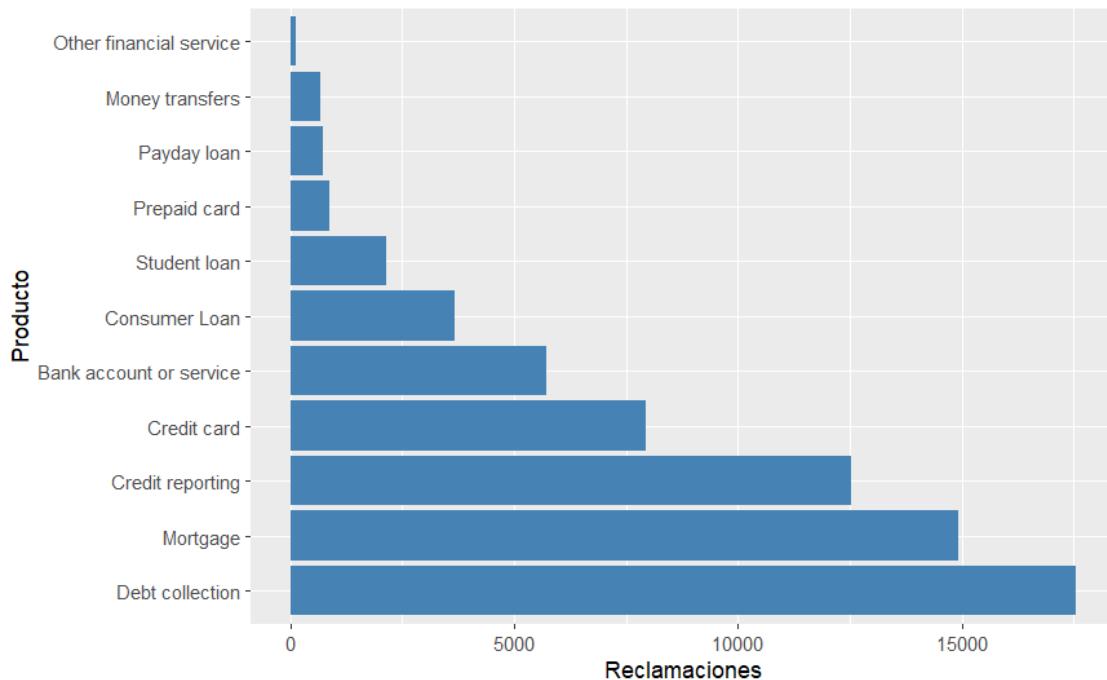
Se analiza el número de muestras de la variable target. Como puede verse, la muestra de trabajo está desbalanceada.

```

g <- ggplot(data=datos, aes(x=reorder(product, -table(product)[product]))) +
geom_bar(stat="count", fill="steelblue")

g + coord_flip() + labs(y= "Reclamaciones", x= "Producto")

```



Como puede verse, se parte de once tipos de productos diferentes; si bien, para varios de ellos el número de reclamaciones no es considerado significativo por el área legal de la entidad. Por ello, y en base a la similitud de los productos, se agrupan las cuatro categorías con un menor número reclamaciones en:

- Prepaid card: se incluye en la categoría de "Credit card"
- Payday loan: se incluye en la categoría "Bank account or service"
- Money transfers y Other financial service: forman un grupo conjunto denominado "Money transfers and Other financial service"

```

# agrupaciones
datos$product <- if_else(datos$product == "Payday loan", 'Bank account or'

```

```

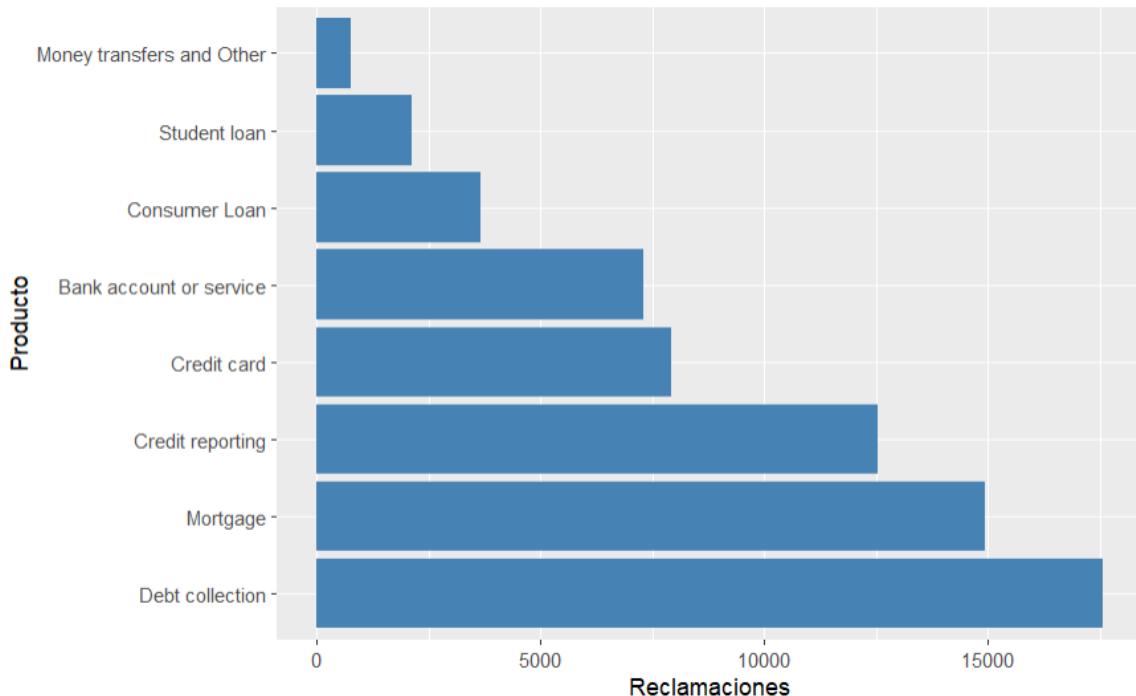
service', datos$product) # préstamos
datos$product <- if_else(datos$product == "Prepaid card", 'Bank account or
service', datos$product) # créditos

tipo_producto <- c('Money transfers', 'Other financial service')
datos$product <- if_else(datos$product %in% tipo_producto, 'Money transfers
and Other', datos$product) # créditos

g <- ggplot(data=datos, aes(x=reorder(product, -table(product)[product]))) +
geom_bar(stat="count", fill="steelblue")

g + coord_flip() + labs(y= "Reclamaciones", x= "Producto")

```



De esta forma, el número de grupos ha sido distribuido de una forma más equitativa. A modo de ejemplo, se muestra una de las reclamaciones:

```

plot_reclamaciones <- function(df, elemento){
  df <- as.list(df[elemento,])
  return(df)
}

plot_reclamaciones(datos, 101)[1]
plot_reclamaciones(datos, 101)[2]

```

```

$product
[1] "Consumer Loan"

$consumer_complaint_narrative
[1] "I contacted Drivetime on several different occasions regarding a car loan I had with them account # XXXX i do not have the number for the other loan that was removed also from the credit bureau with a positive credit rating that was closed and paid in full. The company insisted on avoiding an agreement entered into by myself and XXXX of their company representative in a corporate position they have the agreed upon resolution and can provide your agency with a copy of the same because i do not have the capability to do so with my computer. Drivetime during the month of XXXX XXXX removed my credit reporting information from XXXX credit bureaus reporting agencies I believe XXXX and XXXX XXXX I discovered in or about XXXX XXXX that my positive credit ratings were no longer in the XXXX and XXXX credit bureaus and that another positive account for another car loan was no longer reporting in my credit file. these removals adversely affected my credit ratings and as such my ability to purchase something with a lower interest rate. I have never missed a payment to the Drivetime company on my car loans inclusive of XXXX XXXX XXXX XXXX different auto loans and for them to destroy my credit ratings is despicable. I spoke with XXXX today ( XXXX/XXXX/XXXX regarding those issues and his response to me was that Drivetime is now in the process of updating all consumers credit files and all of the consumers credit files were removed from the credit bureaus and is being updated however he does not know how the process is being performed who's records are being updated/ corrected and what process or timeframes this process will take and furthermore does the process involves all consumers credit reports or are they updating reports for existing customers again which would not apply to me because my loans have been paid off and are closed however they did the same thing to my credit reports. Therefore the process would leave me with no credit ratings for accounts that I have paid off in good faith and never missed a payment in doing so however they kept trying to report my latest account as a repossession and or account was not satisfied I had them make the corrections and now back in XXXX or XXXX of XXXX I discovered yet again they tampered with my accounts by removing my positive information from the bureau for the account paid off in XXXX and for the account prior to that XXXX removed it from the credit bureau entirely and left the XXXX account in XXXX bureau I believe in the XXXX credit Bureau which was the same method of operations that has done to the existing consumers. Therefore Drivetime Representative XXXX gave me erroneous information about my accounts and for those reasons I am submitting a complaint to the Consumer Financial Protection Bureau based on the ruling implemented in terms of enforcement against the Drivetime XXXX XXXX for their violations of consumer rights dated XXXX XXXX, XXXX which is what they did to me in XXXX XXXX XXXX and prior to those dates in XXXX. \n"

```

Como puede verse, la reclamación 101 del dataset está asociada a un préstamo al consumo. Sin embargo, lo verdaderamente interesante del texto de ejemplo es la necesidad de realizar un preprocessado a los textos puesto que algunos símbolos, caracteres o, incluso palabras, no son relevantes para que la red sea capaz de interpretar el contenido del mismo. Por tanto, se lleva a cabo lo siguiente:

- Conversión del texto a minúsculas.
- Exclusión del texto el contenido cifrado (XXXX)
- Eliminación de caracteres extraños.

Para poder hacer este preprocessado de textos se hace uso del paquete base de R.

```

preprocesado <- function(reclamacion) {

  reclamacion <- tolower(reclamacion)
  reclamacion <- gsub("x", "", reclamacion)
  reclamacion <- gsub("[/(){}|#+_@,.;]", "", reclamacion)
}

datos$consumer_complaint_narrative <-
  sapply(datos$consumer_complaint_narrative, preprocesado)

```

Se presenta de nuevo el ejemplo anterior para ver el resultado del procesamiento de textos realizado.

```

plot_reclamaciones(datos, 101)[1]
plot_reclamaciones(datos, 101)[2]

```

```

$product
[1] "Consumer Loan"

$consumer_complaint_narrative
[1] "i contacted drivetime on several different occasions regarding a car loan i had with them account i do not have the number for the other loan that was removed also from the credit bureau with a positive credit rating that was closed and paid in full the company insisted on avoiding an agreement entered into by myself and of their company representative in a corporate position they have the agreed upon resolution and can provide your agency with a copy of the same because i do not have the capability to do so with my computer drivetime during the month of removed my credit reporting information from credit bureaus reporting agencies i believe and i discovered in or about that my positive credit ratings were no longer in the and credit bureaus and that another positive account for another car loan was no longer reporting in my credit file these removals adversely affected my credit ratings and as such my ability to purchase something with a lower interest rate i have never missed a payment to the drivetime company on my car loans inclusive of different auto loans and for them to destroy my credit ratings is despicable i spoke with today regarding those issues and his response to me was that drivetime is now in the process of updating all consumers credit files and all of the consumers credit files were removed from the credit bureaus and is being updated however he does not know how the process is being performed who 's records are being updated corrected and what process or timeframes this process will take and furthermore does the process involves all consumers credit reports or are they updating reports for existing customers again which would not apply to me because my loans have been paid off and are closed however they did the same thing to my credit reports therefore the process would leave me with no credit ratings for accounts that i have paid off in good faith and never missed a payment in doing so however they kept trying to report my latest account as a repossession and or account was not satisfied i had them make the corrections and now back in or of i discovered yet again they tampered with my accounts by removing my positive information from the bureau for the account paid off in and for the account prior to that removed it from the credit bureau entirely and left the account in bureau i believe in the credit bureau which was the same method of operations that has done to the existing consumers therefore drivetime representative gave me erroneous information about my accounts and for those reasons i am submitting a complaint to the consumer financial protection bureau based on the ruling implemented in terms of enforcement against the drivetime for their violations of consumer rights dated which is what they did to me in and prior to those dates in \n"

```

Partición de la muestra: entrentamineto (80%) y test (20%) y normalización de las variables.

```

split <- initial_split(data=datos,
                        prop=0.8,
                        strata=product)

# datos entrenamiento y test
train_data <- training(split)
test_data <- testing(split)

# Target - conversión a variables dummy
y_train <- dummy_cols(train_data, select_columns = 'product',
remove_selected_columns = TRUE)
y_train <- y_train %>% select(-consumer_complaint_narrative)

y_test <- dummy_cols(test_data, select_columns = 'product',
remove_selected_columns = TRUE)
y_test <- y_test %>% select(-consumer_complaint_narrative)

# datos de entrada
x_train_text <- train_data %>% select(-product)
x_test_text <- test_data %>% select(-product)

#
paste('Entrenamiento (instancias, columnas):', dim(x_train_text)[1])
paste('Test (instancias, columnas):', dim(x_test_text)[1])

```

Antes de definir la arquitectura de la red, se lleva a cabo la conversión del texto a variables numéricas que es el input que puede leer la red. Para ello, se realiza:

- La vectorización del texto asociado a las reclamaciones.
- El truncar y rellenado de las secuencias de entrada para igualar la longitud en la modelización.

```

MAX_NB_WORDS = 25000 # frecuencia de palabras
MAX_SEQUENCE_LENGTH = 200 # número de palabras en cada reclamacion
EMBEDDING_DIM = 150 # dimensión del embedding

tokenizer = text_tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;=>?@[\\"`{|}~\t\n', lower=TRUE)
tokenizer$fit_on_texts(x_train_text$consumer_complaint_narrative)
word_index <- tokenizer$word_index
paste('Tokens:', length(word_index))

X_train
tokenizer$texts_to_sequences(x_train_text$consumer_complaint_narrative) <-
X_train = pad_sequences(X_train, maxlen=MAX_SEQUENCE_LENGTH)

paste('Datos de entrada:', dim(X_train)[1], dim(X_train)[2])

```

Por último, se crea la red neuronal siguiendo el método funcional. La red tiene las siguientes capas:

- Entrada: de 200 neuronas pues corresponde con la longitud de las secuencias
- Embedding: de dimensión 200 y toma como input el número máximo de palabras (25.000)
- LSTM: de 128 neuronas y dropout
- Densa: de 32 neuronas y con función de activación "relu"
- Salida: capa densa con 8 neuronas (número de categorías del target) y función de activación "softmax"

```

input <- layer_input(shape=dim(X_train)[2])

embedding <- input %>% layer_embedding(input_dim=MAX_NB_WORDS,
output_dim=EMBEDDING_DIM)
lstm <- embedding %>% layer_lstm(units = 128,
                                      dropout = 0.2,
                                      recurrent_dropout=0.2)
capa_densa <- lstm %>% layer_dense(units = 32,
                                         activation = 'relu',
                                         kernel_regularizer =
regularizer_l2(0.01))

out <- capa_densa %>% layer_dense(units = dim(y_train)[2],
                                         activation = 'softmax')

modelo <- keras_model(inputs = input, outputs = out)

```

El summary nos muestra el número de parámetros por capa y el número de parámetros total. Puede verse que el alto número de parámetros viene, principalmente, por la capa de Embedding.

```
summary(modelo)
```

```
Model: "model"
-----  
Layer (type)          Output Shape       Param #  
=====-----  
input_1 (InputLayer)   [(None, 200)]      0  
embedding (Embedding) (None, 200, 150)    3750000  
lstm (LSTM)           (None, 128)        142848  
dense (Dense)         (None, 32)         4128  
dense_1 (Dense)       (None, 8)          264  
=====-----  
Total params: 3,897,240  
Trainable params: 3,897,240  
Non-trainable params: 0
```

La métrica utilizada para evaluar el desempeño es el accuracy y, como es un problema de clasificación multiclas, como función de pérdida categorical_crossentropy. Por su parte, se emplea Adam para la utilización del algoritmo de propagación del error hacia atrás (parámetros por defecto).

```
modelo %>% compile(optimizer='adam', loss='categorical_crossentropy', metrics = 'accuracy')
```

Para el proceso de entrenamiento de la red destacar:

- Un máximo de 10 épocas y actualización de los pesos cada 128 muestras.
- Reserva del 20% del dataset para ser usado como validación.
- Uso de parada temprana para recoger el mejor modelo posible en el proceso iterativo.

```
epochs <- 10
batch_size <- 128

Y_train <- as.matrix(y_train)

hist <- modelo %>% fit(X_train, Y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_split=0.2,
                        callbacks = c(callback_early_stopping(monitor =
"val_loss", patience = 2))
                      )
```

```

Epoch 1/10
335/335 [=====] - 595s 2s/step - loss: 0.5605 - accuracy: 0.8373 -
val_loss: 1.2577 - val_accuracy: 0.6161
Epoch 2/10
335/335 [=====] - 631s 2s/step - loss: 0.4801 - accuracy: 0.8657 -
val_loss: 1.2658 - val_accuracy: 0.6816
Epoch 3/10
335/335 [=====] - 614s 2s/step - loss: 0.4173 - accuracy: 0.8872 -
val_loss: 1.1302 - val_accuracy: 0.7093
Epoch 4/10
335/335 [=====] - 610s 2s/step - loss: 0.3707 - accuracy: 0.9015 -
val_loss: 1.3794 - val_accuracy: 0.6851
Epoch 5/10
335/335 [=====] - 628s 2s/step - loss: 0.3202 - accuracy: 0.9160 -
val_loss: 1.0912 - val_accuracy: 0.7460
Epoch 6/10
335/335 [=====] - 588s 2s/step - loss: 0.2892 - accuracy: 0.9262 -
val_loss: 1.3514 - val_accuracy: 0.6778
Epoch 7/10
335/335 [=====] - 530s 2s/step - loss: 0.2535 - accuracy: 0.9382 -
val_loss: 1.2070 - val_accuracy: 0.7531
Epoch 8/10
335/335 [=====] - 549s 2s/step - loss: 0.2259 - accuracy: 0.9469 -
val_loss: 1.4539 - val_accuracy: 0.6905
Epoch 9/10
335/335 [=====] - 559s 2s/step - loss: 0.2184 - accuracy: 0.9490 -
val_loss: 1.4880 - val_accuracy: 0.6949
Epoch 10/10
335/335 [=====] - 545s 2s/step - loss: 0.1845 - accuracy: 0.9593 -
val_loss: 1.6113 - val_accuracy: 0.6829

```

Una vez realizado el entrenamiento, se visualiza el proceso para conocer su convergencia

```

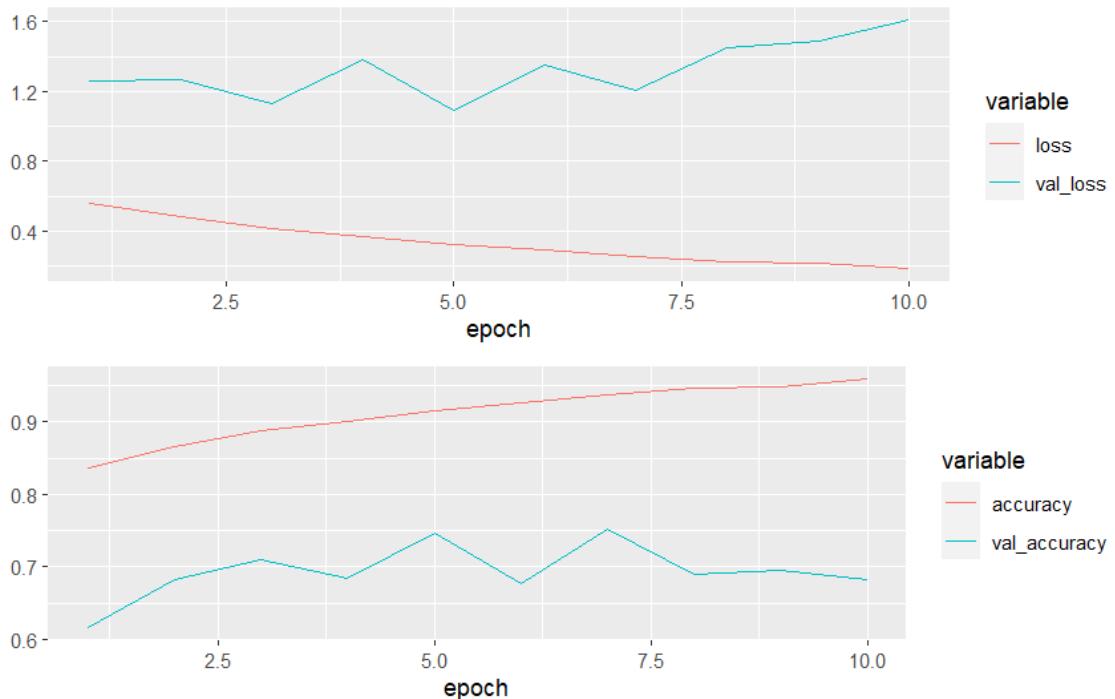
df_train <- as.data.frame(hist$metrics)
df_train$epoch <- c(1:nrow(df_train))

g_train <- df_train %>% gather(variable,value, loss, val_loss) %>%
  ggplot(aes(x=epoch, y=value, colour=variable)) + geom_line() + labs(y='')

g_val <- df_train %>% gather(variable,value, accuracy, val_accuracy) %>%
  ggplot(aes(x=epoch, y=value, colour=variable)) + geom_line() + labs(y='')

grid.arrange(g_train, g_val, ncol=1)

```



Finalmente, se estima la bondad de ajuste con la muestra de test. Para ello, como esta muestra hace referencia a la puesta en producción del modelo, es necesario crear las secuencias de este nuevo dataset en función de la tokenización del modelo.

```
X_test
tokenizer$texts_to_sequences(x_test_text$consumer_complaint_narrative)
X_test = pad_sequences(X_test, maxlen=MAX_SEQUENCE_LENGTH)

Y_test <- as.matrix(y_test)
```

Y ahora ya sí, se realizan las predicciones y se evaluar el performance del modelo creado en la muestra de test.

```
# uso de argmax para pasar de probabilidad a estimación final
Y_test_pred = argmax(predict(modelo, X_test)) # predicción de la etiqueta
Y_test_label = argmax(Y_test) # obtención de las etiquetas sin dummy

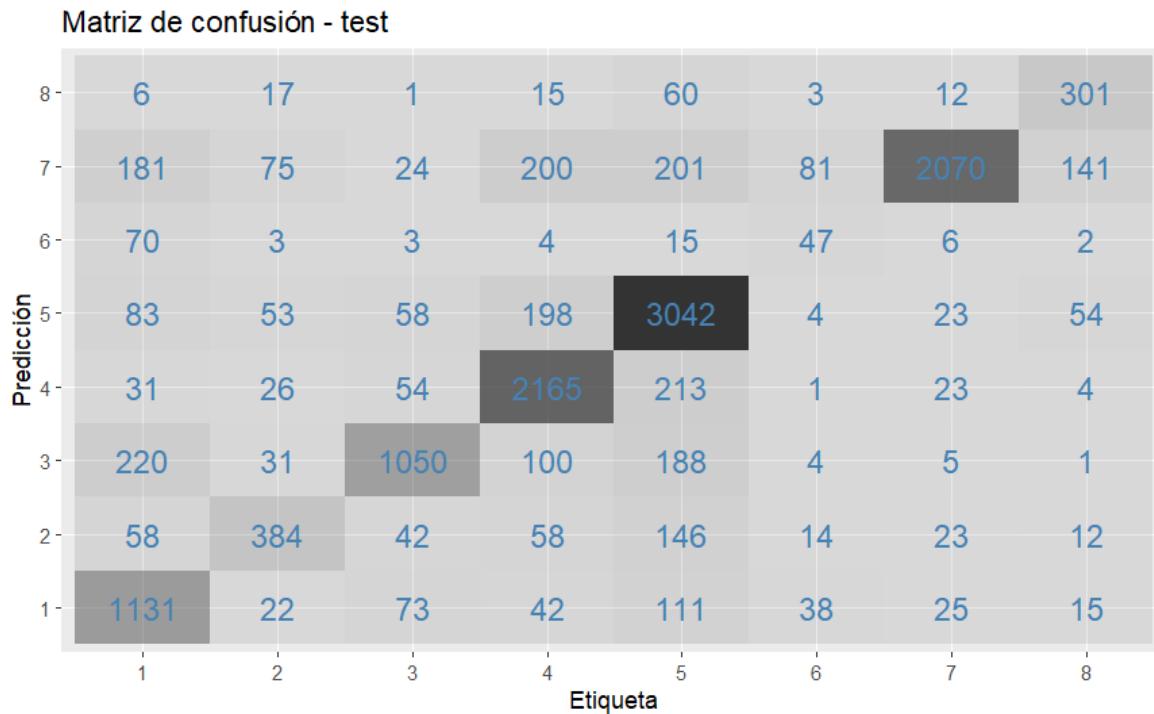
# construcción de data.frame (etiqueta, clase predicha y probabilidad)
pred_test <- as.data.frame(cbind(Y_test_label, Y_test_pred))
colnames(pred_test) <- c('label', 'pred')
pred_test$label <- as.factor(pred_test$label)
pred_test$pred <- as.factor(pred_test$pred)

# métricas
acc <- accuracy(pred_test, estimate=pred, truth=label) %>%
  mutate(accuracy=.estimate) %>% select(-.estimate) # accuracy
paste('accuracy', round(acc[,3],4))

# matriz confusión
mc <- conf_mat(pred_test, truth=label, estimate=pred)
```

```
# visualización
mc %>% pluck(1) %>% as_tibble() %>%
  ggplot(aes(Prediction, Truth, alpha = n)) + geom_tile(show.legend = FALSE) + geom_text(aes(label = n), colour = "steelblue", alpha = 1.5, size = 5) + labs(x= "Etiqueta", y= "Predicción", title="Matriz de confusión - test")
```

[1] "accucay 0.7626"



7.6. Autoencoders

Bases del Autoencoder

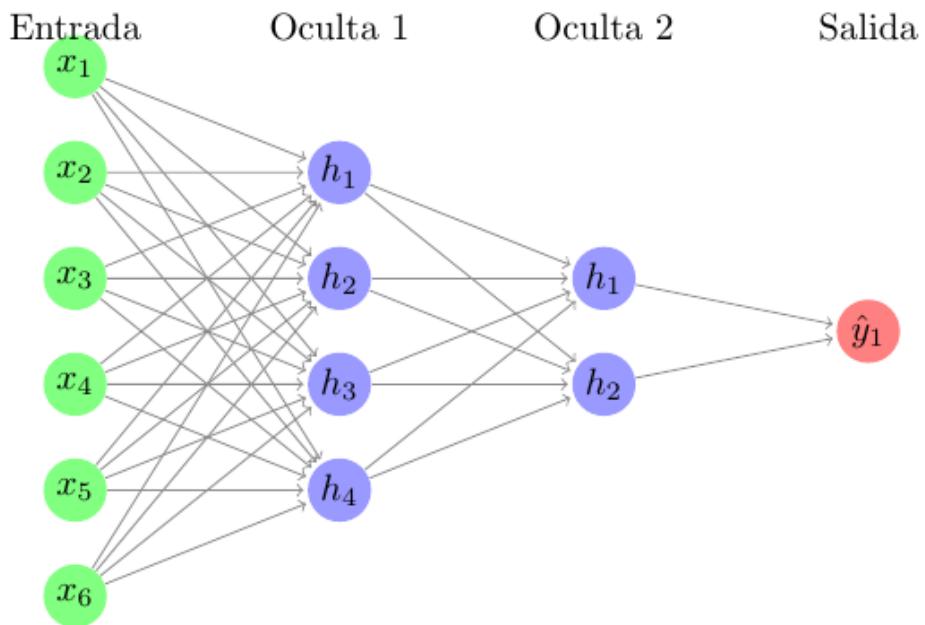
Los **Autoencoders (AE)** son uno de los tipos de redes neuronales que caen dentro del ámbito del **Deep Learning**, en la que nos encontramos con un modelo de **aprendizaje no supervisado**. Ya se empezó a hablar de AE en la década de los 80 (Bourlard and Kamp [1988](#)), aunque es en estos últimos años donde más se está trabajando con ellos.

La arquitectura de un AE es una Red Neuronal Artificial (ANN por sus siglas en inglés) que se encuentra dividida en dos partes, **encoder** y **decoder** (Charte et al. [2018](#)), (Goodfellow, Bengio, and Courville [2016](#)). El **encoder** va a ser la parte de la ANN que va codificar o comprimir los datos de entrada, y el **decoder** será el encargado de regenerar de nuevo los datos en la salida. Esta estructura de codificación y decodificación le llevará a tener una estructura **simétrica**. El AE es entrenado para ser capaz de **reconstruir** los datos de entrada en la capa de salida de la ANN,

implementando una serie de restricciones (la reducción de elementos en las capas ocultas del encoder) que van a evitar que simplemente se copie la entrada en la salida.

Si recordamos la estructura de una **ANN** clásica o también llamada **Red Neuronal Densamente Conectada** (ya que cada neurona conecta con todas las de la siguiente capa) nos encontramos en que en esta arquitectura, generalmente, el número de neuronas por capa se va reduciendo hasta llegar a la capa de salida que debería ser normalmente un número (si estamos en un problema regresión), un dato binario (si es un problema de clasificación).

FIGURA Nº 86: RED NEURONAL CLASICA

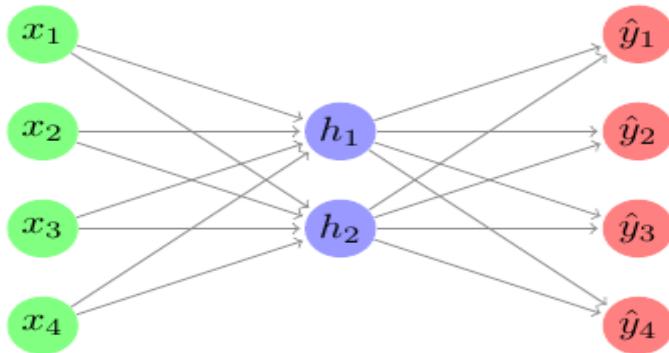


Fuente: Elaboración propia

Si pensamos en una estructura básica de AE en la que tenemos una capa de entrada, una capa oculta y una capa de salida, ésta sería su representación:

FIGURA N° 87: AUTOENCODER BÁSICO

Entrada Oculta Salida



Fuente: Elaboración propia

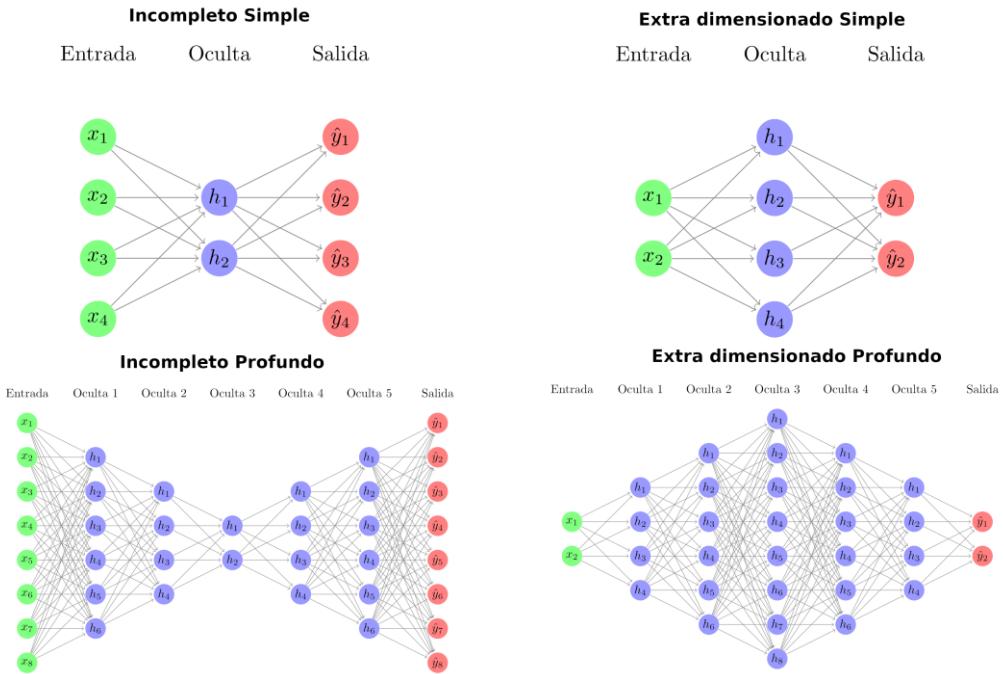
Donde los valores de x_1, x_2, x_3, x_4 son los datos de entrada y los datos $\hat{y}_1, \hat{y}_3, \hat{y}_4, \hat{y}_4$ son la reconstrucción de los mismos después de pasar por la capa oculta que tiene sólo dos dimensiones.

El objetivo del entrenamiento de un AE será que estos valores de \hat{y}_i sean lo más **parecidos** posibles a los x_i .

Según (Charte et al. 2018) los AE se pueden clasificar según el **tipo de arquitectura de red** en:

- Incompleto simple
- Incompleto profundo
- Extra dimensionado simple
- Extra dimensionado profundo

FIGURA Nº 88: TIPOS DE AUTOENCODERS POR ARQUITECTURA



Fuente: Elaboración propia

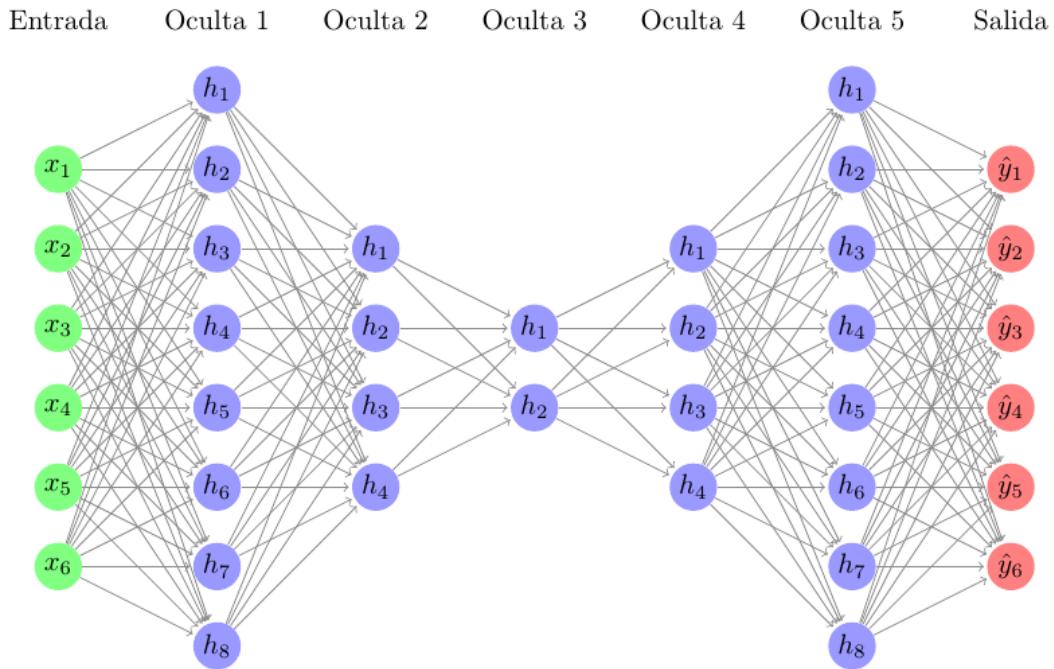
Cuando hablamos de **Incompleto** nos referimos a que tenemos una reducción de dimensiones que permite llegar a conseguir una “compresión” de los datos iniciales como técnica para que aprenda los patrones internos. En el caso de **Extra dimensionado** es cuando subimos de dimensión para conseguir que aprenda esos patrones. En este último caso sería necesario aplicar técnicas de regularización para evitar que haya un sobreajuste en el aprendizaje.

Cuando hablamos de **Simple** estamos haciendo referencia a que hay una única capa oculta, y en el caso de **Profundo** es que contamos con más de una capa oculta.

Normalmente se trabaja con las arquitecturas de tipo Incompleto profundo, sobre todo cuando se está trabajando con tipos de datos que son imágenes. Aunque también podríamos encontrar una **combinación de Incompleto con Extra dimensionado profundo** cuando trabajamos con tipos de datos que no son imágenes y así crecer en la primera o segunda capa oculta, para luego reducir. Esto nos permitiría por ejemplo adaptarnos a estructuras de AE en las que trabajemos con número de neuronas en una capa que sean potencia de 2, y poder construir arquitecturas dinámicas en función del tamaño de los datos, adaptándolos a un tamaño prefijado.

A continuación, vemos un gráfico de una estructura mixta **Extra dimensionado - Incompleto profundo**.

FIGURA Nº 89: AUTOENCODER MIXTO (INCOMPLETO Y EXTRA DIMENSIONADO)



Fuente: Elaboración propia

Idea intuitiva del uso de Autoencoders

Si un AE trata de reproducir los datos de entrada mediante un **encoder** y **decoder**, ¿que nos puede aportar si ya tenemos los datos de entrada?

Ya hemos comentado que la red neuronal de un AE es simétrica y está formada por un encoder y un decoder, además cuando trabajamos con los AE que son incompletos, se está produciendo una reducción del tamaño de los datos en la fase de codificación y de nuevo una regeneración a partir de esos datos más pequeños al original. Ya tenemos uno de los conceptos más importantes de los AE que es la **reducción de dimensiones** de los datos de entrada. Estas nuevas variables que se generan una vez pasado el **encoder** se les suele llamar el **espacio latente**.

Este concepto de reducción de dimensiones en el mundo de la minería de datos lo podemos asimilar rápidamente a técnicas como el **Análisis de Componentes Principales (PCA)**, que nos permite trabajar con un número más reducido de dimensiones que las originales. Igualmente, esa reducción de los datos y la capacidad de poder reconstruir el original podemos asociarlo al concepto de **compresión de datos**, de forma que con el encoder podemos comprimir los datos y con el decoder los podemos descomprimir. En este caso habría que tener en cuenta que sería una técnica de compresión de datos con pérdida de información (JPG también es un formato de

compresión con pérdida de compresión). Es decir, con los datos codificados y el AE (pesos de la red neuronal), seríamos capaces de volver a regenerar los datos originales.

Otra de las ideas alrededor de los AE es que, si nosotros tenemos un conjunto de **datos** de la **misma naturaleza** y los entrenamos con nuestro AE, somos capaces de construir una red neuronal (pesos en la red neuronal) que es capaz de reproducir esos datos a través del AE. Que ocurre si nosotros metemos un dato que no era de la misma naturaleza que los que entrenaron el AE, lo que tendremos entonces es que al recrear los datos originales no va a ser posible que se parezca a los datos de entrada. De forma que el **error** que vamos a tener va a ser mucho mayor por no ser datos de la misma naturaleza. Esto nos puede llevar a construir un AE que permita **detectar anomalías**, es decir, que seamos capaces de detectar cuando un dato es una anomalía porque realmente el AE no consigue tener un error lo bastante pequeño.

Según lo visto de forma intuitiva vamos a tener el **encoder** $f(X)$ que será el encargado de codificar los datos de entrada y luego tendremos el **decoder** $g(H)$ que será el encargado de realizar la decodificación y conseguir acercarnos al dato original X .

Es decir intentamos conseguir $g(f(X)) = \hat{Y} \approx X$.

Si suponemos un **Simple Autoencoder** en el que tenemos una única capa oculta, con una función de activación intermedia $\delta^{(1)}$ y una función de activación de salida $\delta^{(2)}$ y los parámetros $W^{(i)}$ y $b^{(i)}$ representan los parámetros de la red neuronal en cada capa, tendríamos la siguiente expresión:

$$f(X) = \delta^{(1)}(W^{(1)} * X + b^{(1)}) \quad [190]$$

$$g(H) = \delta^{(2)}(W^{(2)} * H + b^{(2)}) \quad [191]$$

$$\text{donde } f(g(H)) = \delta^{(2)}\left(W^{(2)} * \left(\delta^{(1)}(W^{(1)} * H + b^{(1)}) + b^{(2)}\right)\right) = \hat{Y} \approx X$$

Así tendremos que $g(f(X)) = \hat{Y}$ donde \hat{Y} será la reconstrucción de X

Una vez tenemos la idea intuitiva de para qué nos puede ayudar un AE, recopilamos algunos de los **principales usos** sobre los que actualmente se está trabajando. Más adelante, comentaremos algunos de ellos con más detalle.

Principales Usos de los Autoencoders

A continuación, veamos la explicación de cuales son algunos de los principales usos de los **autoencoders**:

Reducción de dimensiones / Compresión de datos

En la idea intuitiva de los AE ya hemos visto claro que se pueden usar para la reducción de dimensiones de los datos de entrada. Si estamos ante unos datos de entrada de tipo estructurado estamos en un caso de reducción de dimensiones clásico, en el que queremos disminuir el número de variables con las que trabajar.

Muchas veces este tipo de trabajo se hace mediante el **PCA (Análisis de Componente Principales, por sus siglas en inglés)**, sabiendo que lo que se realiza es una transformación lineal de los datos, ya que conseguimos unas nuevas variables que son una combinación lineal de las mismas. En el caso de los AE conseguimos mediante las funciones de activación **no lineales** (sigmoid, ReLU, tanh, etc) combinaciones no lineales de las variables originales para reducir las dimensiones. También existen versiones de PCA no lineales llamadas **Kernel PCA** que mediante las técnicas de **kernel** son capaces de construir relaciones no lineales.

En esta línea estamos viendo que cuando el **encoder** ha actuado, tenemos unos nuevos datos **más reducidos** y que somos capaces de **practicamente** volver a reproducir teniendo el **decoder**. Podríamos pensar en este tipo de técnica para simplemente comprimir información. Hay que tener en cuenta que este tipo de técnicas no se pueden aplicar a cualquier dato que queramos comprimir, ya que debemos haber entrenado al AE con unos datos de entrenamiento que ha sido capaz de obtener ciertos patrones de ellos, y por eso es capaz luego de reproducirlos.

Búsqueda de imágenes

Cuando pensamos en un buscador de imágenes nos podemos hacer a la idea que el buscar al igual que con el texto nos va a mostrar entradas que seán imágenes parecidas a la que estamos buscando.

Si construimos un autoencoder, el **encoder** nos va a dar unas variables con información para poder recrear de nuevo la imagen. Lo que parece claro es que si hay muy poca **distancia** entre estas variables y otras la reconstrucción de la imagen será muy parecida.

Así nosotros podemos entrenar el AE con nuestro conjunto de imágenes, una vez tenemos el AE pasamos el **encoder** a todas las imágenes y las tenemos todas en ese nuevo espacio de variables.

Cuando queremos buscar una imagen, le pasamos el autoencoder, y ya buscamos las más **cercanas** a nuestra imagen en el espacio de variables generado por el encoder.

Detección de Anomalías

Cuando estamos ante un problema de clasificación y tenemos un conjunto de datos que está muy desbalanceado, es decir, tenemos una clase mayoritaria que es mucho más grande que la minoritaria (posiblemente del orden de más del 95%), muchas veces es complicado conseguir un conjunto de datos balanceado que sea realmente bueno para hacer las predicciones.

Cuando estamos en estos entornos tan desbalanceados muchas veces se dice que estamos ante un sistema para detectar **anomalías**. Un AE nos puede ayudar a detectar estas anomalías de la siguiente forma:

- Tomamos todos los datos de entrenamiento de la clase mayoritaria (o normales) y construimos un AE para ser capaces de reproducirlos. Al ser todos estos datos de la misma naturaleza conseguiremos entrenar el AE con un **error** muy pequeño.
- Ahora tomamos los datos de la clase minoritaria (o anomalías) y los pasamos a través del AE obteniendo unos **errores de reconstrucción**.
- Definimos el **umbral** de error que nos separará los datos **normales** de las **anomalías**, ya que el AE sólo está entrenado con los normales y conseguirá un error más alto con las anomalías al reconstruirlas.
- Cogemos los datos de test y los vamos pasando por el AE, si el error es menor del umbral, entonces será de la clase mayoritaria. Si el error es mayor que el umbral, entonces estaremos ante una **anomalía**.

Eliminación de ruido

Otra de las formas de uso de los autoencoders en tratamiento de imágenes es para eliminar **ruido** de las mismas, es decir poder quitar **manchas** de las imágenes. La forma de hacer esto es la siguiente:

- Partimos de un conjunto de datos de entrenamiento (imágenes) a las que le metemos **ruido**, por ejemplo, modificando los valores de cada pixel usando una distribución normal, de forma que obtenemos unos datos de entrenamiento con ruido.

- Construimos el AE de forma que los datos de entrada son los que tienen **ruido**, pero los de salida vamos a forzar que sean los **originales**. De forma que intentamos que aprendan a reconstruirse como los que no tienen ruido.
- Una vez que tenemos el AE y le pasamos datos de test con ruido, seremos capaces de reconstruirlos sin el ruido.

Modelos generativos

Cuando hablamos de modelos generativos, nos referimos a AE que son capaces de generar **cosas nuevas** a las que existían. De forma que mediante técnicas como los Variational Autoencoders, los Adversarial Autoencoders seremos capaces de generar nuevas imágenes que no teníamos inicialmente. Es decir, podríamos pensar en poder tener un AE que sea capaz de reconstruir imágenes de caras, pero que además con toda la información aprendida fuera capaz de generar nuevas caras que realmente no existen.

Diseño del modelo de AE

Transformación de datos

Cuando se trabaja con redes neuronales y en particular con AEs, necesitamos representar los valores de las **variables de entrada** en forma **numérica**. En una red neuronal todos los datos son siempre numéricos. Esto significa que todas aquellas variables que sean categóricas necesitamos convertirlas en numéricas. Además es muy conveniente normalizar los datos para poder trabajar con valores entre 0 y 1, que van a ayudar a que sea más fácil que se pueda converger a la solución.

Como ya sabemos normalmente nos encontramos que en una red neuronal las **variables de salida** son:

- un número (**regresión**)
- una serie de números (**regresión múltiple**)
- un dato binario (**clasificación binaria**)
- un número que representa una categoría (**clasificación múltiple**)

En el caso de los AE puede que tengamos una gran parte de las veces valores de series de números, ya que necesitamos volver a representar los datos de entrada. Esto significa que tendremos que conseguir en la capa de salida esos datos numéricos que teníamos inicialmente, como si se tuviera una regresión múltiple.

Arquitectura de red

Como ya se ha comentado en las redes neuronales, algunos de los **hiperparámetros** más importantes en un AE son los relacionados con la arquitectura de la red neuronal.

Para la construcción de un AE vamos a elegir una topología simétrica del **encoder** y el **decoder**.

Durante el diseño del AE necesitaremos ir probando y adaptando todos estos hiperparámetros de la ANN para conseguir que sea lo más eficiente posible:

- Número de capas ocultas y neuronas en cada una
- Función de coste y pérdida
- Optimizador
- Función de activación en capas ocultas
- Función de activación en salida

Número de capas ocultas y neuronas en cada una

La selección del número de capas ocultas y la cantidad de neuronas en cada una va a ser un procedimiento de prueba y error en el que se pueden probar muchas combinaciones. Es cierto que en el caso de trabajar con imágenes y CNN ya hay muchas arquitecturas definidas y probadas que consiguen muy buenos resultados. Por otro lado para tipos de datos estructurados será muy dependiente de esos datos, de forma que será necesario realizar diferentes pruebas para conseguir un buen resultado.

Función de coste y pérdida

En este caso no hay ninguna recomendación especial para las funciones de costes/pérdida y dependerá al igual que en las redes neuronales de la naturaleza de los datos de salida con los que vamos a trabajar.

Optimizador

Se recomienda usar el optimizador **ADAM** (Diederik P. Kingma [2017](#)) que es el que mejores resultados ha dado en las pruebas según ([Walia 2017](#)), consiguiendo una convergencia más rápida que con el resto de optimizadores.

Función de activación en capas ocultas

En un **AE** las funciones de **activación** en las capas ocultas van a conseguir establecer las restricciones **no lineales** al pasar de una capa a la siguiente, normalmente se evita usar la función de activación lineal en las capas intermedias ya que queremos conseguir transformaciones no lineales.

Se recomienda usar la función de activación **ReLU** en las capas ocultas, ya que parece ser que es la que mejores resultados da en la convergencia de la solución y además menor coste computacional tiene a la hora de realizar los cálculos.

Función de activación en salida

En la capa de salida tenemos que tener en cuenta cual es el tipo de datos final que queremos obtener, que en el caso de un AE es el mismo que el tipo de dato de entrada. Normalmente las funciones de activación que se usarán en la **última capa** seran:

- **Lineal** con multiples unidades, para regresión de varios datos numéricos
- **Sigmoid** para valores entre 0 y 1

Tipos de Autoencoders

Una vez entendido el funcionamiento de los AE, veamos algunos de los AE que se pueden construir para diversas tareas.

- Simple
- Multicapa o Profundo
- Sparse
- Convolucional
- Denoising
- Variational

En la descripción de los tipos de AE vamos usar código en **R** y en **python** y el framework **keras** con el backend **Tensorflow**. Todo el código se proporciona aparte.

Usaremos como dataset a **MINIST**, que contiene **60.000/10.000 (entrenamiento/validación)** imágenes de los números del 0 al 9, escritos a mano. Cada imagen tiene un tamaño de $28 \times 28 = 784$ pixels, en escala de grises, con lo que para cada pixel tendremos un valor entre 0 y 255 para definir cuál es su intensidad de gris.

Autoencoder Simple

Vamos a describir como construir un autoencoder **Simple** usando una **red neuronal densamente conectada** en lugar de usar una red neuronal convolucional, para que sea más sencillo comprender el ejemplo. Es decir, vamos a tratar los datos de entrada como si fueran unos datos numéricos que queremos reproducir y no vamos a utilizar ninguna de las técnicas asociadas a las redes convolucionales.

Hay que recordar que las redes convolucionales permiten mediante un tratamiento de las imágenes (convolución, pooling, etc) conseguir mejores resultados que si lo hiciéramos directamente con redes densamente conectadas.

En este caso tendremos una capa de entrada con 784 neuronas (correspondientes a los pixels de cada imagen), una capa intermedia de 32 neuronas, y una capa de salida de nuevo de las 784 neuronas para poder volver a obtener de nuevo los datos originales.

En nuestro ejemplo vamos a tener los siguientes elementos.

- 1 capa de entrada (784 datos), 1 capa oculta (32 datos) y una capa de salida (784 datos)
- La función de coste/pérdida va a ser la **Entropía**
- Usaremos el optimizador **Adam**
- Como función activación intermedia usaremos **ReLU**
- Como función activación de salida **sigmoid** (ya que queremos un valor entre 0 y 1)

Autoencoder Sparse

Ya hemos comentado que una forma de conseguir que un autoencoder **aprenda** estructuras o correlaciones es la reducción del número de neuronas, pero parte de este trabajo también se puede conseguir mediante técnicas de **sparsing** (escasez). Este tipo de técnicas se usan normalmente en las ANN para evitar el **sobreajuste** de nuestro modelo, de forma que en cada actualización de los pesos de la red no se tienen en cuenta todas las neuronas de la capa. Es decir, vamos a conseguir que en las capas que decidamos no todas las neuronas van a estar activadas, de esta manera además de ayudar a evitar el sobreajuste, también conseguiremos crear esas correlaciones que ayudan a construir el autoencoder.

Existen dos métodos básicos para generar el sparse que son:

- Regularización L1
- Regularización L2

Básicamente los dos métodos tratan de hacer que los pesos de las neuronas tengan valores muy pequeños consiguiendo una distribución de pesos más regular. Esto lo consiguen al añadir a la función de pérdida un coste asociado a tener pesos grandes en las neuronas. Este peso se puede construir o bien con la **norma L1 (proporcional al valor absoluto)** o con la **norma L2 (proporcional al cuadrado de los coeficientes de los pesos)**.

Básicamente trabajaremos con un AE Simple, con una red densamente conectada, al que le aplicaremos la **regularización** en su capa oculta.

En nuestro ejemplo vamos a tener los siguientes elementos. - 1 capa de entrada, 1 capaa oculta y una capa de salida - Las capas ocultas tendrán aplicada la **Regularización L2** - La función de coste/pérdida va a ser la **Entropía** - Usaremos el optimizador **Adam** - Como función activación intermedia usaremos **ReLU** - Como función activación de salida **sigmoid** (ya que queremos un valor entre 0 y 1)

Autoencoder Multicapa o profundo

Vamos a pasar ahora a una versión del autoencoder donde habilitamos más capas ocultas y hacemos que el descenso del número de neuronas sea más gradual hasta llegar a nuestro valor deseado, para luego volver a reconstruirlo.

En este caso seguimos con redes densamente conectadas y aplicamos varias capas intermedias reduciendo el número de neuronas en cada una hasta llegar a la capa donde acaba el **encoder** para volver a ir creciendo en las sucesivas capas hasta llegar a la de salida.

En nuestro ejemplo vamos a tener los siguientes elementos. - 1 capa de entrada (784 datos), 5 capa ocultas (32 datos intermedia) y una capa de salida (784 datos) - La función de coste/pérdida va a ser la **Entropía** - Usaremos el optimizador **Adam** - Como función activación intermedia usaremos **ReLU** - Como función activación de salida **sigmoid** (ya que queremos un valor entre 0 y 1)

Autoencoder Convolucional

En nuestro ejemplo al estar trabajando con imágenes podemos pasar a trabajar con **Redes Convolucionales (CNN)** de forma que en lugar de usar las capas **densamente conectadas** que hemos usado hasta ahora, vamos a pasar a usar las capacidades de las redes convolucionales.

Al trabajar con redes convolucionales necesitaremos trabajar con capas de convolución o pooling para llegar a la capa donde acaba el **encoder** para volver a ir creciendo aplicando operaciones de convolución y upsampling (contrario al pooling).

En nuestro ejemplo vamos a tener los siguientes elementos. - 1 capa de entrada, 5 capas ocultas y una capa de salida - La función de coste/pérdida va a ser la **Entropía** - Usaremos el optimizador **Adam** - Como función activación intermedia usaremos **ReLU** - Como función activación de salida **sigmoid** (ya que queremos un valor entre 0 y 1)

Autoencoder Denoising

Vamos a usar ahora un autoencoder para hacer **limpieza en imagen**, es decir, conseguir a partir de una imagen que tiene **ruido** otra imagen sin ese ruido. Entrenaremos al autoencoder para que limpie “ruido” que hay en la imagen y lo reconstruya sin ello.

El **ruido** lo vamos a generar mediante una distribución normal y modificaremos el valor de los pixels de las imágenes. Usaremos estas imágenes con ruido para que sea capaz de reconstruir la imagen original sin ruido con el AE.

Para realizar este proceso lo que haremos será_

- Crear nuevas imágenes con **ruido**
- Entrenar el autoencoder con estas nuevas imágenes
- Calcular el **error** de reconstrucción respecto a las imágenes **originales**

Al estar trabajando con imágenes vamos a partir del Autoencoder de Convolución para poder aplicar el **denosing**.

En nuestro ejemplo vamos a tener los siguientes elementos. - 1 capa de entrada, 5 capas ocultas y una capa de salida - La función de coste/pérdida va a ser la **Entropía** - Usaremos el optimizador **Adam** - Como función activación intermedia usaremos **ReLU** - Como función activación de salida **sigmoid** (ya que queremos un valor entre 0 y 1)

Autoencoder Variational

Los Variational Autoencoder son un tipo de modelo que se denomina generativo, ya que va a permitir construir nuevas imágenes que no existían a partir de otras imágenes con las que se ha entrenado a la red neuronal. En realidad, es un autoencoder que durante el entrenamiento se le

regulariza para evitar un sobreajuste y asegurar que en el **espacio latente (intermedio)** tenga buenas propiedades que permitan un buen proceso generativo.

El proceso de construir este tipo de AE es muy parecido a los que ya hemos visto, con una pequeña diferencia en el paso entre el proceso de **encoder** y el posterior **decoder**. Hasta ahora lo que teníamos era que lo que obteníamos del encoder se lo pasábamos directamente al decoder, en este caso, el resultado del encoder no va a ser realmente un dato, sino una **distribución** de datos. De forma que al decoder no se le pasa directamente lo que ha salido del encoder, si no otro elemento cogido de la distribución generada.

En este caso el proceso sería:

- Se codifica la entrada con el **encoder** no como un dato concreto, sino como una distribución normal (media y desviación)
- Se toma una **muestra de un punto** del espacio latente a partir de la distribución
- Se decodifica el punto de muestra con el **decoder**
- Se calcula la **función de pérdida** con el error de reconstrucción y la parte de regularización
- Se usa el **backpropagation** a través de la red neuronal para ajustar los pesos

FIGURA Nº 90: AUTOENCODER MIXTO (INCOMPLETO Y EXTRA DIMENSIONADO)



Fuente: <https://towardsdatascience.com/understanding-variational-autoencoders-vae-f70510919f73>

Una vez que tenemos entrenado nuestro autoencoder seremos capaces de construir nuevas imágenes partiendo de puntos que estén en la distribución del espacio latente, de forma que esas pequeñas variaciones van a dar lugar a imágenes finales diferentes.

Uso de una máquina de Boltzman Restringida para detección de anomalías

La hipótesis que subyace en el trabajo es que los ritmos anormales van a tener un mayor error de reconstrucción. Así, sería clasificado un ritmo como una anomalía si el error de reconstrucción supera un umbral prefijado a partir de los datos de entrenamiento.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import seaborn as sns

from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import losses

from numpy.random import seed
seed(1)

from tensorflow import random
random.set_seed(2)
```



```
ecg = pd.read_csv('C:/DEEP LEARNING/0. DATOS/ecg.csv', header=None)
ecg.head()
```

	0	1	2	3	4	5	6	7	8	9	...	131	132	133	134	135	136	137	138	139	140
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818287	-1.250522	-0.477492	...	0.792168	0.933541	0.796958	0.578621	0.257740	0.228077	0.123431	0.925286	0.193137	1.0
1	-1.100878	-3.966840	-4.285843	-4.506579	-4.022377	-3.234368	-1.561626	-0.992258	-0.754680	0.042321	...	0.538356	0.656881	0.787490	0.724046	0.555784	0.476333	0.773820	1.119621	-1.436250	1.0
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490658	-1.183580	-0.394229	...	0.886073	0.531452	0.311377	-0.021919	-0.713683	-0.532197	0.321097	0.904227	-0.421797	1.0
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884	-0.965629	...	0.350816	0.499111	0.600345	0.842069	0.952074	0.990133	1.086798	1.403011	-0.383564	1.0
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450	-0.753199	...	1.148884	0.958434	1.059025	1.371682	1.277392	0.960304	0.971020	1.614392	1.421456	1.0

5 rows × 141 columns

```
# Resumen estadístico de las variables
ecg.describe()
```

	0	1	2	3	4	5	6	7	8	9	...	131	132	133	134	135	136
count	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	...	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000	4998.000000
mean	-0.262502	-1.649828	-2.492623	-3.119754	-3.167656	-2.866306	-2.272845	-1.797754	-1.409784	-0.935806	...	0.071312	-0.062622	-0.276086	-0.507750	-0.761323	-0.835173
std	1.152450	1.445542	1.386457	1.302921	1.104535	0.906312	0.731594	0.622794	0.636863	0.657788	...	1.404425	1.646105	1.894515	2.059366	2.086130	2.056296
min	-6.729499	-7.090374	-5.132459	-5.363241	-5.375715	-5.330194	-4.782240	-4.311288	-4.071361	-4.142476	...	-4.167040	-4.557576	-4.530488	-4.563270	-5.150100	-4.880460
25%	-1.004220	-2.701936	-3.668585	-4.227337	-4.007630	-3.480666	-2.779832	-2.165808	-1.773870	-1.362003	...	-1.087720	-1.588237	-2.146835	-2.730750	-3.031224	-2.868603
50%	-0.297541	-1.661892	-2.586129	-3.388210	-3.469899	-2.947061	-2.285427	-1.750084	-1.422457	-0.910384	...	0.658318	0.717808	0.604747	0.334857	-0.030767	-0.195151
75%	0.499909	-0.677998	-1.514187	-2.235690	-2.531153	-2.398797	-1.823480	-1.484107	-1.063592	-0.476477	...	1.169392	1.238651	1.264379	1.234408	1.033957	0.945027
max	4.966414	3.479689	2.660597	1.899798	2.147015	1.614375	1.868728	1.804251	1.683730	1.898627	...	3.007606	2.854680	2.912394	2.937685	3.351437	3.432518

Como se puede ver a continuación, se podría realizar el ejemplo como un problema de aprendizaje supervisado ya que el dataset contiene el etiquetado del problema (variable "140"); si bien, va a ser utilizado para ilustrar conceptos de detección de anomalías.

```
ecg = ecg.rename(columns={140: 'target'})
ecg['target'].value_counts()
1.0      2919
0.0      2079
```

Partición de la muestra: entretenimiento (80%) y test (20%) y normalización de las variables

```

# Partición de la muestra
train_data, test_data = train_test_split(ecg, test_size=0.2,
random_state=123)

# Escalado de las variables
sc = MinMaxScaler().fit(train_data.drop(['target'], axis=1))
train_data_esc = sc.transform(train_data.drop(['target'], axis=1))
test_data_esc = sc.transform(test_data.drop(['target'], axis=1))

```

Se identifican los registros correspondientes con un electrocardiograma normal (la RBM se entrena solo con estos datos).

```

# identificación del target
labels_train = train_data['target']
labels_test = test_data['target']

labels_train = labels_train.astype(bool)
labels_test = labels_test.astype(bool)

# Datos normales y anómalos
normal_train_data = train_data_esc[labels_train]
normal_test_data = test_data_esc[labels_test]

anomalous_train_data = train_data_esc[~labels_train]
anomalous_test_data = test_data_esc[~labels_test]

# conversión de datos a float32 para funcionamiento correcto de la RBM
# construida en tf2
normal_train_data = normal_train_data.astype('float32')
normal_test_data = normal_test_data.astype('float32')
anomalous_train_data = anomalous_train_data.astype('float32')
anomalous_test_data = anomalous_test_data.astype('float32')

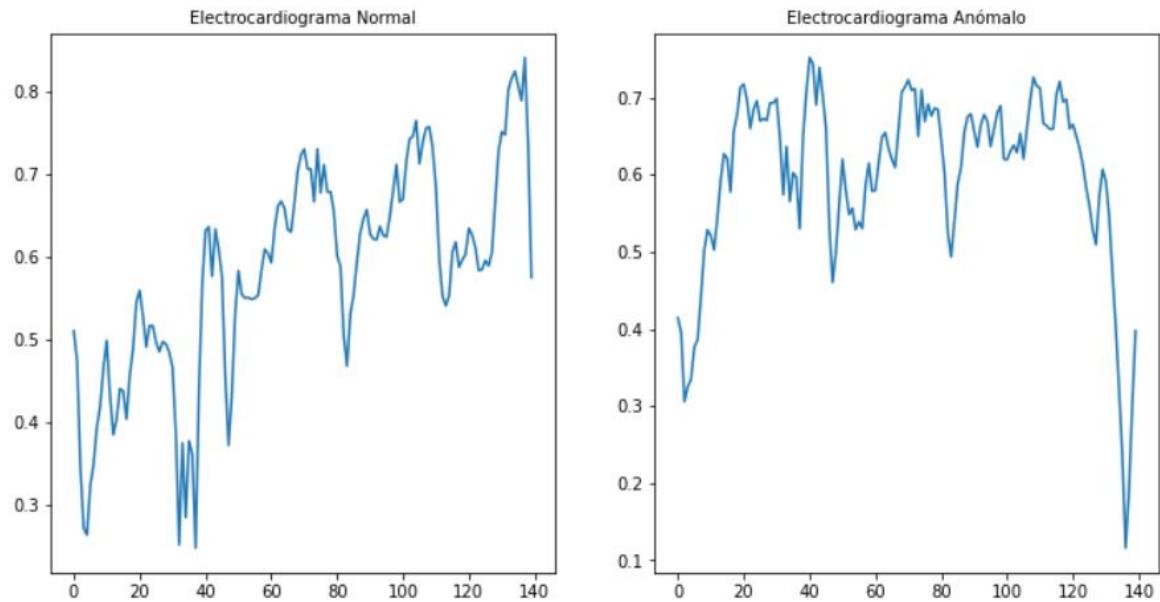
Se grafican los electrocardiogramas (normal y anómalo).

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))

ax1.plot(np.arange(normal_train_data.shape[1]), normal_train_data[0])
ax2.plot(np.arange(anomalous_train_data.shape[1]), anomalous_train_data[0])
ax1.set_title('Electrocardiograma Normal', fontsize=10)
ax2.set_title('Electrocardiograma Anómalo', fontsize=10)

plt.show()

```



Construcción del modelo. Se construye el autoencoder siguiendo la API modular de Tensorflow

```

entrada = normal_train_data.shape[1]

# fase de comprensión de los datos
input_encoder = tf.keras.Input(shape=(entrada,)) # capa de entrada del
autoencoder
enc1 = tf.keras.layers.Dense(units = 32, activation =
"relu")(input_encoder)
encoder = tf.keras.layers.Dense(units = 8, activation = "relu")(enc1)

# modelo encoder
encoder_model = tf.keras.Model(inputs=input_encoder, outputs=encoder)

# fase decodificador de los datos
input_decoder = tf.keras.Input(shape=(8,))
dec1= tf.keras.layers.Dense(units = 32, activation = "relu")(input_decoder)
decoder = tf.keras.layers.Dense(units = entrada, activation =
"sigmoid")(dec1)

decoder_model = tf.keras.Model(inputs = input_decoder, outputs = decoder)

# Modelo autoencoder
encoder_decoder = input_encoder
encoder_decoder = encoder_model(encoder_decoder)
encoder_decoder = decoder_model(encoder_decoder)

autoencoder = tf.keras.Model(inputs = input_encoder, outputs =
encoder_decoder)

# Nota: otra forma de construir el autoencoder (apilamiento de capas)
# recordar que siguiendo la API funcional un modelo se obtiene mediante la
unión de capas

# inputs = tf.keras.Input(shape=(entrada,))
# hidden1 = tf.keras.layers.Dense(32, activation="relu")(inputs)
# hidden2 = tf.keras.layers.Dense(8, activation="relu")(hidden1)

```

```

# hidden3 = tf.keras.layers.Dense(32, activation="relu") (hidden2)
# capas_encoder_decoder = tf.keras.layers.Dense(entrada,
activation="sigmoid") (hidden3)

# autoencoder2 = tf.keras.Model(inputs=inputs,
outputs=capas_encoder_decoder)

```

Se presenta un resumen de las capas, neuronas y parámetros del modelo creado.

```
autoencoder.summary()
```

```

Model: "model_4"

Layer (type)          Output Shape         Param #
=====            =====
input_5 (InputLayer)    [(None, 140)]        0
model_2 (Functional)      (None, 8)           4776
model_3 (Functional)      (None, 140)          4908
=====
Total params: 9,684
Trainable params: 9,684
Non-trainable params: 0
=====
```

Finalmente, se compila y se procede a entrenar el modelo.

```

autoencoder.compile(optimizer='adam', loss='mae')

hist = autoencoder.fit(normal_train_data,
                      normal_train_data,
                      epochs=50,
                      validation_split=0.2,
                      batch_size=128,

callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=15)])

```

```

Epoch 1/50
15/15 [=====] - 1s 26ms/step - loss: 0.0411 - val_loss: 0.0400
Epoch 2/50
15/15 [=====] - 0s 13ms/step - loss: 0.0391 - val_loss: 0.0393
Epoch 3/50
15/15 [=====] - 0s 17ms/step - loss: 0.0387 - val_loss: 0.0384
Epoch 4/50
15/15 [=====] - 0s 21ms/step - loss: 0.0375 - val_loss: 0.0378
Epoch 5/50
15/15 [=====] - 0s 23ms/step - loss: 0.0365 - val_loss: 0.0374
Epoch 6/50
15/15 [=====] - 0s 17ms/step - loss: 0.0367 - val_loss: 0.0370
Epoch 7/50

```

```
15/15 [=====] - 0s 15ms/step - loss: 0.0361 -  
val_loss: 0.0365  
Epoch 8/50  
15/15 [=====] - 0s 12ms/step - loss: 0.0358 -  
val_loss: 0.0360  
Epoch 9/50  
15/15 [=====] - 0s 16ms/step - loss: 0.0359 -  
val_loss: 0.0354  
Epoch 10/50  
15/15 [=====] - 0s 20ms/step - loss: 0.0347 -  
val_loss: 0.0347  
Epoch 11/50  
15/15 [=====] - 0s 17ms/step - loss: 0.0344 -  
val_loss: 0.0339  
Epoch 12/50  
15/15 [=====] - 0s 19ms/step - loss: 0.0335 -  
val_loss: 0.0330  
Epoch 13/50  
15/15 [=====] - 0s 15ms/step - loss: 0.0326 -  
val_loss: 0.0322  
Epoch 14/50  
15/15 [=====] - 0s 19ms/step - loss: 0.0314 -  
val_loss: 0.0313  
Epoch 15/50  
15/15 [=====] - 0s 18ms/step - loss: 0.0307 -  
val_loss: 0.0306  
Epoch 16/50  
15/15 [=====] - 0s 13ms/step - loss: 0.0303 -  
val_loss: 0.0301  
Epoch 17/50  
15/15 [=====] - 0s 10ms/step - loss: 0.0294 -  
val_loss: 0.0297  
Epoch 18/50  
15/15 [=====] - 0s 13ms/step - loss: 0.0293 -  
val_loss: 0.0295  
Epoch 19/50  
15/15 [=====] - 0s 12ms/step - loss: 0.0288 -  
val_loss: 0.0292  
Epoch 20/50  
15/15 [=====] - 0s 9ms/step - loss: 0.0292 -  
val_loss: 0.0289  
Epoch 21/50  
15/15 [=====] - 0s 10ms/step - loss: 0.0286 -  
val_loss: 0.0286  
Epoch 22/50  
15/15 [=====] - 0s 9ms/step - loss: 0.0283 -  
val_loss: 0.0284  
Epoch 23/50  
15/15 [=====] - 0s 10ms/step - loss: 0.0278 -  
val_loss: 0.0283  
Epoch 24/50  
15/15 [=====] - 0s 11ms/step - loss: 0.0276 -  
val_loss: 0.0280  
Epoch 25/50  
15/15 [=====] - 0s 10ms/step - loss: 0.0275 -  
val_loss: 0.0279  
Epoch 26/50  
15/15 [=====] - 0s 11ms/step - loss: 0.0275 -  
val_loss: 0.0276  
Epoch 27/50
```

```
15/15 [=====] - 0s 9ms/step - loss: 0.0269 -
val_loss: 0.0275
Epoch 28/50
15/15 [=====] - 0s 9ms/step - loss: 0.0267 -
val_loss: 0.0274
Epoch 29/50
15/15 [=====] - 0s 11ms/step - loss: 0.0272 -
val_loss: 0.0274
Epoch 30/50
15/15 [=====] - 0s 9ms/step - loss: 0.0269 -
val_loss: 0.0272
Epoch 31/50
15/15 [=====] - 0s 8ms/step - loss: 0.0268 -
val_loss: 0.0270
Epoch 32/50
15/15 [=====] - 0s 8ms/step - loss: 0.0267 -
val_loss: 0.0268
Epoch 33/50
15/15 [=====] - 0s 8ms/step - loss: 0.0264 -
val_loss: 0.0269
Epoch 34/50
15/15 [=====] - 0s 9ms/step - loss: 0.0262 -
val_loss: 0.0267
Epoch 35/50
15/15 [=====] - 0s 10ms/step - loss: 0.0263 -
val_loss: 0.0266
Epoch 36/50
15/15 [=====] - 0s 8ms/step - loss: 0.0261 -
val_loss: 0.0265
Epoch 37/50
15/15 [=====] - 0s 8ms/step - loss: 0.0262 -
val_loss: 0.0265
Epoch 38/50
15/15 [=====] - 0s 7ms/step - loss: 0.0266 -
val_loss: 0.0264
Epoch 39/50
15/15 [=====] - 0s 7ms/step - loss: 0.0258 -
val_loss: 0.0263
Epoch 40/50
15/15 [=====] - 0s 13ms/step - loss: 0.0264 -
val_loss: 0.0262
Epoch 41/50
15/15 [=====] - 0s 13ms/step - loss: 0.0263 -
val_loss: 0.0261
Epoch 42/50
15/15 [=====] - 0s 7ms/step - loss: 0.0259 -
val_loss: 0.0260
Epoch 43/50
15/15 [=====] - 0s 8ms/step - loss: 0.0259 -
val_loss: 0.0260
Epoch 44/50
15/15 [=====] - 0s 17ms/step - loss: 0.0258 -
val_loss: 0.0259
Epoch 45/50
15/15 [=====] - 0s 9ms/step - loss: 0.0258 -
val_loss: 0.0258
Epoch 46/50
15/15 [=====] - 0s 8ms/step - loss: 0.0257 -
val_loss: 0.0258
Epoch 47/50
```

```

15/15 [=====] - 0s 15ms/step - loss: 0.0255 -
val_loss: 0.0257
Epoch 48/50
15/15 [=====] - 0s 8ms/step - loss: 0.0254 -
val_loss: 0.0257
Epoch 49/50
15/15 [=====] - 0s 8ms/step - loss: 0.0252 -
val_loss: 0.0256
Epoch 50/50
15/15 [=====] - 0s 10ms/step - loss: 0.0252 -
val_loss: 0.0255

```

Se muestra el resumen de la bondad de ajuste en el entrenamiento del modelo:

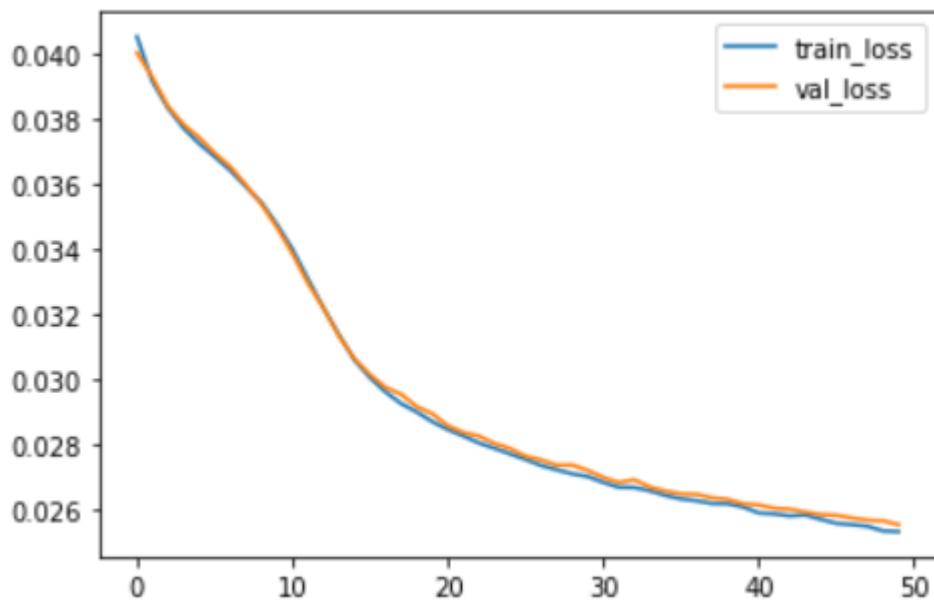
```

# construcción de un data.frame
df_train=pd.DataFrame(hist.history)
df_train['epochs']=hist.epoch

plt.plot(df_train['epochs'], df_train['loss'], label='train_loss')
plt.plot(df_train['epochs'], df_train['val_loss'], label='val_loss')

plt.legend(loc="upper right")
plt.show()

```

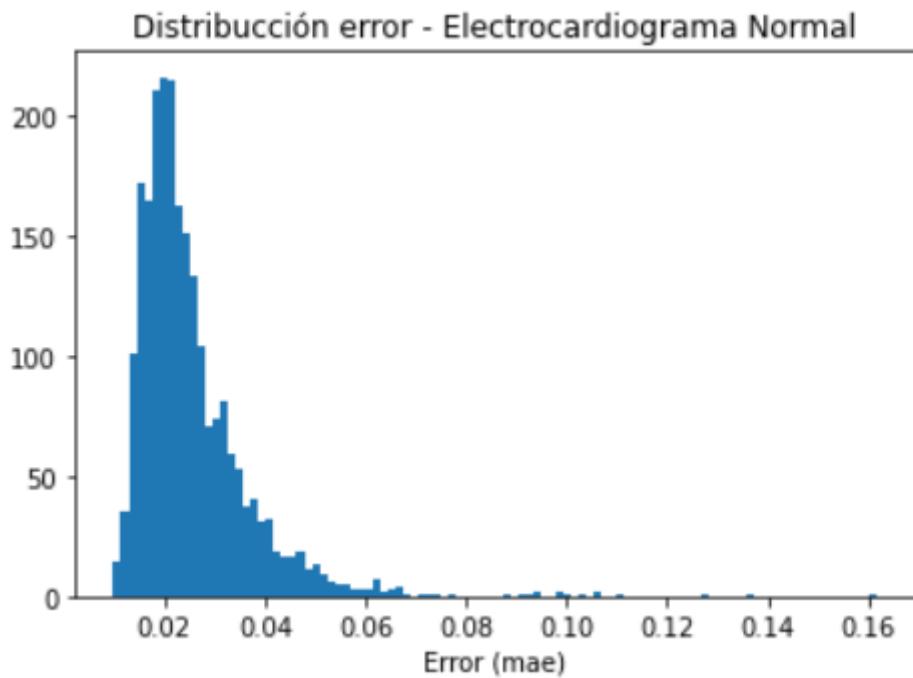


Se reconstruyen los datos de entrenamiento y se calcula su error en la reconstrucción

```

# Predicción con la muestra de electrocardiogramas normales
out = rbm.rbm_reconstruct(normal_train_data)
out = np.array(out)
train_loss = tf.keras.losses.mae(out, normal_train_data)
plt.hist(np.array(train_loss), bins=100)
plt.title('Distribución error - Electrocardiograma Normal')
plt.xlabel('Error (mae)')
plt.show()

```

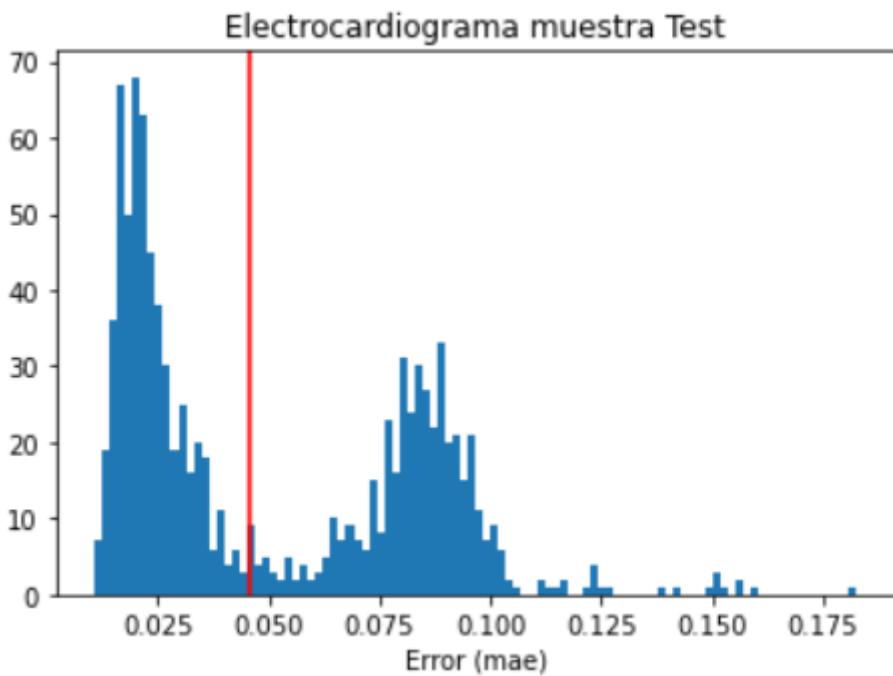


En base a la distribución de pérdidas, se establece un umbral para el cual los electrocardiogramas de los pacientes deberían ser revisados.

```
# uso de la mediana porque la distribución no es simétrica
umbral = np.round(np.median(train_loss) + 2*np.std(train_loss),5)
print('Umbral:', umbral)
Umbral: 0.04597
```

Se aplica el modelo a la muestra de test. Reconstrucción de los datos y cálculo del error

```
test_data_esc = test_data_esc.astype('float32')
out_test = rbm.rbm_reconstruct(test_data_esc)
out_test = np.array(out_test)
test_loss = tf.keras.losses.mae(out_test, test_data_esc)
plt.hist(np.array(test_loss), bins=100)
plt.axvline(x=umbral, color='red')
plt.title('Electrocardiograma muestra Test')
plt.xlabel('Error (mae)')
plt.show()
```



Así, a partir del error de reconstrucción observamos que existen muchos cardioramas con un error superior a 0.1

Por último, mostramos la bondad de nuestro modelo mediante diferentes métricas:

```

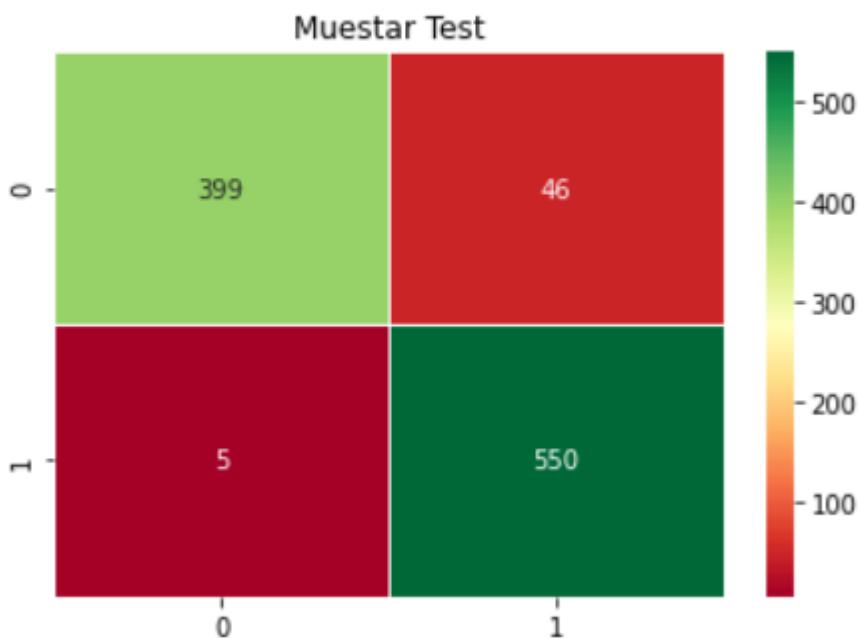
Accuracy, precision, recall
Matriz de confusión
def predict(model, data, umbral):
    reconstructions = rbm.rbm_reconstruct(data)
    loss = tf.keras.losses.mae(reconstructions, data)
    return np.array(tf.math.less(loss, umbral))

def print_sol(predictions, labels):
    print("Accuracy = {}".format(np.round(accuracy_score(labels, preds),3)))
    print("Precision = {}".format(np.round(precision_score(labels, preds),3)))
    print("Recall = {}".format(np.round(recall_score(labels, preds),3)))
preds = predict(RBM, test_data_esc, umbral)
print_sol(preds, labels_test)

# matriz de confusión
sns.heatmap(confusion_matrix(preds, labels_test),
            annot = True, cmap = "RdYlGN", fmt='0f')
plt.title('Muestar Test')
plt.show()

```

Accuracy = 0.949
 Precision = 0.991
 Recall = 0.923



7.7. Redes Generativas Adversarias

Las Redes Generativas Adversarias (GANs, por sus siglas en inglés) son un tipo de arquitectura de aprendizaje profundo capaz de aprender la manera de reproducir datos sintéticos muy parecidos a los reales. Estas redes fueron introducidas por (Goodfellow, 2016) y, en solo tres años, más de 40.000 trabajos de investigación fueron publicados sobre las mismas. Por todo ello, Yann LeCun definió las GANs como unas de las ideas más interesantes de los últimos diez años en el campo del Machine Learning. En la actualidad, estas redes son aplicadas tanto para la creación de imágenes, vídeos, música, etc; de forma que son empleadas en ámbitos tan diversos como traducción de imagen a imagen, superresolución de imagen, descubrimiento de fármacos e incluso predicción del siguiente fotograma en vídeo.

Antes de desarrollar en profundidad este tipo de arquitecturas, se presentan las diferencias entre los modelos discriminativos y generativos:

- **Modelo discriminativo:** aquel modelo que estima de forma directa la probabilidad condicional, $P(y|\bar{X})$, de la etiqueta y dada la característica \bar{X} . Un ejemplo de este tipo de modelos sería la regresión logística.
- **Modelo generativo:** aquel modelo que estima la probabilidad conjunta $P(\bar{X}|y)$; la cual es una probabilidad generativa de una instancia de los datos. Mediante el Teorema de Bayes, la probabilidad conjunta puede ser utilizada para estimar la probabilidad

condicional de la etiqueta y data \bar{X} . Por su parte, el clasificador Naïve Bayes sería un ejemplo de este tipo de modelos.

$$P(y|\bar{X}) = \frac{P(\bar{X}|y)}{P(\bar{X})}$$

Destacar que los modelos generativos pueden ser empleados tanto para problemas supervisados como no supervisados mientras que los modelos discriminativos únicamente pueden ser utilizados para el primero de los casos.

Fundamento y entrenamiento de una GAN

Las Redes Generativas Adversarias son dos modelos de redes neuronales que trabajan de forma simultánea. El primero de ellos, es el modelo generativo el cual produce ejemplos sintéticos similares a los ejemplos reales. El propósito de dicho modelo es crear muestras sintéticas lo más realista posibles para que dificulte al observador distinguir si la muestra pertenece al conjunto real o al generado. Por su parte, el segundo modelo es también conocido como red discriminatoria y se encarga de distinguir las muestras ficticias de las reales. Para poder realizar este proceso, este segundo modelo ha sido entrenado con un dataset en el cual se tienen clasificadas las instancias según su procedencia (real o ficticia).

Por tanto, esta red se puede entender como el juego entre ese “falsificador” que intenta producir muestras ficticias y un “policía” que le persigue. Así, ambas redes son adversarias; haciendo uno al otro cada vez mejor hasta que se alcanza un equilibrio entre ellas.

El hecho de que la red discriminativa identifica una muestra sintética como falsa siendo falsa es utilizado por la red generativa para modificar sus pesos, de modo que la red discriminativa tendrá más dificultades para clasificar las muestras generadas a partir de ella. Después de modificar los pesos de la red generadora, se crean nuevas muestras el proceso se repite. Con el tiempo, la red generativa mejora cada vez más en la producción de falsificaciones. Eventualmente, se vuelve imposible para el discriminador distinguir entre real y objetos generados sintéticamente hasta que alcanzar el equilibrio.

Como se ha dicho, el proceso de entrenamiento de este tipo de redes tiene lugar actualizando alternativamente los parámetros del generador y el discriminador. El discriminador es una red neuronal con entradas de d-dimensiones y una única salida en $(0, 1)$, que indica la probabilidad de que el ejemplo de entrada es real; esto es, 1 indica que el ejemplo es real y 0 cuando es sintético. Por su parte, el generador toma como entrada muestras con ruido de una distribución de probabilidad p-dimensional (distribución gaussiana), y lo utiliza para generar ejemplos de d-dimensiones de los datos.

Destacar que el comportamiento del generador de una red GAN puede entenderse como el decodificador de un autoencoder variacional, VAE. Sin embargo, el proceso de entrenamiento es ligeramente diferente; mientras que en el VAE el error de reconstrucción es usado para mejorar entrenamiento, el error del discriminador se emplea para entrenar al generador para crear otras muestras como la distribución de datos de entrada.

Tras explicar el funcionamiento de este tipo de arquitectura desde un punto de vista funcional, se expresa matemáticamente las funciones objetivo que en las que se basa la modelización de este algoritmo:

Sea $D(\bar{X})$ la salida del discriminador para la entrada \bar{X} , R_m m ejemplos muestreados al azar del conjunto de datos reales y S_m m muestras sintéticas que se generan utilizando el generador. Destacar que para el proceso de generación de las muestras inicialmente se crea un conjunto N_m de p -dimensional muestras de ruido $\{\bar{Z}_m \dots \bar{Z}_m\}$ y, después, a dichas muestras se le aplica el generador para crear las muestras de $S_m = \{G(\bar{Z}_1) \dots G(\bar{Z}_m)\}$

Por tanto, la función objetivo del discriminador, J_D , tiene la forma:

$$\text{Max}_D J_D = \sum_{\bar{X} \in R_m} \log [D(\bar{X})] + \sum_{\bar{X} \in S_m} \log [1 - D(\bar{X})]$$

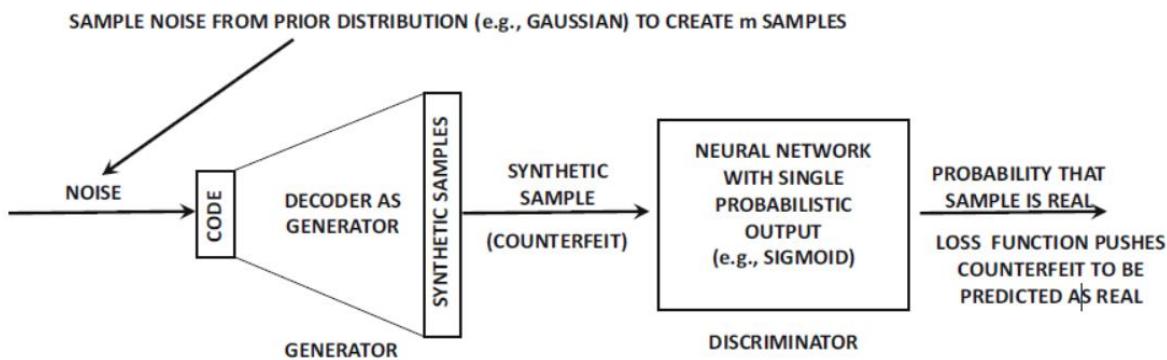
donde el primer sumando denota las m muestras de instancias reales y el segundo, las m de muestras de instancias sintéticas. Por tanto, cuando todas las instancias reales sean clasificadas como 1 y todas las instancias sean clasificadas como 0 la función objetivo será máxima.

Por su parte, la red generadora crea m instancias sintéticas, S_m y como lo que se pretende es engañar al discriminador; es decir, que el discriminador las identifique como reales se define la siguiente función objetivo, J_G :

$$\text{Min}_G J_G = \sum_{\bar{X} \in S_m} \log [1 - D(\bar{X})] = \sum_{\bar{Z} \in N_m} \log [1 - D(G(\bar{Z}))]$$

Así, como el objetivo del generador es engañar al discriminador, su función objetivo se hace mínima cuando los ejemplos sintéticos son clasificados erróneamente a 1.

Por ello, el problema de optimización general se formula como un juego minimax sobre J_D . A su vez, como $J_D - J_G$ no incluye ninguno de los parámetros del generador G , maximizar J_G es lo mismo que maximizar J_D ; de forma que se puede escribir el problema de optimización general (sobre el generador y el discriminador) como $\text{Min}_G \text{Max}_D J_D$.



Fuente: Argwall, 2018

El gradiente estocástico ascendente se emplea para el aprendizaje de los parámetros de la red discriminadora mientras que el gradiente estocástico descendente para aprender los parámetros del generador; alternándose así los pasos de actualización del gradiente entre el generador y el

descriminador. Sin embargo, en la práctica, k pasos en el descritor (típicamente menor a 5) son empleados por cada paso del generador. Se sigue el siguiente proceso:

- Discrminador (se repite k veces): un mini-lote de tamaño $2m$ se construye con un número igual de instancias reales y sintéticas. Las instancias sintéticas se crean introduciendo ruido a las muestras del generador de la distribución anterior, mientras que las muestras reales se seleccionan del dataset de partida. El ascenso del gradiente estocástico se realiza en los parámetros del discriminador para maximizar la probabilidad de que el discriminador clasifique de forma correcta todas las instancias (reales y sintéticas). Para cada paso de actualización, el proceso se consigue mediante la retropropagación en la red discriminadora con respecto al mini-lote de $2m$ de muestras reales y sintéticas.
- Generador (se realiza una vez): al generador se le proporciona m muestras de ruido para crear m ejemplos sintéticos. Se realiza el descenso de gradiente estocástico en los parámetros del generador para minimizar la probabilidad de que el discriminador clasifique correctamente los ejemplos sintéticos. En este sentido, como se ha dicho, al minimizar su función objetivo ayuda a que las instancias generadas se predigan después como reales. La retropropagación calculará automáticamente los gradientes con respecto a las redes generadora y discriminadora para esta configuración conectada, pero solo se actualizan los parámetros de la red de generadores.

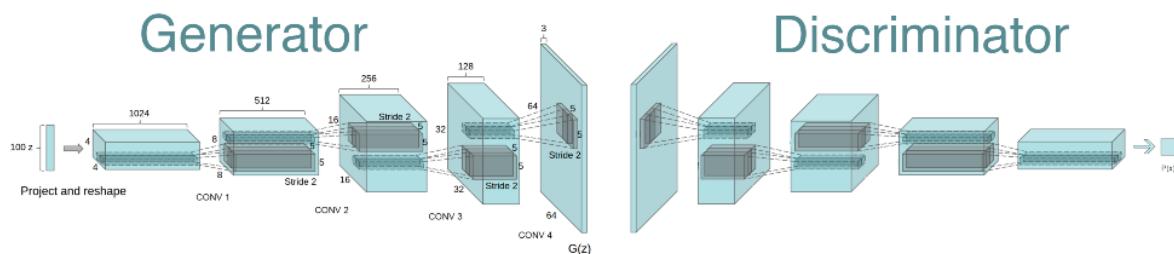
Arquitecturas de las GAN

Deep convolutional GAN – DCGAN

Las DCGANs fueron propuestas en el año 2016 y son una de las arquitecturas de GANs más populares. Los cambios más importantes que se introdujeron en esta arquitectura fueron:

- La red está constituida solo de capas convolucionales. Las capas de pooling se remplazaron por convoluciones escalonadas en el discriminador y convoluciones transpuestas en el generador.
- Se eliminan las capas densamente conectadas usadas en tareas de clasificación
- Se emplea batch normalization tras las capas convolucionales para ayudar al gradiente
- Todas las capas del generador usan la función ReLU como función de activación (salvo en el output que se emplea la tangente hiporbólica)
- Se emplea la función LeakyReLU en todas las capas del discriminador

FIGURA Nº 91. ARQUITECTURA DE UNA DCGAN (GENERADOR Y DISCRIMINADOR)



Fuente: Deep Learning with Tensorflow and keras, 2019

Deep convolutional GAN – DCGAN

La arquitectura de las SRGANs consiste en tres redes neuronales: una red generadora muy profunda basado en un modelo residual (estilo ResNet), una red descriminadora y una red VGG-16 preentrenada.

Esta red emplea como función de pérdida, la pérdida porcentual, la cual es calculada como:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR}$$

El primer término hace referencia a la pérdida de contenido, obtenida utilizando el patrón de características por la red preentrenada VVG19. Matemáticamente, se define como la distancia euclídea entre el patrón de características de la imagen reconstruida y la imagen original de alta resolución. Por su parte, el segundo término es la pérdida asociada a la red adversaria; esto es, es el término estándar de pérdida generativa, diseñado para asegurar que las imágenes generadas por el generador puedan engañar al discriminador.

FIGURA N° 92. EJEMPLO DE UNA IMAGEN GENERADA POR UNA SRGAN



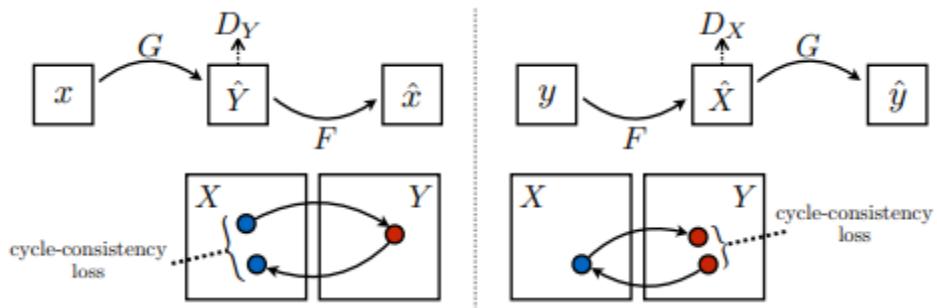
Fuente: Deep Learning with Tensorflow and keras, 2019

CycleGAN

Este tipo de arquitectura realiza la traslación de la imagen, es decir, partiendo de una imagen en un dominio se transfiere a otro en ausencia de ejemplos de entrenamiento. La capacidad de realizar tal traslación en ausencia de datos de entrenamiento es lo que la hace única.

Para lograr la transferencia de imágenes se emplean dos GANs de forma que cada generador realice la traslación de la imagen de un dominio al otro. Así, si tomamos como X la entrada el generador de la primera GAN realiza una traslación $G: X \rightarrow Y$, siendo la salida $Y=G(X)$. En cambio, el generador de la segunda GAN realiza la transformación inversa $F: Y \rightarrow X$, resultando $X=F(Y)$. Por otro lado, cada descriminador es entrenado para distinguir entre las imágenes reales y sintéticas. El funcionamiento de este tipo de arquitecturas se muestra en la siguiente imagen:

FIGURA. ARQUITECTURA DE CYCLEGAN

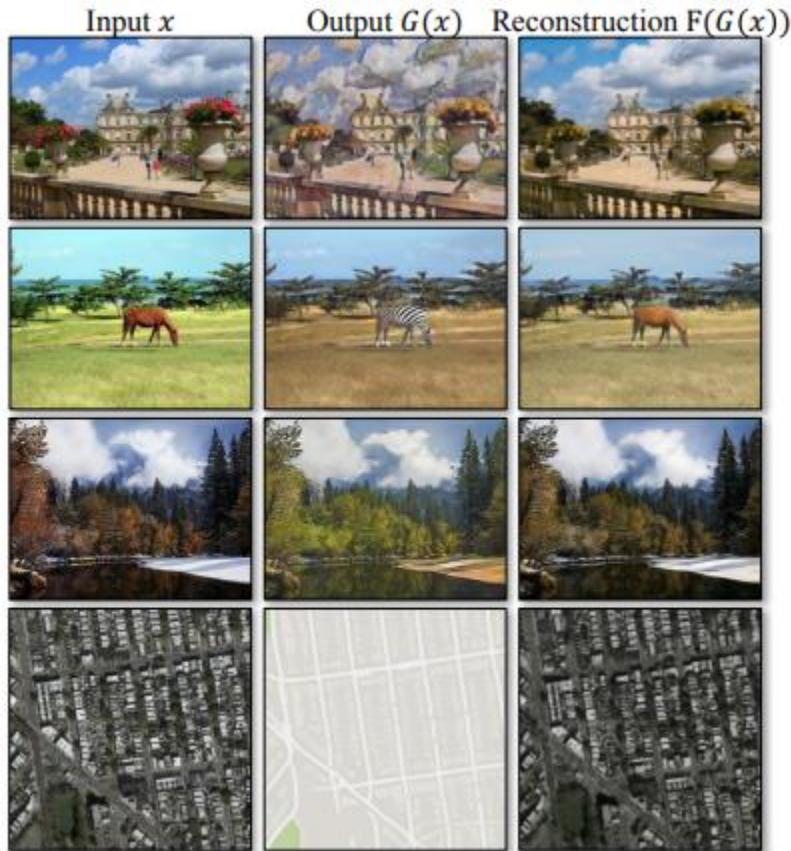


Fuente: Deep Learning with Tensorflow and keras, 2019

Destacar también que para entrenar las GAN combinadas se añade a la pérdida adversaria de la GAN convolucional, una pérdida de consistencia del ciclo (forward y backward) para asegurar que, si se da una imagen X como entrada, tras realizarse las dos transformaciones $F(G(X)) \sim X$, la imagen sigue siendo la misma. De igual forma para las transformaciones $F(G(Y)) \sim Y$.

Por último, se muestran algunas imágenes generadas mediante el uso de CycleGANs:

FIGURA N° 93. APLICACIONES DE CYCLEGAN



Fuente: Deep Learning with Tensorflow and keras, 2019

7.8. Actualidad y algunos conceptos relacionados con el Deep Learning

El Deep Learning ha venido para quedarse. Cada vez está más implicado en todos los ámbitos del conocimiento apareciendo continuas aplicaciones en todos los sectores de la economía y en general de todas las ciencias.

En los últimos meses ha habido una explosión de las Inteligencias Artificiales, sobre todo a partir de la irrupción de ChatGPT (<https://chat.openai.com>), que ha conseguido llegar al usuario final y no quedarse en el ámbito investigador o empresarial. Este software ha conseguido que salgan a la luz otras muchas Inteligencias Artificiales que trabajan en diferentes ámbitos y que están consiguiendo que cada vez más usuarios las utilicen.

Todas estas Inteligencias Artificiales de alguna u otra manera están basadas en Deep Learning, ya que trabajan sobre problemas o datos bastante complejos y necesitan trabajar con Redes Neuronales Profundas para conseguir unos buenos resultados.

Algunos ejemplos de las Inteligencias Artificiales que se pueden usar son:

ChatGPT (<https://chat.openai.com>)

Básicamente entiende el lenguaje natural escrito y contesta a tus preguntas. Recopila y aprende a partir de preguntas, respuestas y correcciones. Posee una inmensa cantidad de datos e información con la que puede contestar a prácticamente cualquier consulta o pregunta de forma detallada, concisa y natural.

Bing Chat (<https://www.bing.com/?scope=web&FORM=HDRSC2>)

Es la IA conversacional de Microsoft.

Bing Image Creator (<https://www.bing.com/images/create>)

También de Microsoft, crea imágenes a partir de una descripción de lo que se quiere conseguir. Las imágenes generadas realmente no existen y están basadas en tu descripción.

Image Elarger (<https://www.imageenlarger.com/es.html>)

Permite agrandar imágenes sin perder detalles, pixelar o distorsionar.

Github Copilot (<https://github.com/features/copilot>)

Es un asistente que ayuda a escribir código de programación en tiempo real. También puede revisar tu código para sugerirte código para completarlo.

Algunas referencias a más Inteligencias Artificiales las podéis encontrar en este artículo:

<https://www.xataka.com/basics/mega-guia-71-herramientas-inteligencia-artificial-dime-necesitas-te-digo-que-ias-mejores>

Aparte de lo anteriormente explicado sobre la situación actual donde nos encontramos, el concepto del estado del arte es una compilación de resultados de otras investigaciones sobre el tema de Deep learning que hemos intentado sintetizar en esta parte del módulo. Así que, en este pequeño apartado, y después de resumir las principales arquitecturas de redes neuronales aplicadas al Deep learning, vamos a intentar ofrecer de forma somera algunos conceptos y técnicas relacionadas con el aprendizaje profundo que son importantes desde hace muy poco tiempo y que parecen iniciar vías importantes de aplicaciones prácticas.

Un concepto importante es el **Deep Reinforcement Learning** (Deep RL). El aprendizaje de refuerzo profundo es la combinación del aprendizaje de refuerzo (RL) y el aprendizaje profundo. Este campo de investigación puede resolver una amplia gama de complejas tareas de toma de decisiones que antes estaban fuera del alcance de los ordenadores. Por lo tanto, Deep RL abre muchas aplicaciones nuevas en dominios como salud, robótica, redes inteligentes, finanzas y muchos más

Otro concepto a tener en cuenta es el **Deep Dream** que según wikipedia lo describe de la siguiente manera *“El software Deep Dream fue desarrollado para el imageNet large scale visual recognition challenge (ILSVRC). Este era un desafío reto, propuesto a diferentes equipos de investigación, que consistió en crear un sistema de reconocimiento de objetos y su localización dentro de una misma imagen, aparte de su detección inmediata. En este Desafío se adjudicó a Google el primer premio en el año 2014, logrado gracias al uso del entrenamiento de redes neuronales. En junio de 2015 Google publicó la investigación, y tras esto hizo su código fuente abierto utilizado para generar las imágenes en un IPython notebook. Con esto se permitió que las imágenes de la red neuronal pudiesen ser creadas por cualquiera. El código se basa en Caffe y utiliza paquetes de códigos abiertos disponibles. Además, está diseñado para tener el menor número de dependencias posible”*

Una forma de transformar imágenes es a través del algoritmo conocido como Neural Style Transfer, el cual consiste en transferir estilos artísticos de una imagen. Generalmente se utilizan cuadros de pintores reconocidos como Vincent Van Gogh, Claude Monet, entre otros, a cualquier imagen transformándola y dando resultados a veces muy estéticos. En la página web de François Chollet creador de la librería Keras se puede encontrar la implementación de este algoritmo, así como diversos ejemplos donde se observa la fusión de dos imágenes para generar otra conteniendo características de las dos.

FIGURA Nº 94: EJEMPLO DE NEURAL STYLE TRANSFER

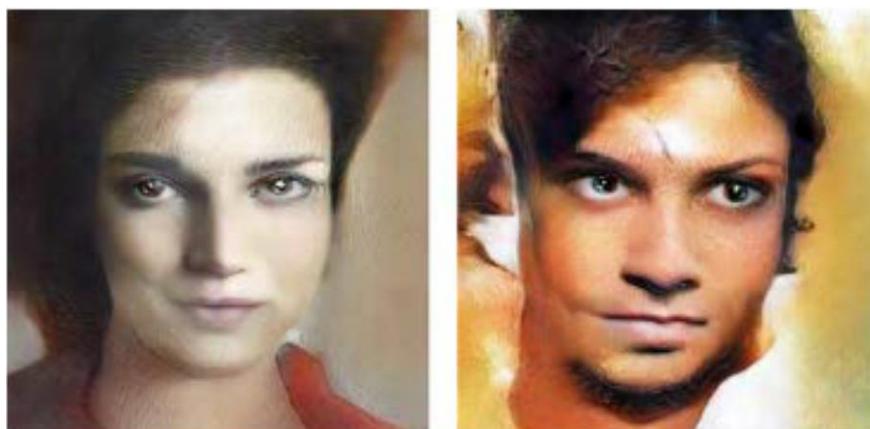


Figure 8.8 Some example results

Fuente: <https://github.com/fchollet/deep-learning-with-python-notebooks>

Otra arquitectura que se ha incorporado recientemente al mundo del Deep Learning son las GAN (Generative Adversarial Networks) que pertenecen a los llamados modelos generativos y que pueden crear videos, música y caras sintéticas extremadamente realistas. Las Redes Neuronales Generativas Adversarias son una forma nueva de usar deep learning para generar imágenes que parecen reales.

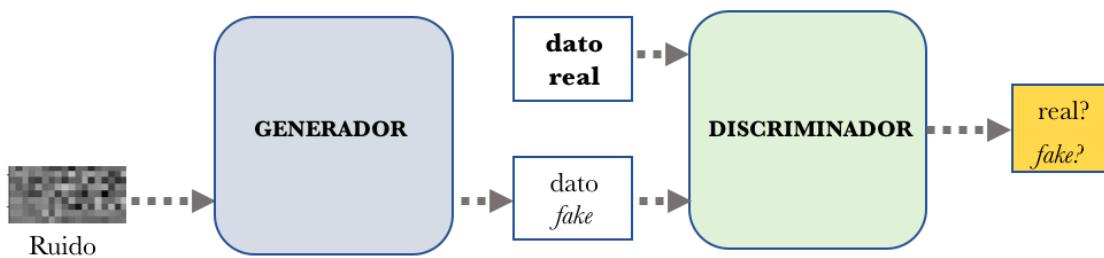
FIGURA Nº 92: IMÁGENES GENERADAS POR MIKE TYKA USANDO UNA RED GAN



Fuente: www.miketyka.com

Esta arquitectura se compone de un generador y un modelo discriminador. El generador es responsable de crear nuevos resultados, como imágenes, que posiblemente provengan del conjunto de datos original. O toma como entrada un vector aleatorio (un punto aleatorio en el espacio latente), y lo decodifica en una imagen sintética. El modelo de generador se implementa normalmente utilizando una red neuronal convolucional profunda y capas especializadas en resultados que aprenden a completar las características de una imagen en lugar de extraerlas de una imagen de entrada.

FIGURA Nº 95: ESQUEMA DE UNA ARQUITECTURA GENERAL DE UNA GAN



Fuente: Torres. J. (2020)

Los dos tipos más comunes de capas que se pueden usar en el modelo del generador son una capa de muestreo (UpSampling2D) que simplemente duplica las dimensiones de la entrada y la capa convolucional de transposición (Conv2DTranspose) que realiza una operación de convolución inversa. El modelo discriminador toma como entrada una imagen (real o sintética), y predice si la imagen provino del conjunto de entrenamiento o si fue creada por la red de generadores.

7.9. Software para aplicar Deep Learning.

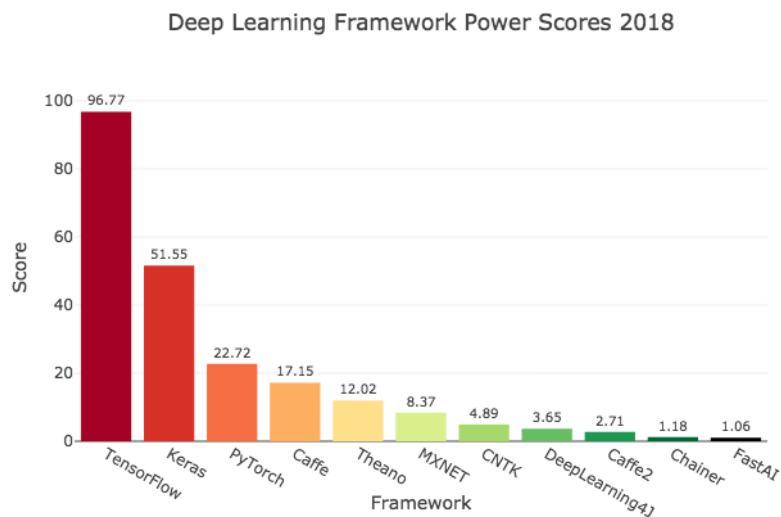
7.9.1. Introducción

Este epígrafe es un pequeño resumen del software para aplicar Deep Learning. Sabemos que conviven multitud de APIs y lenguajes de programación que realizan actualmente tareas de aprendizaje profundo. Existen diferentes productos en el mercado, liderados desde el inicio del Deep learning por Tensorflow de Google, aunque recientemente están tomando impulso algunos planteamientos como Pytorch.

Como habrás podido comprobar en la documentación presentada en este epígrafe la opción preferida para este módulo de Deep Learning es el software llamado Keras que está programado en Python pero que dispone de una API para otros lenguajes como R o Scala. En términos de eficiencia y de aprendizaje Keras presenta unas ventajas importantes que se especifican más adelante.

En la siguiente dirección de internet: <https://www.kdnuggets.com/2019/05/which-deep-learning-framework-growing-fastest.html> podemos encontrar un estudio muy interesante donde a través de varias fuentes de investigación se observa la demanda y el crecimiento de los framework más utilizados por la industria.

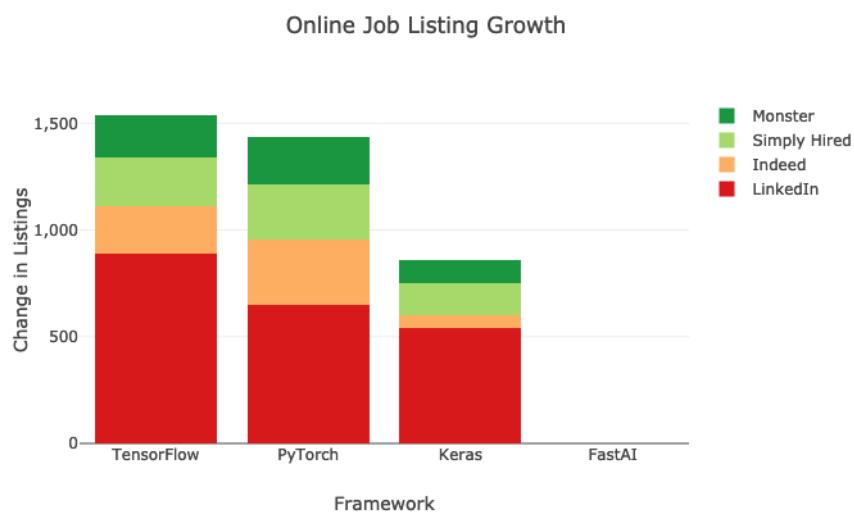
FIGURA Nº 97. RANKING DE LOS FRAMEWORK DE DEEP LEARNING



Fuente: <https://www.kdnuggets.com/2019/05/which-deep-learning-framework-growing-fastest.html>

¿En un estudio titulado “Which Deep Learning Framework is Growing Fastest?”, Jeff Hale refleja los resultados (ver el siguiente gráfico) del número de bibliotecas de aprendizaje profundo que están en demanda en el mercado laboral actual buscando en estas cuatro plataformas: Indeed, LinkedIn, Monster y SimplyHired. El autor buscó el término aprendizaje automático, seguido del nombre de la biblioteca.

FIGURA Nº 96: CRECIMIENTO DE LOS FRAMEWORK DE DEEP LEARNING



Fuente: <https://analyticsindiamag.com/why-tensorflow-is-the-fastest-growing-deep-learning-framework-in-2019>

Nos centraremos y describiremos someramente algunas bibliotecas y programas que están bajo alguna forma de licencia gratuita y que, además, sean robustos. Hemos escogido los siguientes programas informáticos: TensorFlow, Keras, Pytorch, Cafee, Theano, MXNET, CNTK, Deeplearning4j y h2o. También presentaremos la herramienta de Google que dispone en la web y que no requiere ninguna instalación en nuestros ordenadores: Colaboratory Environment (Colab). Esta propuesta de google resulta muy interesante dado que no requiere coste alguno, se puede ejecutar desde cualquier lugar aumentando nuestros recursos a la hora de trabajar con Deep Learning y admite programación de Python y de R.

7.9.2. TensorFlow

TensorFlow es una popular biblioteca de código abierto desarrollada por Google que se utiliza principalmente para la creación y entrenamiento de modelos de aprendizaje automático (machine learning) y redes neuronales. A continuación, se indica una serie de información sobre los usos, aplicaciones y seguimiento de TensorFlow.

Usos y aplicaciones de TensorFlow:

- Aprendizaje automático: TensorFlow es ampliamente utilizado para desarrollar modelos de aprendizaje automático, tanto en tareas de clasificación como de regresión. Permite la construcción de redes neuronales profundas y modelos de aprendizaje profundo para resolver una amplia gama de problemas, como reconocimiento de imágenes, procesamiento de lenguaje natural, traducción automática, reconocimiento de voz, detección de fraudes y análisis de datos, entre otros.
- Investigación y experimentación: TensorFlow es una herramienta valiosa para los investigadores en aprendizaje automático, ya que les permite crear y probar rápidamente nuevos algoritmos y modelos. La flexibilidad de TensorFlow permite implementar fácilmente ideas innovadoras y explorar diferentes enfoques de manera eficiente.
- Desarrollo de aplicaciones móviles y web: TensorFlow proporciona una API llamada TensorFlow Lite, que está optimizada para ejecutar modelos de aprendizaje automático en dispositivos móviles y sistemas integrados. Esto permite desarrollar aplicaciones móviles y web con capacidades de aprendizaje automático, como aplicaciones de reconocimiento de imágenes en tiempo real, asistentes virtuales y recomendaciones personalizadas.
- Procesamiento de datos a gran escala: TensorFlow es escalable y puede aprovechar el poder de cálculo de hardware especializado, como unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU), para acelerar el procesamiento de grandes volúmenes de datos. Esto lo hace adecuado para aplicaciones de procesamiento de datos a gran escala, como análisis de big data, sistemas de recomendación a gran escala y entrenamiento distribuido de modelos.

TensorFlow ha tenido un seguimiento activo y una gran comunidad de desarrolladores desde su lanzamiento inicial en 2015. A lo largo de los años, ha experimentado varias actualizaciones y mejoras, lo que ha llevado a múltiples versiones principales, como TensorFlow 1.x y TensorFlow 2.x. TensorFlow 2.x, lanzado en 2019, introdujo cambios significativos en la API y la usabilidad para hacerla más amigable y accesible. Simplificó la forma en que se construyen y entranan modelos, fomentando un enfoque más intuitivo y declarativo.

Además, TensorFlow ha estado en constante evolución para aprovechar el hardware especializado. Por ejemplo, TensorFlow proporciona soporte nativo para ejecución en GPU y TPU, lo que permite un rendimiento aún más rápido en aplicaciones de aprendizaje automático.

La comunidad de TensorFlow es muy activa y cuenta con una amplia variedad de recursos, como documentación oficial, tutoriales, ejemplos de código y foros de discusión. Esto facilita el aprendizaje de TensorFlow y permite a los desarrolladores obtener ayuda y compartir conocimientos con otros miembros de la comunidad.

7.9.3. Pytorch

PyTorch es otra popular biblioteca de código abierto ampliamente utilizada en el campo del aprendizaje automático y las redes neuronales. A continuación, se indica una serie de información sobre los usos, aplicaciones y seguimiento de PyTorch.

Usos y aplicaciones de PyTorch:

- Aprendizaje automático: PyTorch se utiliza principalmente para el desarrollo de modelos de aprendizaje automático y redes neuronales. Es especialmente conocido por su enfoque basado en tensores dinámicos, lo que significa que los tensores en PyTorch pueden tener una forma y tamaño variables, lo que proporciona una mayor flexibilidad durante el proceso de construcción y entrenamiento de modelos. PyTorch es ampliamente utilizado en tareas como clasificación, detección de objetos, segmentación de imágenes, procesamiento de lenguaje natural y generación de texto e imágenes.
- Investigación: PyTorch ha ganado popularidad en la comunidad de investigación debido a su naturaleza flexible y extensible. Proporciona una interfaz intuitiva para experimentar con nuevos modelos y algoritmos, lo que lo convierte en una elección popular para investigadores en aprendizaje automático. La capacidad de realizar cómputo diferenciable también es una ventaja importante para el desarrollo de nuevos enfoques en el campo.
- Procesamiento de imágenes y visión por computadora: PyTorch ofrece un conjunto de herramientas y bibliotecas especializadas para el procesamiento de imágenes y la visión por computadora. Estas herramientas incluyen módulos para manipulación de imágenes, extracción de características, transformaciones de datos y evaluación de modelos de visión por computadora.

- Procesamiento del lenguaje natural (NLP): PyTorch es ampliamente utilizado en aplicaciones de NLP, como el análisis de sentimientos, la traducción automática, la generación de texto y la respuesta a preguntas. Proporciona una variedad de bibliotecas y modelos pre-entrenados para abordar desafíos comunes en NLP.

Seguimiento de PyTorch: PyTorch ha experimentado un crecimiento significativo en popularidad y adopción desde su lanzamiento inicial en 2016. Ha ganado seguidores debido a su enfoque intuitivo y su facilidad de uso, especialmente en comparación con otras bibliotecas de aprendizaje automático. Además, la comunidad de PyTorch es muy activa y está compuesta por desarrolladores, investigadores y entusiastas que contribuyen al desarrollo y mejora continua de la biblioteca.

PyTorch se actualiza regularmente para agregar nuevas características y mejoras. Además, está respaldado por Facebook AI Research, lo que proporciona un fuerte respaldo y garantiza su desarrollo y mantenimiento a largo plazo.

La comunidad de PyTorch ofrece una amplia gama de recursos, como tutoriales, documentación oficial, ejemplos de código y foros de discusión. Esto facilita el aprendizaje de PyTorch, permite a los desarrolladores obtener ayuda y fomenta la colaboración y el intercambio de conocimientos entre los miembros de la comunidad.

7.9.4. Keras

Otra librería poderosa para Deep Learning también desarrollada para Python es Keras que es una API de redes neuronales de alto nivel, y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollada con un enfoque para permitir la experimentación rápida, es decir que podamos pasar de la idea al resultado con la menor demora posible.

Keras tiene una amplia aceptación en la industria (Netflix, Uber, Yelp, Instacart, Zocdoc, Square y muchas otras.) y en la comunidad científica. Amazon también está trabajando actualmente en el desarrollo de su backend MXNet para Keras.

Keras es una API de redes neuronales y Deep learning de alto nivel que tiene las siguientes características:

- Permite que el mismo código se ejecute en la CPU o en la GPU, sin problemas.

- API fácil de usar con lo que facilita realizar de forma rápida modelos de aprendizaje profundo.
- Compatibilidad incorporada para redes convolucionales (para visión artificial), redes recurrentes (para el procesamiento de secuencias) y cualquier combinación de ambas.
- Admite arquitecturas de red arbitrarias: modelos de múltiples entradas o múltiples salidas, compartición de capas, uso compartido de modelos, etc. Esto significa que Keras es apropiado para construir esencialmente cualquier modelo de aprendizaje profundo, desde una red de memoria a una máquina de Turing neuronal.
- Keras es compatible con varios motores de back-end y no se limita a un solo ecosistema. Los modelos Keras se pueden desarrollar con una gama de diferentes backends de aprendizaje profundo: El backend TensorFlow de Google, el backend CNTK de Microsoft o el backend theano.
- Trabaja con tarjetas GPU de NVIDIA o GPU habilitadas para OpenCL, como las de AMD, a través del backendPlaidMLKeras. Keras dispone de un sólido soporte multi-GPU y soporte de entrenamiento distribuido. También dispone de soporte incorporado para el paralelismo de datos multi-GPU Horovod, de Uber.
- También trabaja con Google TPUs, a través del backendTensorFlow y Google Cloud.

7.9.5. MXNET

MXNet es otra biblioteca de aprendizaje automático de código abierto que se ha vuelto popular en los últimos años. Fue desarrollada por Amazon y se ha convertido en una alternativa sólida a TensorFlow y PyTorch. MXNet está diseñado para ofrecer un alto rendimiento y escalabilidad en una amplia variedad de plataformas, incluidas CPU, GPU y múltiples dispositivos distribuidos.

Uno de los puntos destacados de MXNet es su ecosistema Gluon, que proporciona una interfaz de programación más simple y fácil de usar en comparación con otras bibliotecas. Gluon permite a los desarrolladores construir y entrenar modelos de manera más intuitiva y eficiente, abstrayendo gran parte de la complejidad técnica.

El ecosistema Gluon incluye varias bibliotecas especializadas:

- GluonTS: es una biblioteca de series temporales basada en MXNet. Proporciona herramientas y modelos para el análisis y pronóstico de series temporales, lo que incluye

la capacidad de construir y entrenar modelos para predecir valores futuros en series de tiempo. GluonTS facilita la implementación de modelos avanzados para problemas de series temporales, como la predicción de ventas, la demanda de energía y la predicción de precios.

- GluonCV: es una biblioteca que se centra en la visión por computadora y el procesamiento de imágenes. Ofrece una amplia gama de modelos pre-entrenados para tareas comunes de visión por computadora, como clasificación de imágenes, detección de objetos y segmentación semántica. GluonCV también proporciona herramientas para la manipulación y procesamiento de imágenes, lo que facilita la creación de aplicaciones de visión por computadora utilizando MXNet.
1. GluonNLP: es una biblioteca dedicada al procesamiento del lenguaje natural (NLP). Proporciona una interfaz intuitiva y herramientas para tareas de NLP, como el procesamiento de texto, la clasificación de sentimientos, la traducción automática y la generación de texto. GluonNLP ofrece una variedad de modelos pre-entrenados y herramientas para facilitar el desarrollo de aplicaciones de NLP utilizando MXNet.

El ecosistema Gluon en MXNet ha sido bien recibido por los desarrolladores debido a su facilidad de uso y flexibilidad. Gluon simplifica la construcción y entrenamiento de modelos, lo que acelera el proceso de desarrollo y experimentación.

Además del ecosistema Gluon, MXNet en sí ofrece características avanzadas como el cómputo diferenciable, lo que permite la implementación de técnicas de aprendizaje automático como el aprendizaje profundo y el aprendizaje por refuerzo. También tiene soporte nativo para ejecución en GPU y permite el entrenamiento distribuido para aprovechar el poder de cómputo de múltiples dispositivos.

7.9.6. CNTK

Computer Network Toolkit (CNTK) es una biblioteca de código abierto que contiene todos los componentes básicos para la construcción de redes neuronales y que Microsoft ha puesto a disposición general a través de Github.

La biblioteca CNTK principal está construida en Python sobre un núcleo C++. Aunque también está disponible en C# y Java. Existe otra forma de trabajar con este programa utilizando su propio lenguaje de programación llamado BrainScript.

El equipo de Microsoft Research afirma que a través de un conjunto de redes de computadoras replica las redes neuronales de un cerebro humano, obteniendo así resultados de procesamiento

de información mucho más rápidos que otras máquinas de aprendizaje, como Tensorflow. Aseveran que son especialistas en el área de la Fonética, lo que indica que “*CNTK se ha convertido en algo muy eficiente y podría utilizarse fácilmente en proyectos de pequeña escala o en proyecto con una escabilidad alta*”.

Una característica importante es que CNTK combina una API de bajo nivel con otra de alto nivel para construir redes neuronales. La API de bajo nivel está destinada a científicos e investigadores que buscan construir la próxima generación de componentes de redes neuronales, mientras que la API de alto nivel está orientada a construir redes neuronales de calidad.

Además de estos componentes básicos, CNTK presenta un conjunto de elementos que facilitarán la alimentación de datos en su red neuronal. También contiene varios componentes para monitorear y depurar redes neuronales.

CNTK es compatible con los sistemas operativos Linux de 64 bits o Windows de 64 bits. Para instalar, puede elegir paquetes binarios precompilados o compilar el kit de herramientas de la fuente proporcionada en GitHub.

CNTK es también uno de los primeros juegos de herramientas de aprendizaje profundo que admite el formato ONNX de intercambio de redes neuronales abiertas, una representación de modelo compartido de código abierto para la interoperabilidad de marcos y la optimización compartida. Desarrollado conjuntamente por Microsoft y respaldado por muchos otros, ONNX permite a los desarrolladores mover modelos entre marcos como CNTK, MXNet y PyTorch.

7.9.7. DeepLearning4j

Deeplearning4j es un software de código abierto publicado bajo Apache License 2.0, desarrollado principalmente por un grupo de aprendizaje automático con sede en San Francisco y Tokio y dirigido por Adam Gibson. Este programa se puede instalar en WEKA.

Deeplearning4j es un kit de herramientas basado en Java para construir, entrenar y desplegar redes neuronales profundas. También realiza regresiones y KNN.

Deeplearning4j tiene los siguientes subproyectos:

- ✓ **DataVec** Gestiona datos, así como la normalización y la transformación de los mismos
- Deeplearning4j Proporciona Herramientas para configurar redes neuronales y generar gráficos.

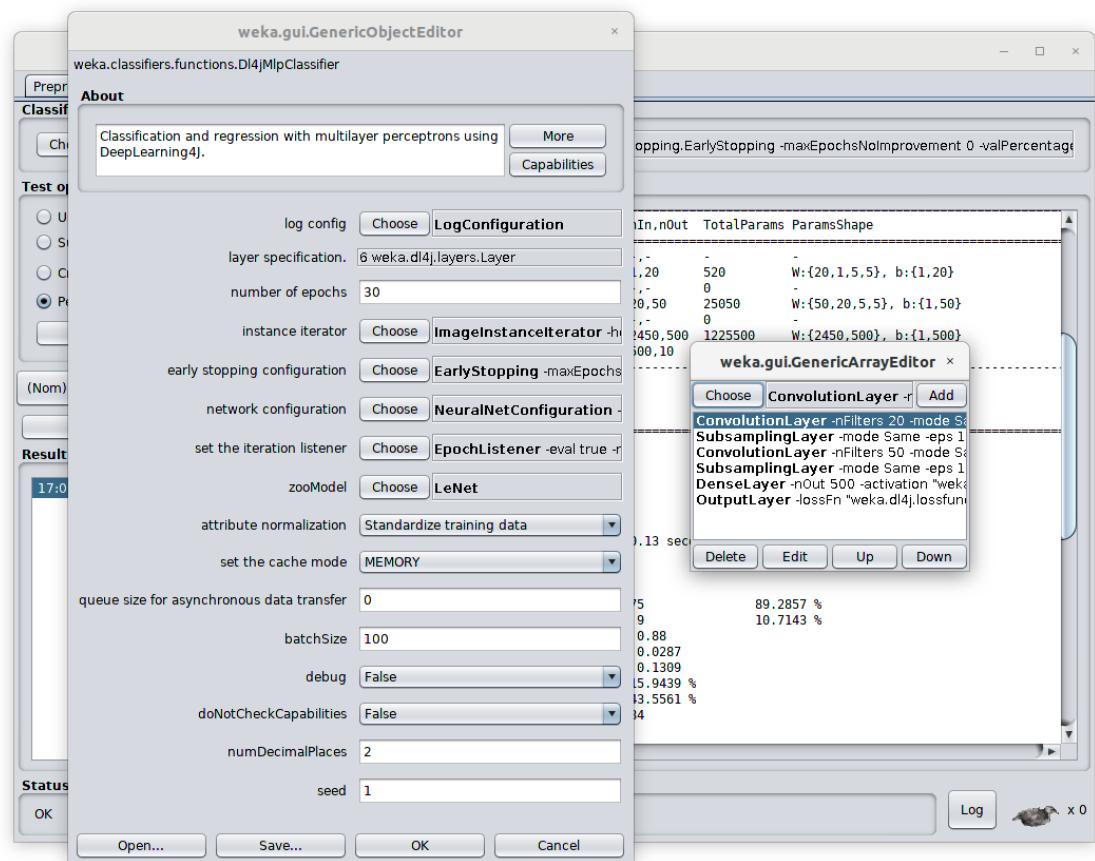
- ✓ **DL4J-Examples** contiene ejemplos de trabajo para clasificación y agrupamiento de imágenes, series de tiempo y texto.
- ✓ **Keras Model Import** ayuda a importar modelos entrenados de Python y Keras a DeepLearning4J y Java.
- ✓ **ND4J** permite a Java acceder a Bibliotecas nativas para procesar rápidamente datos de matriz en CPU o GPU.
- ✓ **ScalNet** es un contenedor de Scala para Deeplearning4j inspirado en Keras. Se ejecuta en multi-GPU con Spark.
- ✓ **RL4J** implementa Deep Q Learning, A3C y otros algoritmos de aprendizaje de refuerzo para la JVM.
- ✓ **Arbiter** ayuda a buscar en el espacio de los hiperparámetros para encontrar la mejor configuración de red neuronal.

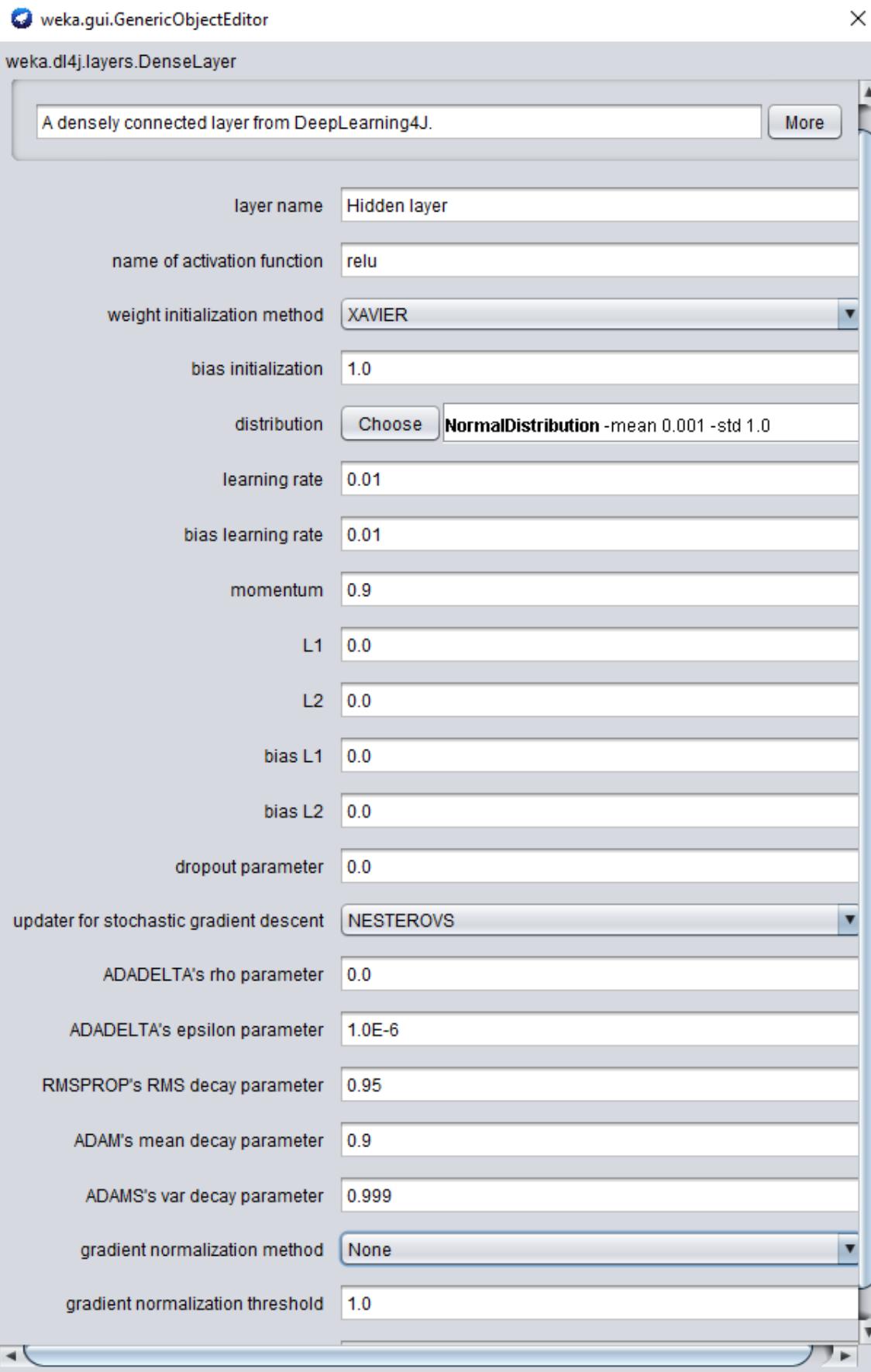
El software WEKA ha incorporado este programa. Para poder activarlo tienes que instalarlo desde la barra de menús Tools/Packages Manager. Puedes ampliar la información desde la siguiente dirección de internet: <https://deeplearning.cms.waikato.ac.nz/>

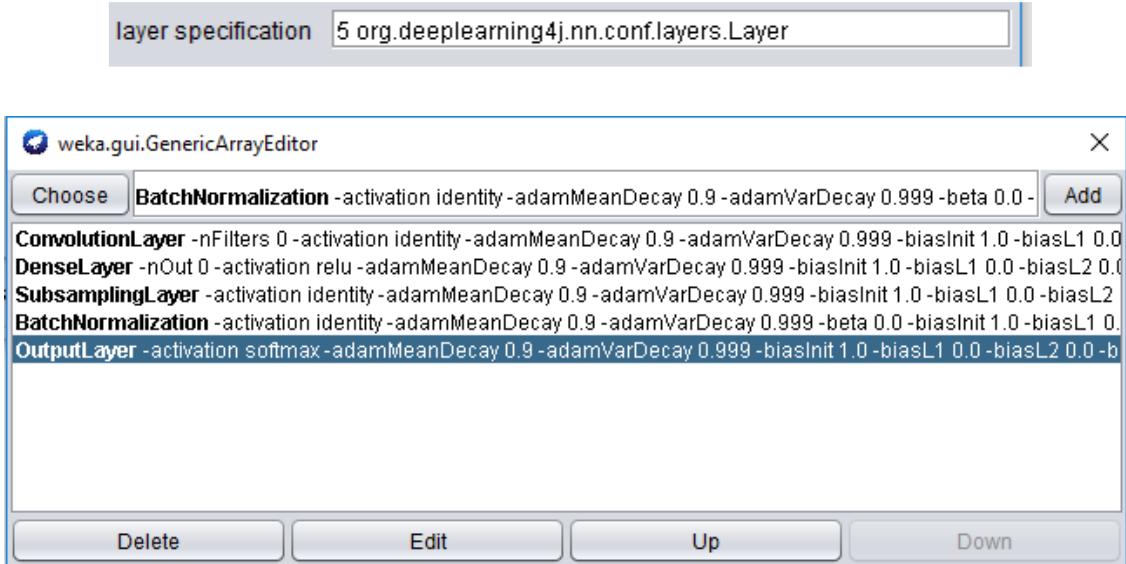
En Weka disponemos de las siguientes capas de red neuronal para construir sofisticadas arquitecturas:

- ✓ **ConvolutionLayer**: para aplicar a problemas relacionados con el tratamiento de imágenes y texto.
- ✓ **DenseLayer**: todas las unidades están conectadas a todas las unidades de su capa principal
- ✓ **SubsamplingLayer**: submuestra de grupos de unidades de la capa primaria por diferentes estrategias (promedio, máximo, etc.)
- ✓ **Normalización de lote**: aplica la normalización de lote común. Estrategia sobre las activaciones de la capa primaria
- ✓ **LSTM**: utiliza un enfoque de memoria a corto y largo plazo
- ✓ **GlobalPoolingLayer**: aplique la agrupación a lo largo del tiempo para RNN y la agrupación para CNN aplicados en secuencias. OutputLayer: genera resultados de clasificación / regresión.

En las imágenes que se presentan a continuación se observa la forma de configurar modelos de Deep learning y algunas capacidades que facilita Weka a través de menús a la hora de implementar arquitecturas de Deep Learning:







7.9.8. H2O

La empresa H2O.ai se centra en llevar la inteligencia artificial a otras compañías e instituciones a través de su producto insignia H2O. Es una plataforma abierta que incluye interfaces para R, Python, Scala, Java, JSON y CoffeeScript / JavaScript, así como una interfaz web incorporada denominada “Fluir”. H2O está diseñada para funcionar independientemente en Hadoop, o dentro de un cluster de Spark

El aprendizaje profundo de H2O se basa en una red neuronal artificial de alimentación de múltiples capas que se entrena con el descenso de gradiente estocástico mediante propagación hacia atrás. La red puede contener una gran cantidad de capas ocultas que consisten en neuronas con diferentes funciones de activación de tanh, rectificador y maxout. Las características avanzadas, como la tasa de aprendizaje adaptativo, la tasa de aprendizaje, el momento, la parada temprana, la regularización L1 o L2 o la búsqueda a través de una cuadrícula permiten una alta precisión predictiva.

Otro proyecto de la compañía es Deep Water que desde diciembre de 2016 es de código abierto. Los usuarios de este software pueden acceder fácilmente a TensorFlow, MXNet y Caffe con interfaces familiares: H2O Flow, R, Python, Spark, Scala, Java o la API REST, incluido modelos de despliegue a través de MOJO y H2O Steam. En H2O Flow, el usuario puede cambiar fácilmente entre los backends desde un menú desplegable.

A continuación, se presenta el código general de un prototipo de autoencoder en Deep Water.

```
h2o.deepwater(x, y, training_frame, model_id= NULL, checkpoint= NULL, autoencoder= FALSE, validation_frame= NULL, nfolds= 0, balance_classes= FALSE, max_after_balance_size= 5, class_sampling_factors= NULL, keep_cross_validation_predictions= FALSE, keep_cross_validation_fold_assignment= FALSE, fold_assignment= c("AUTO", "Random", "Modulo", "Stratified"), fold_column= NULL, offset_column= NULL, weights_column= NULL, score_each_iteration= FALSE, categorical_encoding= c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"), overwrite_with_best_model= TRUE, epochs= 10, train_samples_per_iteration= -2, target_ratio_comm_to_comp= 0.05, seed= -1, standardize= TRUE, learning_rate= 0.001, learning_rate_annealing= 1e-06, momentum_start= 0.9, momentum_ramp= 10000, momentum_stable= 0.9, distribution= c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"), score_interval= 5, score_training_samples= 10000, score_validation_samples= 0, score_duty_cycle= 0.1, classification_stop= 0, regression_stop= 0, stopping_rounds= 5, stopping_metric= c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error"), stopping_tolerance= 0, max_runtime_secs= 0, ignore_const_cols= TRUE, shuffle_training_data= TRUE, mini_batch_size= 32, clip_gradient= 10, network= c("auto", "user", "lenet", "alexnet", "vgg", "googlenet", "inception_bn", "resnet"), backend= c("mxnet", "caffe", "tensorflow"), image_shape= c(0, 0), channels= 3, sparse= FALSE, gpu= TRUE, device_id= c(0), cache_data= TRUE, network_definition_file= NULL, network_parameters_file= NULL, mean_image_file= NULL, export_native_parameters_prefix= NULL, activation= c("Rectifier", "Tanh"), hidden= NULL, input_dropout_ratio= 0, hidden_dropout_ratios= NULL, problem_type= c("auto", "image", "dataset"))
```

7.9.9. Google Colaboratory para Python y R

El entorno Colab (Google Colaboratory) es una potente herramienta de google para ejecutar código incluido el deep Dearning y que está disponible en la web (<https://colab.research.google.com/>). Se ha desarrollado para Python, pero actualmente también se puede ejecutar código de R. Esta funcionalidad puede importar un conjunto de datos de imágenes, entrenar un clasificador con este conjunto de datos y evaluar el modelo con tan solo usar unas pocas líneas de código. Los cuadernos de Colab ejecutan código en los servidores en la nube de Google, lo que nos permite aprovechar la potencia del hardware de Google, incluidas las GPU y TPU, independientemente de la potencia de tu equipo. Lo único que se necesita es un navegador.

Con Colab se puede aprovechar toda la potencia de las bibliotecas más populares de Python para analizar y visualizar datos. La celda de código de abajo utiliza NumPy para generar datos aleatorios y Matplotlib para visualizarlos. Para editar el código, solo se tiene que hacer clic en la celda.

The screenshot shows the Google Colab interface. At the top, there's a navigation bar with the title "Copia de Copia de ejemplo_serie_temporal_colab.ipynb" and icons for star, file, edit, view, insert, execution environment, tools, help, and last modified. Below the title, there are buttons for "+ Código" and "+ Texto". A sidebar on the left has icons for file, code, and folder. The main area contains a code cell with the following Python code:

```
[ ] import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
```

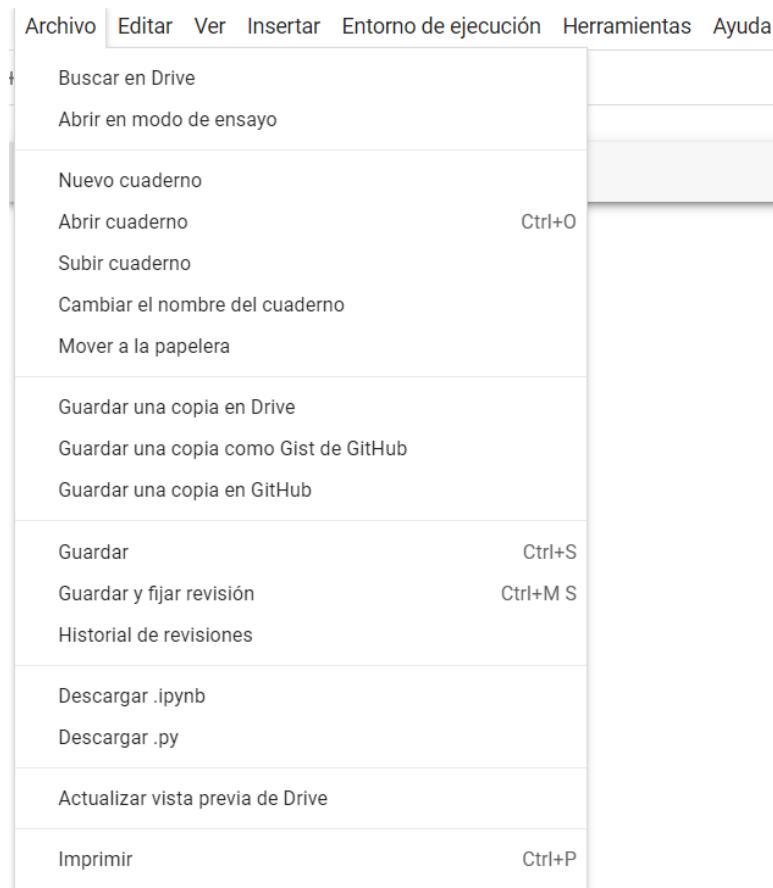
Below the code cell, there's a note: "Using TensorFlow backend." followed by a play button icon. Another code cell below it contains the following Python code:

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

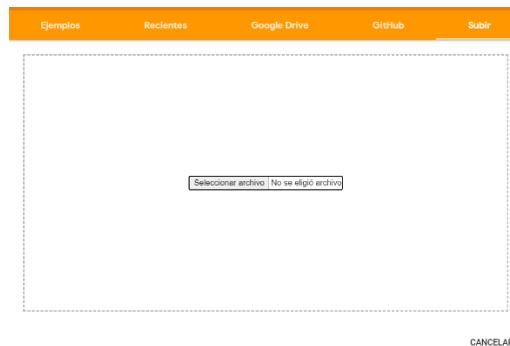
Este es el menú principal de colab desde donde podemos gestionar nuestros proyectos

The screenshot shows the Google Colab interface. At the top, there's a navigation bar with the title "Untitled15.ipynb" and icons for star, file, edit, view, insert, execution environment, tools, help, and last modified. Below the title, there are buttons for "+ Código" and "+ Texto". A sidebar on the left has icons for file, code, and folder. The main area contains a code cell with a play button icon and a vertical line indicating where the code starts.

Desde el Menú archivo, como en la mayor parte de los programas, podemos llevar a cabo las operaciones habituales de abrir y guardar los ficheros en diferentes formatos. En este caso se pueden abrir ficheros de Jupyter/Python desde cualquier dispositivo externo, desde el repositorio Drive o de Github:



Si queremos subir un fichero que tenemos en nuestro ordenador vamos a Archivo/ Subir cuaderno y podemos elegir nuestro archivo cuando se despliegue la siguiente pantalla:



Normalmente, cuando ya se ha trabajado y se han guardado los proyectos en colab se inicia la sesión abriendose la pantalla que se muestra a continuación:

The screenshot shows the 'Recientes' tab selected in the top navigation bar. Below it is a table listing five notebooks:

Título	Abierto por primera vez	Abierto por última vez	Actions
Untitled2.ipynb	hace 1 minuto	hace 1 minuto	
primera_prueba_colab.ipynb	hace 13 horas	hace 58 minutos	
Te damos la bienvenida a Colaboratory	hace 9 días	hace 59 minutos	
Untitled1.ipynb	hace 14 horas	hace 14 horas	
Markdown Guide	hace 9 días	hace 9 días	

CANCELAR

Como se ha comentado también se pueden importar archivos desde GitHub introduciendo la url de GitHub:

The screenshot shows the 'GitHub' tab selected in the top navigation bar. A search bar at the top says 'Introduce una URL de GitHub o busca por organización o usuario'. Below it, there are dropdown menus for 'Repositorio:' set to 'BeltranPascual' and 'Rama:' set to 'master'. To the right is a checked checkbox 'Incluir repositorios privados' and a magnifying glass search icon.

The main area lists four files from the repository:

- 04_Redes_neuronales_densamente_conectadas_(1).ipynb
- 05_redes_neuronales_en_keras.ipynb
- [08_redes_neuronales_convolucionales_\(2\).ipynb](#)
- 09_etapas_de_un_proyecto_deep_learning.ipynb

CANCELAR

Vamos a desarrollar un programa de redes neuronales, con los datos de tráfico de pasajeros ajustando la serie temporal con una red neuronal con el programa keras. Para subir un fichero a colab se tiene que importar el método files de la siguiente manera: from google.colab import files y después poner el siguiente comando: files.upload() lo que nos habilitará poder elegir y subir el fichero desde nuestro ordenador. El fichero se coloca en el directorio raíz de Colab “/content”. Para habilitarlo lo importaremos ya con los comandos de pandas o con los comandos de R.

```
[ ] from google.colab import files  
files.upload()  
  
[ ] dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='pyth  
  
[ ] dataframe.head()  
  
[ ] International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60  
0 112  
1 118  
2 132  
3 129  
4 121  
  
[ ] # fix random seed for reproducibility  
numpy.random.seed(7)  
dataset = dataframe.values  
dataset = dataset.astype('float32')  
  
[ ]  
  
[ ] # split into train and test sets  
train_size = int(len(dataset) * 0.67)  
test_size = len(dataset) - train_size  
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]  
  
[ ] # reshape dataset  
look_back = 3  
trainX, trainY = create_dataset(train, look_back)  
testX, testY = create_dataset(test, look_back)
```

```
[ ] # create and fit Multilayer Perceptron model
model = Sequential()
model.add(Dense(12, input_dim=look_back, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=400, batch_size=2, verbose=2)
```

↳ Epoch 1/400
- 0s - loss: 56022.6876
Epoch 2/400
- 0s - loss: 35496.0284
Epoch 3/400
- 0s - loss: 22916.6523
Epoch 4/400
- 0s - loss: 12226.6756
Epoch 5/400
- 0s - loss: 5351.1794
Epoch 6/400
- 0s - loss: 2155.4325
Epoch 7/400

```
[ ] # Estimate model performance
trainScore = model.evaluate(trainX, trainY, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore, math.sqrt(trainScore)))
testScore = model.evaluate(testX, testY, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore, math.sqrt(testScore)))
```

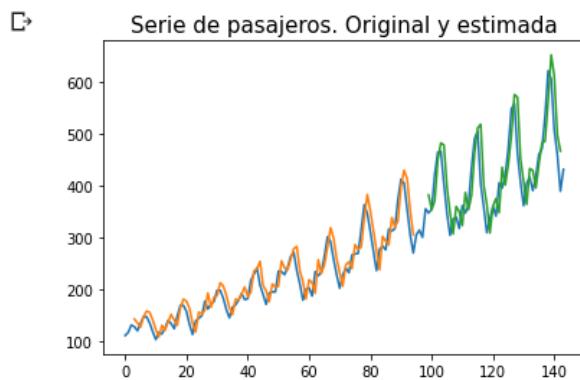
↳ Train Score: 564.73 MSE (23.76 RMSE)
Test Score: 2244.72 MSE (47.38 RMSE)

```
[ ] # generate predictions for training
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

```
[ ] # shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
```

```
[ ] # plot baseline and predictions
plt.plot(dataset)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.title('Serie de pasajeros. Original y estimada', fontsize=15)
plt.show()
```



Código de R en Colab

Si queremos utilizar código en R necesitaremos lanzar la creación del notebook con la siguiente URL: <https://colab.research.google.com/notebook#create=true&language=r>

Con esta UR se le dice a Colab que el lenguaje a usar es R.

Luego, como en cualquier entorno necesitaremos instalar las librerías que vayamos a usar, en este caso se importan, a modo de prueba, las librerías caret, keras, tensorflow y ggplot2 y se realiza un ejemplo sencillo de regresión lineal con caret. Abrimos el fichero Ejemplo_colab_R.ipynb.

```
[ ] Instalamos e importamos diferentes librerías
[ ] install.packages('keras')
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'rappdirs', 'config', 'reticulate',
[ ] library(tensorflow)
```

```
<> [ ] library(caret)
     └─ Loading required package: lattice
         Loading required package: ggplot2
         Attaching package: 'caret'
         The following object is masked from 'package:tensorflow':
             train
```

```
[ ] install.packages('mlbench')
     └─ Installing package into '/usr/local/lib/R/site-library'
         (as 'lib' is unspecified)
```

```
[ ] library(mlbench)
[ ] library(ggplot2)
```

```
# Importamos la base de datos Boston Housing
data(BostonHousing)

head(BostonHousing)

sum(is.na(BostonHousing))

set.seed(100)

TrainingIndex <- createDataPartition(BostonHousing$medv, p=0.8, list = FALSE)
TrainingSet <- BostonHousing[TrainingIndex,] # Training Set
TestingSet <- BostonHousing[-TrainingIndex,] # Test Set

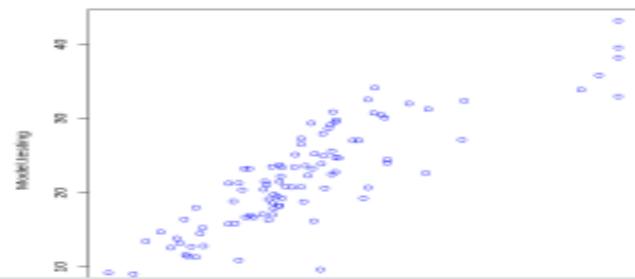
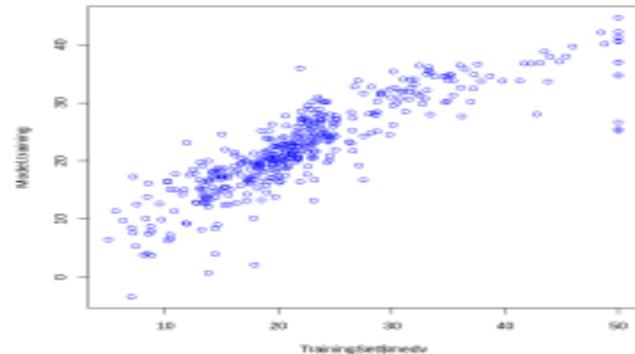
# Construimos el modelo
Model <- train(medv ~ ., data = TrainingSet,
                method = "lm",
                na.action = na.omit,
                preProcess=c("scale","center"),
                trControl= trainControl(method="none")
            )
```

```

# Realizamos predicciones
Model.training <- predict(Model, TrainingSet)
Model.testing <- predict(Model, TestingSet)
# Realizamos un gráfico
plot(TrainingSet$medv,Model.training, col = "blue" )
plot(TestingSet$medv,Model.testing, col = "blue" )

A data.frame: 6 x 14
  orim   zn  indus chas nox    rm   age   dis    rad   tax ptratio    b   lstat medv
<dbl> <dbl>
1 0.00632 18  2.31  0  0.538 6.575 65.2  4.0900 1  296 15.3  396.90 4.98  24.0
2 0.02731 0   7.07  0  0.469 6.421 78.9  4.9671 2  242 17.8  396.90 9.14  21.6
3 0.02729 0   7.07  0  0.469 7.185 61.1  4.9671 2  242 17.8  392.83 4.03  34.7
4 0.03237 0   2.18  0  0.458 6.998 45.8  6.0622 3  222 18.7  394.63 2.94  33.4
5 0.06905 0   2.18  0  0.458 7.147 54.2  6.0622 3  222 18.7  396.90 5.33  36.2
6 0.02985 0   2.18  0  0.458 6.430 58.7  6.0622 3  222 18.7  394.12 5.21  28.7

```



7.9.10. Conclusiones

El campo de la investigación del Deep Learning y el desarrollo sobre la inteligencia artificial, en general, propone nuevos retos rápidos con el discurrir del tiempo por lo que hay que adaptarse a las nuevas necesidades. En el campo tan fecundo de las herramientas informáticas cada vez aparecen nuevos diseños y también algunas de ellas se quedan rápidamente obsoletas.

Es por ello, que en este pequeño epígrafe sólo hemos recogido unos cuantos marcos de Deep Learning que nos han parecido interesantes y que esperamos pueden servir de ayuda a los estudiantes de este máster, siendo conscientes de que existen muchas otras propuestas también interesantes como Gluon, Chainer, Deep Learning Studio o Fastai que pretenden combinar las mejores prestaciones de Keras y Pytorch.

8. Bibliografía

Acid, S. Campos, L.M., y J.G. Castellano, J.G. (2005). Learning Bayesian network classifiers: Searching in a space of partially directed acyclic graphs. *Mach. Learn.*, 59:213–235.

Akaike, H. (1974). A new look at the statistical model identification. *Trans. Autom. Control* 19:716-723.

Andrew, N.G. (2011): Sparse Autoencoder. S294A Lecture notes, 72.

Baldi, P. y Soren, B. (2001). Bioinformatics: The machine learning approach. 2nd Ed.,

Bengio, Y. (2009): Learning deep architectures for AI. Foundations and trends® in Machine Learning, vol. 2, no 1, p. 37.

Bengio, Y. y LeCun, Y. (2007): "Scaling learning algorithms towards AI," in Large Scale Kernel

Bengio, Y. , Lamblin, P., Popovici, D y Larochelle,H. (2007): Greedy layer-wise training of deep networks. Advances in neural information processing systems, 2007, vol. 19, p. 153.

Bengio, Y. and Pascal Lamblin, Dan Popovici, Hugo Larochelle

Bouckaert, G. (1995) 'Remodeling Quality and Quantity in a Management Context', in A. Halachmi and Geert Bouckaert (eds) Public Productivity Through Quality and Strategic Management.

Bourlard, H., and Y. Kamp. 1988. "Auto-Association by Multilayer Perceptrons and Singular Value

Charte, David, Francisco Charte, Salvador García, María J. del Jesus, and Francisco Herrera. 2018. "A Practical Tutorial on Autoencoders for Nonlinear Feature Fusion: Taxonomy, Models Software and Guidelines."

Decomposition." Massachusetts Institute of Technology: MIT Press: 365-369.Battiti (1996).

Diederik P. Kingma, Jimmy Lei Ba. 2017. "ADAM: A Method for Stochastic Optimization."

Doersch, Carl. 2016. "Tutorial on Variational Autoencoders."

Campos, L.M. (2006). A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research* 7:2 pp. 149-2187.

Castillo, E., Gutiérrez, J.M. y Hadi, A. (1998). *Sistemas Expertos y modelos de redes Probabilísticas*. Monografías de la Academia de Ingeniería.

Cerny, V. (1985). Thermodinamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41-45.

Cerquides, J. y López, R. (2005). Robust bayesian linear classifier ensembles. In *Machine Learning: ECML 2005*, pages 72–83. Springer.

Cheng, J., Bell, D.A. y Liu, W. (1997). Learning belief networks from data: an information theory based approach. *Proceedings of the sixth international conference on Information and knowledge management*. pp 325 – 331.

Chow, K., Liu, C.N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467.

Cooper, G.F. (1990). The computational complexity of probabilistic inference using Bayesian belief network. *Artificial Intelligence* nº 42, 393-405. Cooper y Herskovitz, (1992).

Cowell, R.G., David, A.P., Lauritzen, S.L. y Spiegelhalter. D.J. (1999). *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York.

Cowell, R. (2001). On searching for optimal classifiers among Bayesian networks. T. Jaakkola and T. Richardson, editors, *Proceedings of the 8th International Conference on Artificial Intelligence and Statistics*, pages 175-180.

De Campos, L.M. (2006). A scoring function for learning bayesian networks based on mutual information and conditional independence tests. *The Journal of Machine Learning Research*, 7:2149–2187.

Dietterich, T.G. (2000). Ensemble methods in machine learning. *Multiple classifier systems*, pages 1–15. Springer Duda y Hart (1973).

Dietterich,T.G., Ashenfelter,T.D.A., Bulatov, Y. (2004). "Training conditional random fields via gradient treeboosting," in Proceedings of the 21st International Conference on Machine Learning (ICML), (Banff, AB).

Edwards, W. (1998). Hailfinder. Tools for and experiences with bayesian normative modeling. American Psychologist, 53, 416-428.

El-Hay, T. (2001). Efficient Methods for exact and approximate inference in discrete Graphicals Models. Master of Science Thesis, Supervisor Nir Friedman: 17-18.

Ferreira, J.T.A.S., Denison, D.G.T. y Hand, D.J. (2001). Weighted naive Bayes modelling for data mining. Technical report. Department of Mathematics, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK.

Fischer, A., Igel, C. (2012): An introduction to restricted Boltzmann machines. En Progress in Pattern Recognition, Image Analysis, Computer Vision and Applications. Springer Berlin Heidelberg, 2012. p. 14-36.

Flores, M.J., Gámez J. A., Martínez A.M., Puerta J.M. (2009). GAODE and HAODE: two proposals based on AODE to deal with continuous variables, in A. P. Danyluk, L.

Friedman, J.H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics 29(5):1189-1232.

Friedman, J.H. (2002). "Stochastic Gradient Boosting," Computational Statistics and Data Analysis 38(4):367-378.

Friedman, J.H., Hastie, T., Tibshirani, R. (2000). "Additive Logistic Regression: a Statistical View of Boosting," Annals of Statistics 28(2):337-374.

Friedman, N., Getoor, L., Koller, D. y Pfeffer, A. (1999). Learning probabilistic relational models, Proceedings of the Sixteenth International Joint Conferences on artificial Intelligence, pp.1300-1309.

Friedman, N. y Goldszmidt, M. (1996). Building classifiers using Bayesian networks. In Proceedings of the 13th National Conference on Artificial Intelligence, pp 1277-1284, 1996.

Gama, J. (2000). A cost-sensitive iterative Bayes. Proceedings of Workshop on Cost-Sensitive Learning at the 17th International Conference on Machine Learning, Stanford University, June 2000.

Garbolino, P. y Taroni, F. (2002). Evaluation of scientific evidence using bayesian networks. Forensic Science International, 125, pp. 149-155.

Geiger, D., Heckerman, D. (1994). Learning Gaussian networks. In Proceedings of Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, WA, pp 235-243. Morgan Kaufmann.

Geiger, D., Heckerman, D. (1995). Likelihoods and Parameter Priors for Bayesian Networks.

Gers, F, Felix A. Jürgen Schmidhuber, Fred A. Cummins (2000) Learning to Forget: Continual Prediction with LSTM

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the Difficulty of Training Deep Feedforward Neural Networks."

Glover, F. (1986). Future paths for integer programming and link to artificial Intelligence. Comput. Oper. Res., 13(5):533-549.

Glover, F. y Laguna, M. (1997). Tabu Search. Kluver Academic Publishers, Norwell, MA, USA.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Graff, P., Feroz, F., Hobson, M.P., Lanseby, A. Skynet (2014): An efficient and robust neural network training tool for machine learning in astronomy. Monthly Notices of the Royal Astronomical Society 441. p. 1741-1759.

Greiner, R. y Zhou, W. (2002). Structural extensión to logistic regression: discriminant parameter learning of belief net classifiers. Proceedings of the 18th National Conference on Artificial Intelligence, pages 167-173.

Grossman, D. y Domingos, P. (2004). Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. 21st International Conference on Machine Learning.

Gu, Xiuxiang, Zhenhua Wang, Jason Kuen, Liyang Ma, Amir Shahroudy, Bing Shuai, TingLiu, et

Heckerman, D. (1991). Probabilistic similarity nets dissertation award series. MIT Press.

Heckerman, D. (1996). A tutorial on learning with Bayesian networks. Tech. Rep. Nº. MSR-TR-95-06. Redmond, WA: Microsoft Research.

Heckerman, D., Geiger, D. y Chickering, D.M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In R. Lopez de Mantaras and D. Poole, editors, Tenth Conference on Uncertainty in Artificial Intelligence, pages 293–301. Morgan-Kaufmann.

Hernández, O. (2004) Introducción a la minería de datos. Prentice Hall.

Hinton, G.E., Osindero, S., y Teh, Y. (2006): "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, pp. 1527-1554.

Hinton, G.E. y Salakhutdinov, R. (2006): "Reducing the dimensionality of data with neural networks," Science, vol. 313, pp. 504-507.

Hinton, G.E. y Sejnowski, T.J. (1986): "Learning and relearning in Boltzmann machines," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, (D. E. Rumelhart and J. L. McClelland, eds.), pp. 282{317, Cambridge, MA: MIT Press.

Hinton, G.E. y Zemel, R.S. (1994): "Autoencoders, minimum description length, and Helmholtz free energy," in Advances in Neural Information Processing Systems 6 (NIPS'93), (D. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 3{10, Morgan Kaufmann Publishers, Inc.

Hoeting, J.A., Madigan, D., Raftery, A.E. y Volinsky, C.T. (1999). Bayesian model averaging: A tutorial. Statistical science, pages 382–401.

Inza, P., Larrañaga, R., Etxeberria y Sierra, B. (2000). Feature subset selection by Bayesian network-based optimization. Artificial Intelligence, 123:157-184.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift."

Jensen, F.V. (2001). Bayesian Networks and Decision Graphs. Springer-Verlag.

J Jensen, F.V. y Nielsen, T.D. (2007). Bayesian Networks and Decisions Graphs. Information Science and Statistics. Series, Springer Verlag, New York segunda edición: 294 páginas.

Jiang, L. y Zhang, H. (2006). Weightily averaged one-dependence estimators. PRICAI 2006: trends in artificial intelligence, pages 970–974. Springer.

Jiuxiang. G. et al. (2019) Recent Advances in Convolutional Neural Networks.

Kadie, C.M., Hovel, D. y Hovitz, E. (2001). A component-centric toolkit for modeling and inference with bayesian networks. (Tech. Rep. MSR-TR-2001-67). Redmond, WA: Microsoft Corporation. 2008, Vol. 13 nº 1, pp. 13-25.

Kaiming H. et al. (2016) Aprendizaje profundo residual para el reconocimiento de la imagen.

Karpathy. A. (2015) "The unreasonable effectiveness of recurrent neural networks"

Krizhevsky. A. et al. (2012) Clasificación de ImageNet con redes neuronales convolucionales profundas "

Junyoung Chung et al. (2014) titulado "Evaluación empírica de redes neuronales recurrentes cerradas en modelo secuencial ".

Keogh, E.J. y Pazzani. M. (1999). Learning augmented Bayesian classifiers: a comparison of distribution-based and non distribution-based approaches. Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics, pp. 225-230.

Keogh, E.J. y Pazzani, M.J. (2002). Learning the structure of augmented Bayesian classifiers. International Journal on Artificial Intelligence Tools, 11(04):587–601.

Kirkpatrick, S., Gelatt, C.D. y Vecchi, M.P. (1983). Optimization by simulated annealing, Science, 220(4598):671-680.

Kjærulff, U.B. y Madsen, A.L. (2008). Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis. Springer Verlag, Series: Information Science and Statistics, New York XVIII 318 páginas.

Kohavi, R. (1996). Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pages 202-207.

Kohavi, R. y John, G.H. (1997). Wrappers for feature subset selection. Artificial Intelligence, 97(1-2) :273-324, 1997.

Lam, W. y Bacchus, F. (1994). Learning Bayesian belief networks: an approach based on the MDL principle. Comp. Intelligence, 10: pp. 269-293.

Langseth, H. y Nielsen, T.D. (2002). Classification using hierarchical naïve-Bayes models. URL citeseer.ist.psu.edu/langseth02classification.html.

Langseth, H. y Nielsen, T.D. (2006). Classification using hierarchical Naïve Bayes models. Machine Learning. 63, 2.

Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R.H. y Kuijpers, C.M.H. (1996). Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters. Pattern Analysis and Machine Intelligence, IEEE Transactions on Sep 1996. Volume: 18, Issue: 9. Page(s): pp. 912 – 926.

Liu, J.N.K., Li, B.N.L. y Dillon, T.S. (2001). An improved naive Bayesian classifier technique coupled with a novel input solution method. IEEE Transactions on Systems, Man and Cybemetics - Part C: Applications and Reviews, 31(2):249- 256.

Lopex, N., Ribeiro, B., Gonçalves, J. (2012): Restricted Boltzmann machines and deep belief networks on multi-core processors. En Neural Networks (IJCNN), The 2012 International Joint Conference on. IEEE, 2012. p. 8.

López, J., García, J. y De la Fuente, L. (2006). Modelado causal con redes bayesianas. Actas de las XXVII Jornadas de Automática, 198–202.

Madsen, A.L. y Jensen, F.V. (1999). Lazy propagation: A junction tree inference algorithm based or lazy evaluation. Artificial Intelligence 113(1-2): 203-245.

Martínez, I. y Rodríguez, C. (2003). Modelos gráficos. En Y. del Águila et. al. (Eds.), Técnicas estadísticas aplicadas al análisis de datos (pp. 217-257). Almería: Servicio de Publicaciones de la Universidad de Almería.

Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (1999). «Boosting Algorithms as Gradient Descent». En S.A. Solla and T.K. Leen and K. Müller, ed. Advances in Neural Information Processing Systems 12. MIT Press. pp. 512-518.

Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (16 de mayo de 1999). Boosting Algorithms as Gradient Descent in Function Space. Archivado desde el original el 22 de diciembre de 2018. Consultado el 29 de septiembre de 2017.

Mitchell, T.M. (1997). Machin Learning. MacGraw-Hill, 1997.

Murphy, P.M. y Aha, D.W. (1995). UCI repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/>.

Nadkarni, S., y Shenoy, P.P. (2001). A bayesian network approach to making inferences in causal maps. European Journal of Operational Research, 128, 479-498.

Nadkarni, S., y Shenoy, P.P. (2004). A causal mapping approach to constructing bayesian networks. Decision Support Systems, 38, 259-281.

Narang, Sharan, Gregory Diamos, Erich Elsen, Paulius Micikeviciusand Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston andOleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. "MIXED Precision Training."

Natekin, A., y Knoll, A. (2013). Gradient Boosting Machines, a tutorial. Frontiers in Neurorobotics

Neapolitan R. (2003). Learning Bayesian Networks. Prentice Hall.

Neapolitan, R.E. (2004). Learning bayesian networks. Pearson Prentice Hall.

Nielsen, T.D., y Jensen, F.V. (2009). Bayesian networks and decision graphs. Springer Science & Business Media.

Pazzani, M (1997). Searching for dependencies in Bayesian classifiers. Learning from Data: Artificial Intelligence and Statistics V, 239–248, Springer Verlag.

Pearl, V. (1988). Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference. Morgan Kaufmann Publisher, Inc.

Peña, J.M., Lozano, J.A. y Larrañaga, P. (2002). Learning recursive Bayesian multinets for data clustering by means of constructive induction. Machine Learning, 47:63-89.

Quinlan, J.R. (1993). C4.5: Programs for machine learning, Morgan Kaufmann Publishers, Inc., California (USA).

Ramoni. M. y Sebastiani, P. (2001). Robust learning with missing data. Machine Leaming, 45(2):147-170.

Raschka y Mirjalili (2019) Python Machine Learning. Aprendizaje automático y aprendizaje profundo con Python,scikit-learn y Tensor Flow. Editorial Marcombo.

Ruder, Sebastian. 2017. "An Overview of Gradient Descent Optimizationalgorithms."

Robles, V. (2003). Clasificación Supervisada basada en redes bayesianas. Aplicación en biología computacional. Universidad Politécnica de Madrid, Facultad de Medicina. Tesis Doctoral.

Ruder, Sebastian. 2017. "An Overview of Gradient Descent Optimizationalgorithms."

Sahami, M. (1996). Learning limited dependence bayesian classifiers. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pages 335–338.

Salakhutdinov, R. (2009): Learning Deep Generative Models. Ph. D. Thesis. Department of Electrical and Computer Engineering, University of Toronto. 84 p.

Salmerón, A. (1998). Algoritmos de propagación II: Métodos de monte Carlo. En J. A. Gámez y J. M. Puerta editores, Sistemas Expertos Probabilísticos, Ciencia y Técnica nº 20, págs. 65-88. Universidad de Castilla-La mancha, 1 ed.

Schachter, R.D., Anderson, S.K. y Szolovits, P. (1994). Global Conditioning for Probabilistic Inference in Belief Networks. Proceedings of the Uncertainty in AI Conference, San Francisco, CA, Morgan Kaufman: 514–522.

Schapire, R. (2002).The boosting approach to machine learning: an overview. Nonlin. Estimat. Classif. Lect. Notes Stat. 171, 149–171.doi:10.1007/978-0-387- 21579-2_9

Shenoy, P.P. (1992). Valuation-based systems for Bayesian decision analysis. Operation Research 40(3): 463-484.

Simonyan. K. y Zisserman. A. (2014) Redes convolucionales muy profundas para el reconocimiento de imágenes a gran escala

Sutton, C.D. (2005). Classification and regression trees, bagging, and boosting. Handb.Stat. 24, 303–329.doi:10.1016/S0169-7161(04)24011-1

Szegedy. C. et al. (2015) Profundizando con las convoluciones .

Viola, P. y Jones, M. (2001). "Rapid object detection using a boosted cascade of simple features," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001, (Kauai, HI).

Torres. J. (2020) Python Deep Learning. Introducción práctica con Keras y Tensor Flow 2. Editorial Marcombo

Walia, Anish Singh. 2017. "Types of Optimization Algorithms Used in Neural Networks and Ways to Optimize Gradient Descent." 2017. <https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f>.

Webb, G.I., Boughton, J.R. y Wang, Z. (2005). Not so naive bayes: aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24.

Webb, G.I. y Pazzani, M.J. (1998). Adjusted probability naive Bayesian induction. *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence*, pages 285-295.

Xu, Bing, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. "Empirical Evaluation of Rectified Activations in Convolutionnetwork."

Yang, Y., Webb, G.I., Cerquides, J., Korb, K.B., Boughton, J., y Ting, K.M. (2007). To select or to weigh: A comparative study of linear combination schemes for superparent-one-dependence estimators. *Knowledge and Data Engineering, IEEE Transactions on*, 19(12), 1652-1665.

Zaffalon, M. (2002). The naive credal classifier. *Journal of Statistical Planning and Inference*, 105(1):5-21.

Zhang,T. y Yu,B.(2005).Boosting with early stopping: convergence and consistency. *Ann.Stat.* 33, 1538–1579.

Zhang, N. y Poole, D. (1996). Exploiting Causal Independence in Bayesian Network Inference. *Journal of Artificial Intelligence Research*, v.5, p. 301-328.

Zheng, F. y Webb, G.I. (2007). Finding the right family: parent and child selection for averaged one-dependence estimators. *Machine Learning: ECML 2007*, pages 490–501. Springer.