Aprendizaje Automático I

Ana X. Ezquerro

ana.ezquerro@udc.es, GitHub

Grado en Ciencia e Ingeniería de Datos
Universidad de A Coruña (UDC)

Curso 2020-2021

Tabla de Contenidos

Bloque I
Introducción

Tema 1: Introducción al Aprendizaje Automático

- Motivación: Desarrollar sistemas que aprendan solos, sin necesidad de programarlos.
- Introducir conocimiento a través de ejemplos.
- Un sistema que interactúa con un entorno o lo observa, modifica el comportamiento de sus sistemas o de su representación interna para mejorar de acuerdo a algún criterio de evaluación.

1.1. Aprendizaje Natural

1.1.1. Perspectiva conductista

- **Habituación**. El sistema aprende a no prestar atención de ciertas entradas y se habitua a ello (muy complicado en sistemas artificiales).
- **Aprendizaje asociativo**.
  - Condicionameinto clásico. Responder a un estímulo con la misma acción que se utilizaría en respuesta a otro estímulo asociado (i.e. el perro de Paulov).
  - Prueba y error. Realizar acciones aleatoriamente hasta encontrar la acción asociada a una recompensa.
  - Aprendizaje latente. No se obtiene una asociación con una respuesta inmediata, sino un conocimiento del entorno que permite aprender posteriormente de forma mucho más rápida.
- **Impronta**. Proceso biológico de aprendizaje por el cual las crías se identifican con los adultos de su especie y aprenden de ellos (observación e imitación). No existe dentro de la inteligencia artificial.
- **Imitación**. Imitar el comportamiento de otro ser. No existe dentro de sistemas artificiales.

1.1.2. Perspectiva fisiológica

- **Conexionismo**. Estudiar la estructura del sistema nervioso de los animales, su capacidad de aprendizaje y almacenamiento de conocimiento mediante las características del conexionado de neuronas.
- **Aprendizaje genético**.
  - Estudia la evolución de las especies por medio de cruces y mutaciones en los cromosomas de los individuos.

- La adaptación al medio funciona como criterio de selección a la hora de evaluar el fruto de los cambios genéticos de los individuos.

1.1.3. Perspectiva del aprendizaje humano

- **Aprendizaje deductivo**. Obtener nuevos conocimientos a partir de una información previa por medio de razonamientos lógicos.
- **Aprendizaje inductivo**. Obtención de conceptos y relaciones a partir de una serie de eejmplos y contraejemplos.
- **Aprendizaje por analogía**. Afrontar nuevas situaciones estudiado la analogía con otras situaciones ya conocidas, adaptado un conocimiento previo a la nueva situación.

1.2. Tipos de Aprendizaje Automático

1.2.1. Clasificación en base a la información respecto al comportamiento deseado

- **Aprendizaje Supervisado**. El comportamiento deseado se representa por medio de un conjunto de *ejemplos* que permiten definir un criterio para evaluar el comportamiento real del sistema.
- **Aprendizaje No Supervisado**. Sin ningún tipo de señal que indique un comportamiento deseado para el sistema. El criterio de evaluación se basa en la *regularidad* de los grupos de datos identificados.
- **Aprendizaje por refuerzo**. El comportamiento deseado no se representa mediante ejemplos, sino mediante una cierta evaluación sobre los resultados que genera el sistema en su entorno.

1.2.2. Clasificación en base a la representación del conocimiento obtenido

- **Aprendizaje simbólico**. Adquirir conocimiento que permita representaciones de conceptos (reglas lógicas, valores de atributos...)
- **Aprendizaje no simbólico**. Adquirir un conocimiento representado por medio de factores numéricos no relacionados directamente con un concepto determinado (pesos de una RNA, cromosomas en un sistema genético...)
- **Aprendizaje mixto**. Se trata de adquirir conceptos expresados por medio de factores numéricos.

1.3. Aprendizaje Inductivo

- **Atributo**: Característica que define a un elemento de un conjunto.
- **Instancia**: Colección de valores de atributos.
- **Clase**: Cada uno de los subconjuntos disjuntos en los que se quiere dividir el conjunto de instancias.

El **aprendizaje inductivo** consiste en la adquisición de un nuevo conocimiento por inferencia inductiva sobre los datos del entorno.

- Deducción: Razonamiento de lo general a lo específico.
- Inducción. Razonamiento de lo específico a lo general.

El aprendizaje inductivo se basa en la siguiente hipótesis: "Si una hipótesis describe bien el concepto de acuerdo a un número suficientemente significativo de ejemplos de aprendizaje, también describirá bien el concepto en futuros ejemplos".

1. Proceso inductivo: Desarrollar un sistema de AA que describa bien el concepto para un número suficientemente significativo de ejemplos.
2. Proceso deductivo: Aplicar ese sistema de AA a nuevos ejemplos.

El conjunto de ejemplos del proceso inductivo, también denominado conjunto de entrenamiento, debe:

- Ser significativo: Número suficiente de ejemplos.
- Ser representativo: Todas las regiones del espacio de estados deben estar cubiertas.

1.4. Representaciones del conocimiento

1.4.1. Redes semánticas

- Se usan para representar **conceptos**.
- Los **nodos** son objetos o conceptos del dominio del problema.
- Los **arcos** son las relaciones o asociaciones entre los nodos.
- El aprendizaje puede consistir en crear/eliminar nodos y arcos.

1.4.2. Árboles de decisión

- Permite clasificar ejemplos basados en valores de atributos.
- Nodos hoja: Representan una clasificación.
- Nodos internos: Representan una cuestión respecto al valor de un atributo.
- Ramas: Diferentes valores.
- El aprendizaje consiste en la creación y modificación de estos árboles.

1.4.3. Reglas de clasificación

- Son reglas IF-THEN en las que el antecedente se refiere a conjunciones de valores de atributos y el consecuente es la elección de una clase.

- El aprendizaje consiste en la creación y modificación de estas reglas.

1.4.4. Programación lógica

- Basada en la **lógica de predicados**.
- La representación del conocimiento se compone de reglas y hechos.
- El mecanismo de inferencia es la deducción automática.
- El aprendizaje consiste en la construcción de hechos y reglas de forma automática a partir de los ejemplos (Programación Lógica Inductiva).

1.4.5. Sistemas difusos

- Conjuntos en los que la pertenencia puede tomar un valor continuo entre 0 y 1.
- Puede utilizar reglas IF-THEN basadas en conceptos descritos por conjuntos difusos.
- El aprendizaje consiste tanto en la construcción y modificación de reglas (aprendizaje simbólico) como en la de los conjuntos difusos (aprendizaje subsimbólico).

1.4.6. Redes neuronales

- Formadas por elementos de procesamiento simple (neuronas) con conexiones que poseen un peso asociado.
- El aprendizaje consiste en la modificación de los pesos para adaptar el comportamiento en la red.

Bloque II
Aprendizaje Supervisado

Tema 2: Introducción al Aprendizaje Supervisado

2.1. Conceptos del Aprendizaje Supervisado

El aprendizaje supervisado se divide en dos fases:

1. **Entrenamiento**. Etapa en la que se presenta al *sistema de aprendizaje* un conjunto de entrenamiento ara que "aprenda" a partir de ellos.
   - El sistema modifica sus parámetros de forma gradual hasta que su salida se ajuste a la salida deseada.
   - Se requiere una **medida de rendimiento/precisión/error**.
2. **Utilización**. Etapa posterior en la que se le pide al sistema que *generalice* con nuevos ejemplos no vistos anteriormente (conjunto de test).

Notación:

- Conjunto de entrenamiento: T_{train}.
- Conjunto de test: T_{test}.

2.1.1. Generalización, Sobreajuste y Sobreentrenamiento

- **Generalización**: El sistema no debe *memorizar* el conjunto de entrenamiento y sus respuestas, sino que debe *aprender* (conceptualizar) la estructura de los datos.
- **Sobreajuste** (*overfitting*): El modelo aprende en profundidad del conjunto de entrenamiento, pero no generaliza bien en un nuevo caso.

- **Sobreentrenamiento** (*overtraining*): Se obliga al modelo a aprender los patrones de entrenamiento, perdiendo capacidad de generalización. El resultado es un modelo sobreajustado.

Sobreajuste vs Sobreentrenamiento: El sobreajuste no tiene por qué aparecer solo por el sobreentrenamiento. Un modelo se sobreajusta también cuando es demasiado complejo para el problema que se quiere resolver.

2.1.2. ¿Cómo evitar el sobreajuste?

- Controlar la complejidad del modelo.
- Preferir el modelo más simple capaz de explicar los datos frente al complejo.

[Dietterich] Para cualquier algoritmo de aprendizaje se dabe encontrar el equilibrio entre:

- La complejidad del modelo.
- El número de ejemplos disponibles.
- El error de generalización.

2.2. Dimensión Vapnik-Chervonenkis (VC)

Sea una familia de funciones:

$$f(x;\Theta) \longrightarrow \begin{cases} f & \text{representa la familia} \\ \Theta & \text{es el conjunto de parámetros} \end{cases}$$

La dimensión Vapnik-Chervonenkis mide la complejidad/capacidad de $f(x;\Theta)$. Esto es, el **máximo número de ejemplos** (*cardinalidad*) que pueden explicar.

A mayor complejidad de $f(x;\Theta)$:

- Mayor capacidad para ajustar los datos de entrenamiento.
- Menor capacidad de generalización.

2.2.1. Rotura de los datos (*shatter*)

Sea $T_{\text{train}} = (\mathbf{x}, \vec{y})$ el conjunto de entrenamiento de n patrones, donde:

- $\mathbf{x} = (\vec{x}_1, ..., \vec{x}_n)^t$ es la matriz de los valores de los atributos, $\mathbf{x} \in \mathbb{R}^{n \times m}$ (n patrones, m atributos).
- $\vec{y} = (y_1, ..., y_n)$ sus correspondientes clases de pertenencia, $Y \in \{0, 1\}$ (2 clases).

Dada una distribución de $\mathbf{x}$ en el espacio $\mathbb{R}^m$, existen 2^n posibles formas (también denominadas **dicotomías**) de asignar las clases.

Se dice que f hace un *shatter* (rotura) de $\mathbf{x}$ si dada una distribución de $\mathbf{x}$ en el espacio, para cada dicotomía existe un Θ tal que $f(x;\Theta)$ la resuelve.

La dimensión VC de $f(x;\Theta)$ es el mayor número de patrones n distribuidos en el espacio $\mathbb{R}^m$ de una forma concreta cuyas dicotomías se pueden resolver con f.

Dicho de otra forma, si la dimensión VC de $f(x;\Theta)$ es h, entonces:

- Existe una distribución de $\mathbf{x} = (\vec{x}_1, ..., \vec{x}_h)$ en el espacio para la cual se puede hacer un *shatter* de todas sus dicotomías.
- No existe ninguna distribución de $\mathbf{x} = (\vec{x}_1, ..., \vec{x}_h, \vec{x}_{h+1})$ en el espacio para la cual se puede hacer un *shatter* de todas sus dicotomías.

2.2.2. Error en el test

La dimensión VP permite calcular el **riesgo esperado** (también conocdio como *error de test*, e_{test}) a partir del **riesgo empírico** (*error de entrenamiento*, e_{train}).

Para $n > h$, la **cota superior** para el riesgo esperado es:

$$\text{CotaSuperior}(e_{\text{test}}) = e_{\text{train}} + \underbrace{\sqrt{\frac{h\log(2n/h+1) - \log(n/4)}{n}}}_{\text{Confianza VC}}$$

Cuando n/h aumenta:

- n aumenta (más patrones en T_{train}).
- h disminuye (modelo menos complejo).
- Confianza VC disminuye.
- $e_{\text{train}} \approx e_{\text{test}}$.
- Más poder de generalización.

Cuando n/h disminuye:

- n disminuye (menos patrones en T_{train})
- h aumenta (modelo más complejo).
- Confianza VC aumenta.
- e_{train} ń e_{test}.
- Sobreajuste, menor poder de generalización.

Este razonamiento matemático nos da a entender que:

- Muchos patrones de entrenamiento + modelo menos complejo = Generalización.
- Pocos patrones de entrenamiento + modelo complejo = Sobreajuste.

2.3. ¿Cómo conseguir un buen modelo?

Un buen modelo es aquel que tenga un **error de test bajo**:

1. Error de entrenamiento bajo (evitar el *underfitting*).

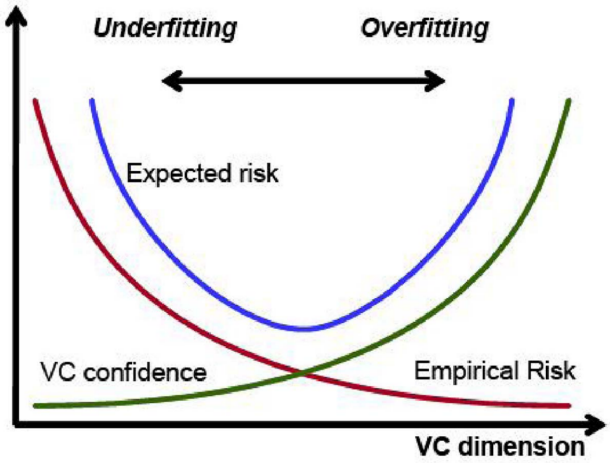
2. Confianza VC baja (evitar el *overfitting*).

Valores altos en *underfitting* se deben a que el modelo no es lo suficientemente complejo para el problema a resolver. La solución está en aumentar la complejidad del modelo.

Para evitar el *overfitting* se debe disminuir la confianza VC. Esto se consigue:

1. Aumentando el número de patrones de entrenamiento n (disminuye también el efecto del ruido).
2. Disminuyendo la complejidad del modelo h. Cuanto más sencillo sea el modelo, menos capacidad tiene para aprender la forma compleja dle ruido.
3. Si se escoge el modelo complejo, detener de forma prematura el entrenamiento para no permitir que se "desarrolle" su complejidad.

El **modelo óptimo** se consigue buscando un equilibrio entre el error de entrenamiento y la dimensión VC del modelo.



2.3.1. ¿Cómo saber la complejidad idónea?

- En teoría se selecciona el modelo para el cual la cota al error de test es mínima.
- En la práctica, calcular la confianza VC no es viable en la mayoría de las situaciones. Se toman expresiones explícitas.

2.4. Diseño experimental

Una vez elegido el modelo y un conjunto de entrenamiento T_{train}, el diseño experimental que consta de las siguientes fases:

1. Preprocesado de los datos.
2. Entrenamiento eficaz del modelo.
3. Estimación del rendimiento real (error cometido) por el modelo entrenado.

2.5. Preprocesado de datos

1. Transformación de datos (*Data Transformation*).
   - Limpieza de datos (*Data Cleaning*).
   - Normalización.
   - Agregación
   - Generalización.
   - Construcción de nuevos atributos.
2. Reducción de datos (*Data Reduction*).

2.5.1. Limpieza de datos con valores nulos

Sea $(\vec{x}_0, y_0) = (x_{01}, ..., x_{0m}, y_0)$ un dato de nuestro conjunto de entrenamiento con un **valor nulo** en el atributo A_i ($x_{0i} = \text{NULL}$). Entonces podemos tomar las siguientes estrategias:

- Ignorar el dato.
- Rellenar el dato manualmente.
- Usar una constante global para rellenarlos (*unknown*).
- Usar la media del atributo para sustitución.
- Usar la media del atributo obtenida con los datos que pertenecen a la misma clase.
- Usar técnicas de minería de datos para predecir el valor más probable en cada caso.

Las **técnicas de minería de datos** consisten en construir un modelo de regresión para estimar el valor nulo x_{0i}.

En el nuevo problema de regresión, las variables utilizadas serán $\vec{A} - A_i$ (todos los atributos del problema original excepto aquel con el valor nulo), los patrones utilizados serán todos excepto $(\vec{x}_0, y_0)$ y la variable respuesta que se quiere estimar será A_i.

- Si el problema original es de regresión, podemos introducir la salida del problema original (Y) en el nuevo problema como otra variable explicativa.
- Si el problema original es de clasificación no debemos introducir Y como variable explicativa. Además, podemos:
  - Utilizar todos los patrones de T_{train}.
  - Utilizar todos los patrones de la misma clase que $(\vec{x}_0, y_0)$

2.5.2. Limpieza de datos con ruido

El **ruido** de los datos se modela como una variable aleatoria e:

$$\tilde{X} = X + e$$

donde X son los valores numéricos que podemos obtener en el conjunto de entrenamiento y test. Para suavizar estos valores minimizando e tenemos distintas estrategias:

1. **Binning**. Los valores ordenados se agrupan en porciones y se suaviza cada porción (tratamiento local del ruido).
   - Suavizado por media.
   - Suavizado por límita.
   - División por anchura (mismo rango de valores de cada bin).
   - División por longitud (mismo número de valores en cada bin).
2. **Clustering**. Se crean clusters y los valores que "caen" fuera de los clusters se consideran valores fuera de rango.
3. **Regresión lineal**. Aplicar un modelo de regresión lineal donde las variables explicativas son el resto de atributos y sustituir los datos del atributo que funciona como variable respuesta por los valores que estime dicho modelo.

2.5.3. Normalización de los datos

Normalización mín/máx:

$$x' = \frac{x - \text{mín}}{\text{máx} - \text{mín}}(\text{nuevo máx} - \text{nuevo mín}) + \text{nuevo mín}$$

- Adecuada si los valores están acotados por un rango.
- Generalmente entre [0, 1] o [-1, 1].

Normalización mediante escalado decimal: $x' = \frac{x}{10^j}$

Normalización de media 0:

$$x' = \frac{x - \mu}{\sigma}$$

- Si los valores son resultado de una medición (no hay rangos)

2.5.4. Otras transformaciones de datos

- **Agregar** valores de atributos. Cuando es útil tener como atributos relaciones matemáticas entre atributos originales.

- **Generalizar** mediante jerarquías de conceptos y sustituir valores (categóricos o numéricos) por otros más abstractos.
- **Construir** atributos nuevos (añadiendo o combinando atributos originales) para facilitar el aprendizaje del algoritmo.
- **Reducir** los datos:
  - Reducir el número de datos.
  - Discretizar y jerarquizar los atributos.
  - Reducir las dimensiones con PCA.

2.5.5. La maldición de las dimensiones

Cuando los datos se ubican en un espacio de alta dimensionalidad (muchos atributos), la cantidad de valores crece exponencialmente.

- Es muy posible que dichos atributos contentan información irrelevante o poco significativa.
- Además, el número de patrones necesarios para obtener un buen modelo aumenta con la dimensionalidad.

Solución: Reducción de la dimensionalidad.

- *Principal Component Analysis* (PCA)
- *Fisher's Linear Discrminant*
- *Multi-Dimensional Scaling*
- *Independnet Component Analysis* (ICA)

Ventajas de tener menos atributos:

- Reducción de la complejidad y tiempo de ejecución.
- Mejora de la capacidad de generalización del sistema.

2.5.6. Análisis de Componentes Principales (PCA)

PCA es una técnica que transforma un conjunto original de p variables/atributos en otro conjunto de m nuevas variables **incorreladas** entre sí ($m < p$). Las nuevas variables son **combinaciones lineales** de las anteriores y se van construyendo según el orden de variabilidad total que recogen del conjunto original.

Características:

- La suma de las varianzas de las variable originales es igual a la suma de las varianzas de las componentes principales.

2.5.7. Análisis de Componentes Independientes (ICA)

ICA permite la descomposición de la señal medida en distintos sitios como una mezcla lineal de fuentes de señales que son **máximamente independientes**.

1. Cálculo de las componentes independientes. Es decir, combinaciones lineals de las variables originales.
2. Las componentes independientes son estadísticamente independientes (ejes no ortogonales).
3. Eliminación de las componentes menos relevantes.

2.6. Aprendizaje supervisado

Dado un conjunto de entrenamiento $T = (\mathbf{x}, \vec{y})$, se desea un modelo que, ante nuevos valores de los atributos $\vec{x}$ sea capaz de devolver un valor consistente con el conjunto de datos.

La capacidad de un modelo para resolver un problema está ligada a los ejemplos utilizados durante el aprendizaje. El conjunto de entrenamiento debe ser:

1. Significativo. Número suficiente de ejemplos.
2. Representativo. Todas las regiones del espacio de estados deben estar suficientemente cubiertas.
3. Equilibrado. El número de patrones de cada tipo debe estar equilibrado con el del resto.

Si un modelo ha sido bien escogido, con una complejidad adecuada, y entrenado con un conjunto de entrenamiento bien escogido, entonces tendrá buena capacidad de generalización.

2.7. Aplicaciones del Aprendizaje Supervisado

2.7.1. Predicción

Objetivo. Determinar la función que mapea un conjunto de variables de entrada en una (o más) variables de salida.

- De tipo numérico.
- Basada en supuestos estadísticos.

2.7.2. Regresión

Objetivo. A partir de un conjunto de patrones de la forma $T_{\text{train}} = (\mathbf{x}, \vec{y})$, encontrar modelo de relación entre $\mathbf{x}$ (entradas) y $\vec{y}$ (salidas deseadas).

El modelo debe ser generalizable: ante nuevos patrones desconocidos debe devolver salidas coherentes.

- Predicción, ajuste de curvas, etc... son casos particulares de regresión.

- Para evaluar la bondad de un modelo de regresión se calcula algún **tipo de error** (error medio, ECM, etc.).

2.7.3. Clasificación supervisada

Objetivo. Dadas las entradas de un patrón ($\mathbf{x}$) clasificarlo en un número finito de clases (conjunto C). Busca una función de mapeo que permita separar las clases.

- El número de clases es finito y conocido.
- Se conoce la pertenencia a la clase de cada patrón de T_{train}.
- Para evaluar la bondad de un clasificador existen unos criterios de evaluación.

2.8. Criterios de evaluación de clasificadores

2.8.1. Matriz de Confusión

		Predicción	
		Negativo	Positivo
Real	Negativo	VN	FP
	Positivo	FN	VP

$$\underset{(accuracy)}{\text{Precisión}} = \frac{VN + VP}{VN + FN + VP + FP}$$

$$\underset{(error\ rate)}{\text{Tasa de error}} = \frac{FN + FP}{VN + FN + VP + FP} = 1 - \text{Accuracy}$$

$$\underset{(recall)}{\text{Sensibilidad}} = \frac{VP}{FN + VP}, \qquad \underset{(VPP,\ precision)}{\text{Valor Predictivo Positivo}} = \frac{VP}{VP + FP}$$

$$\underset{(specificity)}{\text{Especificidad}} = \frac{VN}{VN + FP}, \qquad \underset{(VPN)}{\text{Valor Predictivo Negativo}} = \frac{VN}{VN + FN}$$

$$\text{Media armónica } (F_1\text{-score}) = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Observaciones:

- **Sensibilidad**: "De los positivos, cuántos acierta correctamente". Usar cuando es importante no clasificar un negativo como positivo.
- **VPP**: "De los que clasifica como positivos, cuántos acierta correctamente". Usar cuando no es importante clasificar un negativo como positivo.
- F_1-**score**: La precisión se suele usar cuando la distribución de clases es parecida, mientras que F_1-score es mejor métrica cuando hay clases desbalanceadas.
  - Cuando VP y VN más importantes $\longrightarrow$ usar la precisión.
  - Cuando FN y FP son más importantes, $\longrightarrow$ usar F_1-score.
- Si en los casos reales no hay predicciones positivas, VPP = NaN.

Otras medidas:

- Tasa de Falsos Positivos = 1 - Especificidad.
- Tasa de Falsos Negativos = 1 - Sensibilidad.

2.8.2. Métricas para problemas multiclase

		Predicción		
		Clase 1	$\cdots$	Clase c
Real	Clase 1	v_{11}	$\cdots$	v_{1c}
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	Clase c	v_{c1}	$\cdots$	v_{cc}

Para calcular las métricas a partir de la matriz de confusión multiclase:

- Considerar c problemas binarios.
- Calcular las métricas para cada subproblema.

La correspondiente métrica global se obtiene siguiendo cualquiera de estas tres estrategias:

1. *Macro*. Se calcula la métrica para cada subproblema y se realiza la media arimética.
2. *Weighted*. Se calcula la métrica para cada clase y se calcula la media ponderada por el número de instancias de cada clase. Útil cuando las clases están desbalanceadas.
3. *Micro*. Se calculan las métricas de forma global contando el número total de VP, FN, FP y VN. Útil cando las clases no son mutuamente excluyentes.

2.8.3. Repetibilidad

La repetibilidad es el grado con el cual otras mediciones o cálculos muestran el mismo o similares resultados. Se refiere a la fiabilidad del experimento, o a la estabilidad de un sistema.

Para obtener la repetibilidad es necesario repetir el experimento para:

- Promediar los resultados de las métricas obtenidas.
- Calcular la desviación típica.

2.8.4. Curva ROC (*Receiver Operating Charactersitic*)

Definición. Espacio bidimensional (X, Y) entre 0 y 1.

- Eje X: Tasa de falsos positivos (1-especificidad)
- Eje Y. Tasa de verdaderos positivos (sensibilidad)

El clasificador se sitúa en un punto con coordenadas (x, y).

Si $(x, y) = (0, 1)$:

- Tasa de falsos positivos = 0 %. Todos los negativos fueron clasificados correctamente.
- Tasa de verdaderos positivos = 100 %. Todos los positivos fueron clasificados correctamente.
- **Clasificador perfecto**. Clasifica correctamente todos los positivos y negativos.

Si $(x, y) = (0, 0)$:

- Tasa de falsos positivos = 0 %. Todos los negativos fueron clasificados correctamente.
- Tasa de verdaderos positivos = 0 %. Ningún positivo fue clasificado correctamente.
- **Se clasificaron todos los casos como negativos**.

Si $(x, y) = (1, 1)$:

- Tasa de falsos positivos = 100 %. Todos los negativos fueron clasificados como positivos.
- Tasa de verdaderos positivos = 100 %. Todos los positivos fueron clasificados como positivos.
- **Se clasificaron todos los casos como positivos**.

Si $(x, y) = (1, 0)$:

- Tasa de falsos positivos = 100 %. Todos los negativos fueron clasificados como positivos.
- Tasa de verdaderos positivos = 0 %. Todos los positivos fueron clasificados como negativos.
- **Todas las clasificaciones son incorrectas**.

Si $x = y$: Clasificador aleatorio.

En muchas ocasiones los clasificadores tienen algún parámetro que se puede variar para:

- Incrementar VP a costa de incrementar FP.
- Drecementar FN a costa de decrementar VN.

Alterando el valor de dicho parámetro se podrían obtener distintas coordenadas en el espacio bidimensional (X, Y). Dicho conjunto de coordenadas se denomina curva ROC.

En general, cuanto más se acerque la curva al valor $(0, 1)$, mejor es el clasificador. Para medir la bondad del clasificador se puede usar el AUC (*Area Under the Curve*), que es el porcentaje de espacio comprendido debajo de la curva y encapsula toda la información contenida en la matriz de confusión.

2.8.5. Índice Kappa

El índice Kappa mide la concordancia entre dos clasificadores con la matriz de confusión entre ambos:

		Método B	
		Negativo	Positivo
Método A	Negativo	a	b
	Positivo	c	d

La forma más sencilla de medir la concordancia es calculan la proporción de coincidencias frente al total de individuos:

$$\frac{a+d}{n}$$

Sin embargo, aunque no exista ninguna relación entre los dos métodos de clasificación, se encontrará algún grado de concordancia entre ellos debido al azar.

El **índice de concordancia Kappa** permite determinar hasta qué punto la concordancia observada es superior a la que se espera obtener por puro azar.

		Método B		
		Negativo	Positivo	
Método A	Negativo	a	b	$f_1 = a + b$
	Positivo	c	d	$f_2 = c + d$
		$c_1 = a + c$	$c_2 = b + d$	n

- $\mathbb{P}(A^-) = f_1/n =$ probabilidad de que el método A clasifique un negativo.
- $\mathbb{P}(A^+) = f_2/n =$ probabilidad de que el método A clasifique un positivo.

- $\mathbb{P}(B^-) = c_1/n =$ probabilidad de que el método B clasifique un negativo.
- $\mathbb{P}(B^+) = c_2/n =$ probabilidad e que el método B clasifique un positivo.

1. Sea P_e la probabilidad de concordancia esperaza por azar, si el método A y el método B son independientes:

$$\mathbb{P}(A^- \wedge B^-) = \frac{f_1}{n} \cdot \frac{c_1}{n} = \frac{f_1 c_1}{n^2}$$

$$\mathbb{P}(A^+ \wedge B^+) = \frac{f_2}{n} \cdot \frac{c_2}{n} = \frac{f_2 c_2}{n^2}$$

$$P_e = \frac{f_1 c_1 + f_2 c_2}{n^2}$$

2. Entonces $1 - P_e$ es el margen de concordancia posible no atribuible al azar.
3. Sea P_0 la proporción de concordancia observada:

$$P_0 = \frac{a + d}{n}$$

4. Entonces $P_0 - P_e$ es la **concordancia observada no atribuible al azar**.

El **índice Kappa** K se define como:

$$K = \frac{P_0 - P_e}{1 - P_e}$$

- Si $P_0 = 1$, entonces $K = 1$. Hay concordancia perfecta.
- Si $P_0 = P_e$, entonces $K = 0$. La concordancia observada es igual a la esperada por el azar.
- Si $P_0 < P_e$, entonces $K < 0$. La concordancia observada es inferior a la esperaza por azar.

Índice Kappa	Grado de acuerdo
< 0	Sin acuerdo
$[0, 0{,}2]$	Insignificante
$[0{,}2, 0{,}4]$	Bajo
$[0{,}4, 0{,}6]$	Moderado
$[0{,}6, 0{,}8]$	Bueno
$[0{,}8, 1]$	Muy bueno

El índice Kappa es generalizable a clasificaciones multinomiales:

	M_1	$\cdots$	M_l	
M_1	v_{11}	$\cdots$	v_{1l}	f_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
M_l	v_{l1}	$\cdots$	v_{ll}	f_l
	c_1	$\cdots$	c_l	n

$$P_e = \sum_{i=1}^{l} \frac{f_i c_i}{n^2}$$

2.8.6. Sobre los criterios de evaluación de clasificadores

- La bondad de un clasificado nunca se evalúa sobre el conjunto de entrenamiento.
- Es necesario el conjunto de test.
- El conjunto de test nunca participa en el entrenamiento, por tanto los resultados obtenidos con çel dictaminan la bondad de un clasificador.

2.8.7. Técnicas de partición del conjunto de patrones

En clasificación supervisada se suele dividr el conjunto de patrones T en dos subconjuntos disjuntos: el conjunto de *entrenamiento* (T_{train})y el conjunto de *test* (T_{test}).

HoldOut. Partición simple del conjunto de datos T en T_{train} y T_{test}.

- T_{train} se usa para generar el clasificador.
- T_{test} se usa para evaluar.

Validación cruzada (*Cross-Validation*).

1. Se divide T en k subonjuntos del mismo tamaño ($T_1, ..., T_k$).
2. Para cada subconjunto i se usan los $k-1$ subconjuntos restantes como conjunto de entrenamiento ($T_{\text{train}} = T - T_i$) y dicho subconjunto como conjunto test $T_{\text{test}} = T_i$.
3. Se calcula la media de la evaluación.

Leave-One-Out Cross Validation (LOO-CV). Validación cruzada donde $k = n$.

Bootstrapping. El conjunto de entrenamiento se escomo como una muestra aleatorio con reemplazamiento.

2.9. Codificación de la salida

En la codificación de las salidas de un modelo de aprendizaje supervisado tenemos dos opciones:

- Como números reales (únicamente si en el mundo real existe un orden entre las clases equivalentes).
- Como valores booleanos (problemas de clasificación).

2.9.1. Codificación de la salida como valores booleanos

Cuando el problema tiene $c > 2$ clases se usa la codificación **One-Hot-Encoding**.

- Una salida por clase ($\vec{y} = (y_1, ..., y_c)^t$) .
- **Salida deseada**: Toma el valor 1 si esa instancia pertenece a esa clase, y 0 en caso contrario.

$$y_1, ..., y_c \in \{0, 1\}$$

- **Salida obtenida**:
  1. La salida obtenida son valores reales: $\vec{y} \in \mathbb{R}^c$.
  2. Se aplica la función softmax a cada valor de $\vec{y}$:

$$\widehat{y}_i = \frac{e^{y_i}}{\sum_{j=1}^{c} e^{y_j}} \in [0, 1], \qquad i = 1, ..., c$$

  3. Se cumple que $\sum_{i=1}^{c} \widehat{y}_i = 1$.
  4. Cada $\widehat{y}_i$ representa la probabilidad de pertenencia a la clase i, para $i = 1, ..., c$.
  5. Salida final: Clase con mayor probabilidad.
- Cálculo del error entre $(\vec{y}, \widehat{\vec{y}})$ para todos los patrones del conjunto.

$$\text{Cross-Entropy} = -\sum_{i=1}^{n} \vec{y}_i \cdot \log(\widehat{\vec{y}}_i) = -\frac{1}{n} \sum_{i=1}^{n} \log(\widehat{\vec{y}}_i)$$

Cuando el problema tiene 2 clases se usa la codificación binaria:

- **Salida deseada**. Se codifica como $y \in \{0, 1\}$.
- **Salida obtenida**.
  1. El modelo devuelve una salida real.
  2. Se aplica la función sigmoidal.

$$f(y) = \frac{1}{1 + e^{-y}}$$

- Cálculo del error entre $(y, \widehat{y})$:

$$\text{Cross-Entropy Binaria} = -\frac{1}{n} \sum_{i=1}^{n} \Big(\widehat{y}_i \log(y_i) + (1 - \widehat{y}_i) \log(1 - y_i) \Big)$$

2.10. Estrategias para problemas multiclase

Cuando un modelo solo permite realizar problemas de clasificación binaria, pero queremos aplicarlos a problemas multiclase, podemos usar dos estrategias.

2.10.1. "Uno contra el resto"

Se considera como una colección de problemas binarios (c clasificadores binarios).

1. Para cada subproblema i, los datos pertenecientes a las clase i se les asigna la etiqueta 1 y al resto se les asigna 0.
2. El clasificador i tendrá que detectar si un dato es de la clase i o no.
3. Esquema de votación: Sea $f_i(x)$ la salida del clasificador i (salida numérica), se asigna la etiqueta del clasificador que tiene mayor certeza de que el dato x está en la clase i.

$$y = \arg\max_i \{f_i(x)\}$$

2.10.2. "Uno contra uno"

Se construyen $c(c-1)/2$ problemas donde se separa una clase con cada una de las restantes.

- Para cada par (i, j), donde $i < j$ y $i, j \in \{1, ..., c\}$, se construye un clasificador binario para separar la clase i (a la que se le asigna la etiqueta 1) de la clase j (a la que se le asigna la etiqueta 0).
- Esquema de votación para seleccionar la clase de un dato x.

2.11. Clasificación Supervisada vs Clasificación No Supervisada

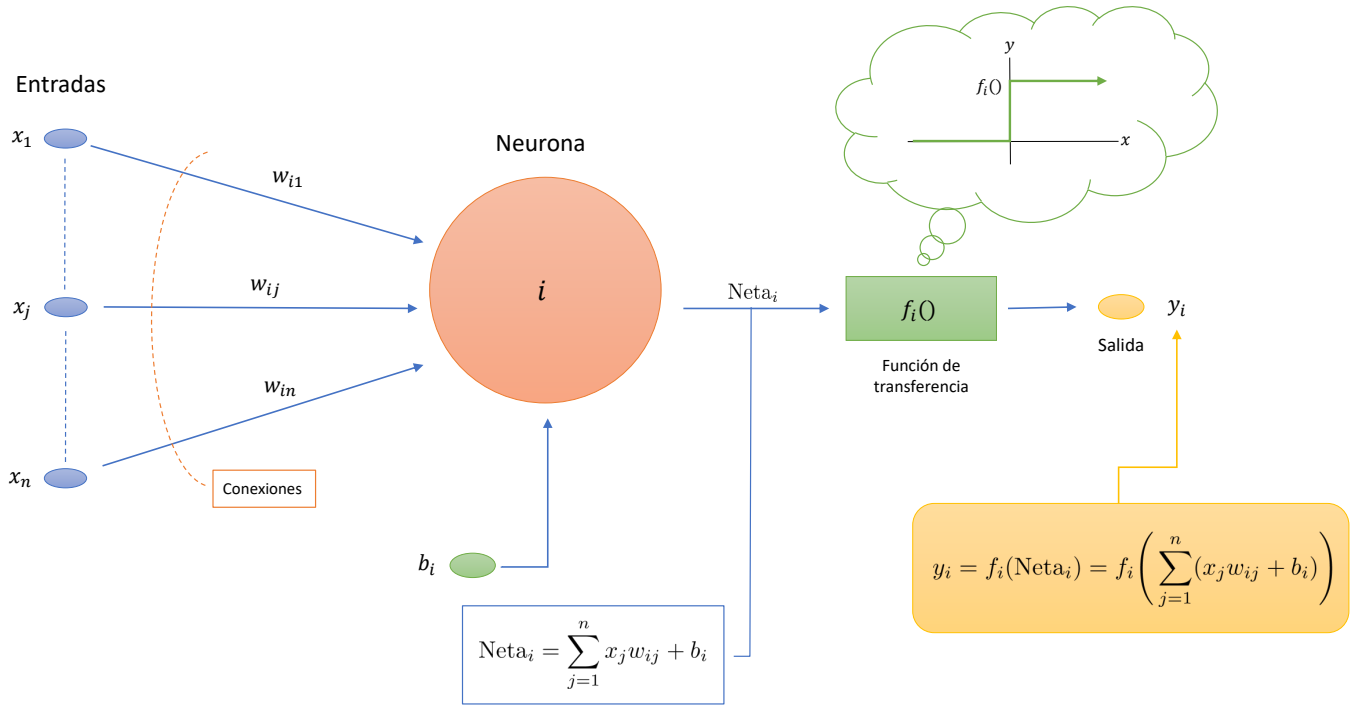
- Clasificación Supervisada: Desarrollar un modelo que, ante un nuevo patrón de clase desconocida, lo clasifique en una de las clases conocidas de forma coherente con los patrones de entrenamiento.
- Clasificación No Supervisada: Analizar los datos mediante algún tipo de agrupamiento.

Clasificación No Supervisada:

- Intenta agrupar las instancias en grupos/clases de forma que los objetos que forman cada grupo deben ser similares.
- No se conoce *a priori* el número de clases.
- No se conoce a qué clase pertenece cada patrón.

Tema 3: Redes de Neuronas Artificiales

3.1. Esquema de una Neurona Artificial



Proceso de aprendizaje/entrenamiento

- Modificación de los pesos de las conexiones + bias.
- Conjunto de entrenamiento:

$$\begin{aligned} \mathbf{x} &= \{\vec{x}_1, ..., \vec{x}_L\} \quad \text{Entradas} \\ y &= \{y_1, ..., y_L\} \quad \text{Salidas deseadas} \end{aligned}$$

- $\vec{x} = (x_1, ..., x_n)$ está compuesto por las entradas que se deberán introducir a la RNA.

Valores de entrada y salida: $\{x_1, ..., x_n, y_i\}$

- Números reales acotados en un intervalo.
- Típicamente $[0, 1]$ o $[-1, 1]$.
- La codificación más simple es la binaria.

Bias: b_i

- Predisposición de una neurona a activarse.
- Valor constante
- Similar a un peso, que no recibe entrada (o entrada$= 1$).

Conexiones: $(w_{i1}, ..., w_{in})$

- Codifican el conocimiento de la red.
- Poseen valores asociados (pesos).
  - Excitatorias ($w_{ij} > 0$).
  - Inhibitorias ($w_{ij} < 0$).
  - Inexistentes ($w_{ij} = 0$).

3.2. ADALINE (*Adaptive Linear Element*)

- Modelo básico de RNA con una única neurona.
- La función de transferencia es tal que $y_i = f_i(\text{Neta}_i) = \text{Neta}_i$.

3.2.1. LMS o Regla Delta

El LMS (*Least Mean Square*) o Regla Delta es un algoritmo de aprendizaje supervisado por corrección del error para determinar los parámetros de Adaline.

1. Aprendizaje: Fija los valores de los pesos de las conexiones y del bias.
2. Supervisado: Necesita para cada salida un error que se comete respecto a la salida deseada.
3. Por corrección de error: Minimiza el ECM sobre todos los patrones de entrenamiento.

Sea el conjunto de entrenamiento $T_{\text{train}} = (\mathbf{x}, \vec{y})$. Para cada patrón k $(1 \leq k \leq L)$ la RNA emite una salida y_k:

$$y_k = \sum_{j=0}^{n} x_j w_j$$

El error cometido en el patrón k se define como la salida deseada menos la salida omitida:

$$\begin{aligned} e_k &= d_k - y_k && \text{Error cometido} \\ E_k &= \frac{1}{2}(d_k - y_k)^2 && \text{Error cuadrático} \\ \text{ECM} &= \sum_{k=1}^{L} e_k = \frac{1}{2}\sum_{k=1}^{L}(d_k - y_k)^2 && \text{Error Cuadrático Medio} \end{aligned}$$

La idea del algoritmo LMS es minimizar el error según $\vec{w} = (w_1, ..., w_n)$, el vector de los pesos de las conexiones, aplicando la regla del gradiente descendente.

- Al derivar el error cuadrático E_k de un patrón k con respecto a un peso w_j, se obtiene la **pendiente de la tangente en** w_j.
- Cuanto **mayor** sea la magnitud de la pendiente, mayor será la componente vertical (y) en relación a la componente horizontal (x), y por tanto más **alejado** estará el punto w_j del mínimo.
- Cuando **menor** sea la magnitud de la pendiente, menor será la componente vertical (y) en relación a la componente horizontal (x), y por tanto más **cercano** estará el punto w_j del mínimo.

La base del algoritmo LMS es realizar un cambio en los pesos proporcional a la derivada del error medido en el patrón k respecto al peso j. Esta derivada toma la forma:

$$\frac{\nabla E_k}{\nabla w_j} = -(d_k - y_k)x_{jk}$$

Y la modificación del peso viene dada por:

$$\Delta_k w_j = -\mu \frac{\nabla E_k}{\nabla w_j}$$

Y por tanto, en cada ciclo t del algoritmo, la modificación del peso j para un patrón k en el instante t es:

$$w_j(t+1) = w_j(t) + \mu(d_k - y_k)x_{jk}$$

donde μ es la tasa o velocidad de aprendizaje y controla la convergencia del entrenamiento.

Variante: Usar ECM sobre todos los patrones y actualizar los pesos según ese error. De esta manera, el algoritmo no es dependiente del orden de introducción de los patrones.

3.3. Perceptrón

- Una capa de entrada y un único elemento en la capa de salida.
- Función de transferencia de tipo umbral.

$$f(\text{Neta}) = \begin{cases} 1 & \text{Neta} > 0 \\ 0 & \text{Neta} < 0 \end{cases}$$

- Capaz de resolver problemas linealmente separables.

Reglas para la actualización de pesos

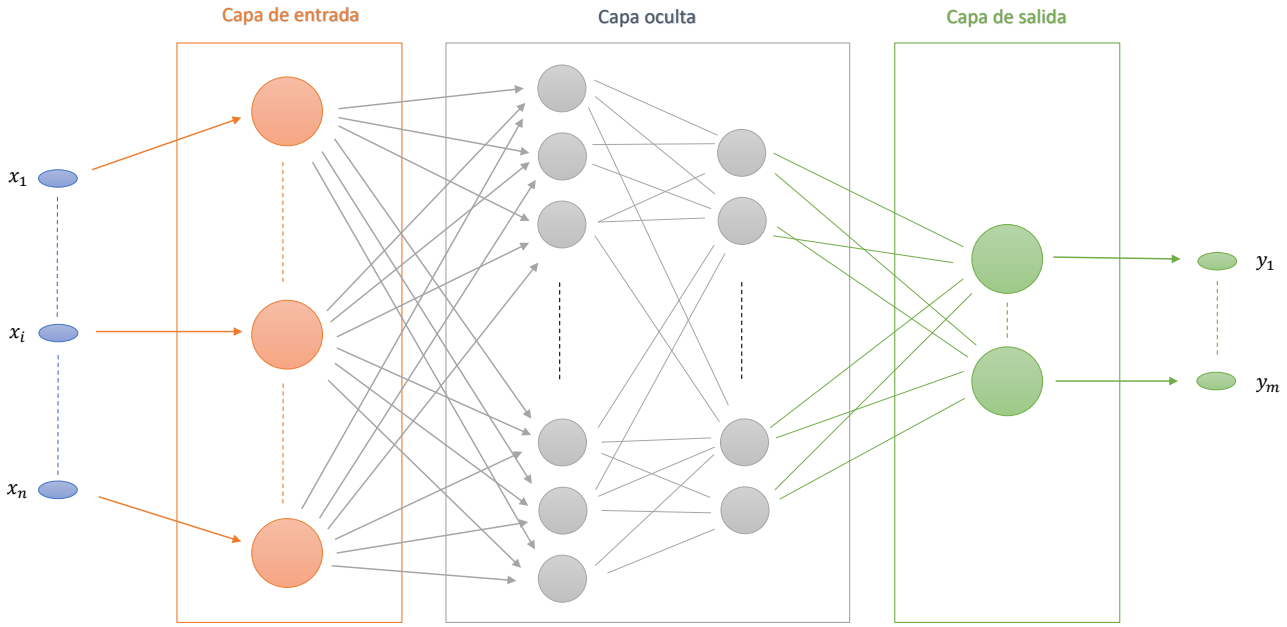
1. $w_i(t+1) = w_i(t)$ cuando la salida es correcta.
   $w_i(t+1) = w_i(t) + x_i(t)$ cuando la salida es -1 y debería ser 1.
   $w_i(t+1) = w_i(t) - x_i(t)$ cuando la salida es 1 y debería ser -1.
2. $w_i(t+1) = w_i(t)$ cuando la salida es correcta.
   $w_i(t+1) = w_i(t) + \mu x_i(t)$ cuando la salida es -1 y debería ser 1.
   $w_i(t+1) = w_i(t) - \mu x_i(t)$ cuando la salida es 1 y debería ser -1.
3. Regla Delta:

$$w_i(t+1) = w_i(t) + \mu[d(t) - y(t)]x_i(t)$$

Se obtiene una **convergencia finita** si el conjunto de entrenamiento es linealmente separable.

3.4. Perceptrón Multicapa (Multilayer Perceptron, MLP)

- Múltiples capas de neuronas que se conectan a las salidas de otras capas.
- Notación: w_{ij} es la conexión que va **desde** la neurona j **hacia** la neurona i.
- Capa de entrada: no computan, sólo almacenan las entradas para pasarlas a la siguiente capa.
- Capas ocultas: formadas por las neuronas ocultas.
- Capas de salida: emiten las salidas de la RNA.
- El conocimiento de la red reside en los pesos de las conexiones y los bias. No es centralizado, sino que está distribuido.
- Ventajas: autoorganización, tolerancia a fallos.
- Desventajas: imposibilidad de explicar su funcionamiento.



3.4.1. Parámetros a escoger

Funciones de transferencia

- No están limitades a ser umbrales o lineales.
- Se asume la misma función para todas las neuronas de la RNA, o al menos para todas las neuronas de una capa.
- La función de trasnferencia de las capas internas no puede ser lineal.
- La función de transferencia en la capa de salida depende del problema a resolver.

Funciones más típicas

- Escalón o umbral (para problemas de clasificación).
- Lineal o lineal mixta (para problemas de regresión y ajuste de curvas).
- Logarítmica sigmoidal (logsig): $f(n) = \dfrac{1}{1+e^{-n}}$
- Tangente sigmoidal hiperbólica (tansig): $f(n) = \dfrac{2}{1+e^{-2n}} - 1.$

Teorema de aproximación universal: Un perceptrón multicapa con suficientes unidades no lineales puede aprender cualqueir tipo de función o relación continua entre un grupo de variables de entrada y salida.

3.4.2. Método de entrenamiento: Backpropagation

- h: capa oculta.
- o: capa de salida.
- p: patrón ($1 \leq p \leq L$).
- j: neurona j en la capa oculta h.
- k: neurona k en la capa de salida.
- w_{ji}^h: conexión desde la neurona $i^{(h-1)}$ hasta la neurona $j^{(h)}$.
- i_{pj}^h: salida de la neurona $j^{(h)}$ para el patrón p.
- o_{pj}: salida de la neurona $j^{(o)}$ para el patrón p.
- y_k: salida deseada para la neurona $k^{(o)}$.
- Error de la neurona $k^{(o)}$ de la capa de salida para el patrón p:

$$\delta_{pk} = (y_{pk} - o_{pk})$$

En la capa de salida puede haber más de una neurona, por tanto el error a minimizar es la suma de los cuadrados de los errores de **todas** las unidades de salida (supongamos que hay un total de n salidas para el patrón p):

$$E_p = \frac{1}{2}\sum_{k=1}^{n} \delta_{pk}^2 = \frac{1}{2}\sum_{k=1}^{n}(y_{pk} - o_{pk})^2$$

La derivada del error es:

$$\nabla E_p = \frac{\partial E_p}{\partial w_{kj}^o} = (y_{pk} - o_{pk})\ f_k^o(\text{Neta}_{pk}^o)\ i_{pj}^{o-1}$$

Por tanto los **pesos en la capa de salida** desde la neurona $j^{(o-1)}$ hasta la neurona $k^{(o)}$ se modifican para el patrón p:

$$\begin{aligned} w_{kj}^o(t+1) &= w_{kj}^o(t) + \Delta_p w_{kj}^o(t) \\ &= w_{kj}^o(t) + \mu \underbrace{(y_{pk} - o_{pk})\ f_k^o(\text{Neta}_{pk}^o)}_{\delta_{pk}^o}\ i_{pj}^{o-1} \end{aligned}$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \mu\ \delta_{pk}^o\ i_{pj}^{o-1}$$

En las capas ocultas no hay valores de salidas deseadas, por tanto no se puede calcular el error ni modificar los pesos de las neuronas en las capas ocultas. La solución viene dada por propagar el error hacia atrás.

- El error de la capa de salida E_p está relacionado con la salida de las neuronas de la capa anterior $(o-1)$.
- Las actualizaciones de los pesos en esa capa $(o-1)$ dependen de todos los términos de errores de la capa de salida.

Escribimoa el error de la capa de salida E_p en función de los pesos de la capa de salida y de las entradas que recibe esa capa, i_{pj}^{o-1}:

$$\begin{aligned} E_p &= \frac{1}{2}\sum_{k=1}^{n}\delta_{pk}^2 = \frac{1}{2}\sum_{k=1}^{n}(y_{pk} - f_k^o(\text{Neta}_{pk}^o))^2 \\ &= \frac{1}{2}\sum_{k=1}^{n}\left(y_{pk} - f_k^o\Big(\sum_j w_{kj}^o i_{pj}^{o-1}\Big)\right)^2 \end{aligned}$$

Las entradas de la capa de salida o son las salidas de las neuronas de la capa oculta $h = o - 1$. Por tanto, las entradas de la capa de salida se pueden escribir en función de las entradas que reciben las neuronas de la capa oculta h (i_{pi}^{h-1}) y de sus pesos (w_{ji}^h).

$$i_{pj}^h = f_j^h(\text{Neta}_{pj}^h) = f_j^h\Big(\sum_i w_{ji}^h i_{pi}^{h-1}\Big)$$

Si reescribimos el error en función de los pesos de la capa oculta:

$$\begin{aligned} E_p &= \frac{1}{2}\sum_{k=1}^{n}\left(y_{pk} - f_k^o\Big(\sum_j w_{kj}^o i_{pj}^h\Big)\right)^2 \\ &= \frac{1}{2}\sum_{k=1}^{n}\left(y_{pk} - \Big(\sum_j w_{kj}^o\ f_j^h\ \Big(\sum_i w_{ji}^h i_{pi}^{h-1}\Big)\Big)\right) \end{aligned}$$

Obtenemos que se puede derivar el error con respecto al peso w_{ji}^h de la neurona oculta j para modificar ese peso.

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k \underbrace{(y_{pk} - o_{pk})\ f_k^o(\text{Neta}_{pk}^o)}_{\delta_{pk}^o}\ w_{kj}^o\ f_j^h(\text{Neta}_{pj}^h)\ i_{pi}^{h-1}$$

Y por tanto:

$$\Delta w_{ji}^h = \mu \cdot f_j^h(\text{Neta}_{pj}^h)\ i_{pi}^{h-1}\ \sum_k \delta_{pk}^o w_{ij}^o$$

Las actualizaciones de los pesos en esa capa dependen de todos los términos de errores de la capa de salida:

$$\delta_{pj}^h = f_j^h(\text{Neta}_{pj}^h)\ \sum_k \delta_{pk}^o w_{kj}^o$$

Las actualizaciones de los pesos para la capa oculta h, anterior a la capa de salida, vienen dadas por:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \mu\delta_{pj}^h i_i^{h-1}$$

	Capa de salida (o) - Neurona k	Capa oculta (h) - Neurona j
Término de error	$\delta^o_{pk} = \delta_{pk}\ f^o_k(\text{Neta}^o_{pk})$	$\delta^h_{pj} = f^h_j(\text{Neta}^h_{pj})\ \sum_k \delta^o_{pk} w^o_{kj}$
Variación de los pesos	$w^o_{kj}(t+1) = w^o_{kj}(t) + \mu\ \delta^o_{pk}\ i^{o-1}_{pk}$	$w^h_{ji}(t+1) = w^h_{ji}(t) + \mu\ \delta^h_{pj}\ i^{h-1}_{pi}$

Proceso del algoritmo de backpropagation

1. Se inicializan los pesos de forma aleatoria.
2. Se calculan los errores de todos los patrones y se modifican los pesos según ese valor de error global.
3. Se repite este proceso durante un número de ciclos (epochs) hasta que se da una condición de parada.

3.4.3. Entrenamiento

El conjunto de entrenamiento debe ser:

- Significativo: Número suficiente de ejemplos.
- Representativo: Todas las regiones del espacio de estados deben estar lo suficientemente cubiertas.
- Equilibrado: El conjunto de entrenamiento no debe tener más ejemplos de un tipo que el resto.

Las entradas deben normalizarse:

- Si todos los valores están acotados usar la normalización de máximo y mínimo.
- Si los valores no están acotados, usar la normalización de media cero.
- Las salidas estarán normalizadas, por tanto habrá que desnormalizarlas.

3.4.4. Control de convergencia

El incremento μ influye en la velocidad con la que converge el algoritmo.

- μ grande: Se pasa por encima del punto mínimo sin conseguir estacionarse en él.
- μ pequeño. Se tarda mucho más en llegar a un mínimo.

Otras opciones:

1. Ir disminuyendo su valor a medida que avanzan los ciclos de entrenamiento:
2. Utilizar el parámetro **Momento**.

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \mu\ \delta^o_{pk}\ i^{o-1}_{pk} + \alpha\ \Delta_p\ w^o_{kj}(t-1)$$

   - Cambios previos en los pesos deberían influir en la dirección del movimiento en el espacio de pesos.

- Introduce cierta inercia en la actualización de los pesos.

Existe la posibilidad de que el algoritmo converja a mínimos locales y no al mínimo global:

- No se puede asegurar a ningún momento que el mínimo que se encuentre sea global o local (**proceso no determinístico**).
- Si se alcanza un mínimo local y el error es satisfactorio, el entrenamiento termina con éxito.
- Si no es así, se pueden realizar las siguientes acciones para solucionar el problema:
  - Cambio de arquitectura (más capas ocultas o más neuronas).
  - Modificación de parámetros de aprendizaje.
  - Emplear un conjunto de pesos iniciales diferentes (entrenar otra vez).
  - Modificar el conjunto de entrenamiento o presentar los patrones en distinta secuencia.

3.4.5. El problema del sobreentrenamiento

- Buen error de entrenamiento, pero mala generalización al evaluar con el conjunto de test.
- La red ha aprendido los patrones de entrenamiento y su ruido, no las características que lo definen.

Las principales causas son:

- Red demasiado ocmpleja para el problema a resolver.
- Entrenamiento durante demasiado tiempo.
- No comprobar la generalización de la red mientras entrena.

Cómo evitar el sobreentrenamiento:

- Añadir a la función de coste la suma de los valores absolutos de los pesos.
- Añadir a la función de coste la suma de los cuadrados de los valores de los pesos.
- Decaimiento de pesos.
- Dropout.
- Normalización por lotes (*batch normalization*).
- Aumento de datos (*Data Augmentation*).
- Parada temprana (*Early Stopping*).
- Utilizar un conjunto de validación.

3.4.6. Conjunto de validación

Al mismo tiempo que se entrena una red con el conjunto de entrenamiento, se va evaluando con el conjunto de validación.

- Si el error del conjunto de validación aumenta, la red se está sobreentrenando.
- Nuevo criterio de parada: Número de ciclos sin mejorar el error de validación.
- Cuando finaliza el entrenamiento, se devuelve la red con el mejor error de validación.

3.4.7. Conjuntos de patrones

- Conjunto de **entrenamiento**.
- Conjunto de **validación**:
  - No participa directamente en el entrenamiento.
  - Supervisa el entrenamiento y lo detiene cuando es necesario.
  - Dicta la red a devolver.
- Conjunto de **test**:
  - No interviene en nada del entrenamiento.
  - Se evalúa una vez finalizado el entrenamiento con la red resultante.
  - Analiza la capacidad de generalización de la red.

3.5. Aplicaciones y conclusiones

3.5.1. Problemas de Clasificación

Si hay dos clases:

- **Una** neurona de salida.
- Función de transferencia (capa de salida): Logarítmica sigmoidal.
- Salida: Probabilidad de clasificar un caso como positivo (1).

Si hay más de dos clases:

- Una neurona de salida por clase.
- Para un patrón, la salida deseada es 1 para aquella neurona que corresponda a esa clase. Para el resto, es 0.
- Función de transferencia (capa de salida): Lineal.
- A las salidas de las neuronas de salida se les aplica la función *softmax*:
- Cada valor representa la probabilidad de pertenencia a esa clase.

3.5.2. Otras aplicaciones

- Predicción

- Clustering.
- Aproximación de curvas.
- Regresión.

3.5.3. Ventajas del las RR.NN.AA

- **Aprendizaje adaptativo**. Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.
- **Generalización** de casos anteriores a nuevos casos.
- **Abstracción** de características esenciales a partir de entradas con información irrelevante.
- **Autoorganización**. Organizan la información del aprendizaje para responder apropiadamente cuando se les presenta datos que no estabana en el entrenamiento.
- **Tolerancia a fallos**.
  - Pueden aprender a reconocer patrones con ruido, distorsionados, o incompletos.
  - Pueden seguir funcionando aunque se destruya parte de la red, pues su información está distribuida en las conexiones entre neuronas con cierto grado de redundancia.
- **Procesamiento en paralelo**.
- **Operación en tiempo real**. Los computadores neuronales pueden ser realizados en paralelo, y se diseñan y fabrican con HW especial para obtener esa capacidad.
- **Fácil inserción dentro de la tecnología existente**.

3.5.4. Desventajas de las RR.NN.AA

- Implementación deficiente como sistemas paralelos.
- No existencia de una regla para construir una RNA para un problema dado.
- Imposibilidad de explicar el funcionamiento de una RNA.
- Cuanto más flexible se requiera que sea una RNA, más información hay que enseñarle.

Tema 4: Máquinas de Soporte Vectorial (SVM)

Las Máquinas de Soporte Vectorial (*Support Vector Machines*) son una técnica de clasificación con:

- Base matemática muy firme.
- Clasificador de dos clases (aunque generalizable a más clases).

Tenemos una serie de observaciones:

$$\begin{array}{ll} \text{Vector:} & \vec{x}_i \in \mathbb{R}^m, \\ & \qquad\qquad i = 1, .., n \\ \text{Etiqueta:} & y_i \in \{-1, +1\} \end{array}$$

Se desea encontrar el hiperplano $\vec{w}^t\vec{x} + b = 0$ definido por el par $(\vec{w}, b)$ tal que se puedan separar los puntos $\vec{x}_i$ de acuerdo a la función:

$$f(\vec{x}_i) = \begin{cases} 1 & \vec{w}^t\vec{x}_i + b > 0, \quad y_i = 1 \\ -1 & \vec{w}^t\vec{x}_i + b < 0, \quad y_i = -1 \end{cases}$$

$$\implies f(x_i) = \text{sign}(\vec{w}^t\vec{x}_i + b) = \begin{cases} 1 & y_i = 1 \\ -1 & y_i = -1 \end{cases}$$

Dicho hiperplano es una **superficie de decisión** y debe construirse de tal forma que la **separación de las dos clases sea máxima** (principio de generalización).

1. Definir un margen alrededor de dicho hiperplano e intentar maximizar el margen.
2. El hiperplano que tenga el mayor margen es el mejor clasificador de los datos.
3. Los **vectores de soporte** son los patrones de entrenamiento que tocan el límite del margen.

Los hiperplanos de los márgenes son:

- Hiperplano positivo: $\vec{w}^t\vec{x} + b = 1$
- Hiperplano negativo: $\vec{w}^t\vec{x} + b = -1$.

Y por definición, los patrones estarán "más allá" de los márgenes:

- Patrones positivos: $\vec{w}^t\vec{x}_i + b \geq 1$.
- Patrones negativos: $\vec{w}^t\vec{x}_i + b \leq -1$.

4.1. Problemas linealmente separables

4.1.1. Formulación del problema

Sea T el conjunto de patrones $(\vec{x}_i, y_i)$, se dice que son **linealmente separables** si $\exists(\vec{w}, b)$ tal que las inecuaciones:

$$\begin{array}{lll} \vec{w}^t\vec{x}_i + b \geq 1 & y_i = 1 & \text{Casos positivos} \\ \vec{w}^t\vec{x}_i + b \leq -1 & y_i = -1 & \text{Casos negativos} \end{array} \qquad i = 1, ..., n$$

sean válidas para todos los elementos del conjunto T.

Se puede encontrar un único **hiperplano óptimo** para el cual el margen entre los puntos de entrenamiento de dos clases distintas es maximizado.

1. Las dos inecuaciones anteriores se pueden reformular a:

$$y_i(\vec{w}^t\vec{x}_i + b) \geq 1, \quad i = 1, ..., n$$

2. Los **vectores de soporte** cumplen:

$$\begin{array}{ll} \vec{w}^t\vec{x}_i + b = 1 & y_i = 1 \\ \vec{w}^t\vec{x}_i + b = -1 & y_i = -1 \end{array} \implies y_i(\vec{w}^t\vec{x}_i + b) = 1$$

3. Sea d el ancho del margen, que queremos maximizar:

$$\begin{array}{ll} \vec{w}^t\vec{x}_i + b \geq d/2 & y_i = 1 \\ \vec{w}^t\vec{x}_i + b \leq -d/2 & y_i = -1 \end{array} \implies y_i(\vec{w}^t\vec{x}_i + b) \geq d/2$$

4. Si $\vec{x}_i$ es vector de soporte, entonces $y_i(\vec{w}^t\vec{x}_i + b) = d/2$
5. La distancia de $\vec{x}_i$ al plano separador es: $r = \dfrac{\vec{w}^t\vec{x}_i + b}{\|\vec{w}\|}$
6. Un vector de soporte está en el límite del margen: $r = \dfrac{y_i(\vec{w}^t\vec{x}_i + b)}{\|\vec{w}\|} = \dfrac{1}{\|\vec{w}\|}$

Por tanto el **margen de separación a maximizar** es:

$$d = 2r = \frac{2}{\|\vec{w}\|}$$

4.1.2. Problema de optimización

$$\max : \quad d = \frac{2}{\|\vec{w}\|} \qquad \text{sujeto a:} \quad y_i(\vec{w}^t\vec{x}_i + b) \geq 1$$

$$\min : \quad d^{-1} = \frac{1}{2}\|\vec{w}\|^2 \quad \text{sujeto a:} \quad y_i(\vec{w}^t\vec{x}_i + b) \geq 1$$

- Optimización cuadrática sujeta a restricciones lineales.
- Para resolver este problema se usan multiplicadores de Lagrange y el problema dual.

El problema dual debe ser minimizado con respecto a $(\vec{w}, b)$ y maximizado con respecto a $\alpha_i \geq 0$.

$$L_P = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\vec{w}^t\vec{x}_i + b) - 1 \right)$$

1. Se halla el gradiente de L_p con respecto a $(\vec{w}, b)$ y se igual a 0:

$$\frac{\partial L_P}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\frac{\partial L_P}{\partial w} = 0 \implies \vec{w} = \sum_{i=1}^{n} \alpha_i y_i x_i$$

2. Se sustituye esta expresión en la ecuación anterior:

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \vec{x}_i^t\vec{x}_j$$

4.1.3. Formulación del Problema Dual

$$\max_{\alpha_i} : \quad L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j \; y_i y_j \; \vec{x}_i^t\vec{x}_j$$

sujeto a :

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \qquad \alpha_i \geq 0$$

La solución al problema primal sería:

$$\vec{w} = \sum_{i=1}^{n} \alpha_i y_i x_i, \qquad b = y_k - \vec{w}^t\vec{x}_k, \qquad \text{para } \alpha_k > 0$$

En la práctica, b se promedia para evitar problemas de redondeo:

$$b = \frac{1}{N_{VS}} \sum_{i=1}^{N_{VS}} (y_i - \vec{w}^t\vec{x}_i)$$

Y la función de clasificación es:

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i \ \vec{x}_i^t \vec{x} + b$$

- Si $f(\vec{x}) > 0$, se clasifica como positivo (1).
- Si $f(\vec{x}) < 0$, se clasifica como negativo (-1).
- $f(\vec{x})$ da un nivel de **confianza** sobre la pertenencia de ese patrón a una de las clases.

4.1.4. Teorema de Karush Kuhn Tucker

- Si los $\alpha_i \geq 0$, $\implies$ $\alpha_i \Big(y_i(\vec{w}^t \vec{x}_i + b) - 1 \Big) = 0$
- Si $\alpha_i \neq 0$, $\implies$ $y_i(\vec{w}^t \vec{x}_i + b) = 1$ y por tanto $\vec{x}_i$ es un vector de soporte.

4.2. Problemas no linealmente separables

Se añaden las variables ξ_i para permitir instancias incorrectamente clasificadas. Intuitivamente, este parámetro es la "distancia" que tiene que recorrer un patrón mal clasificado para llegar al margen correspondiente de la clasificación.

4.2.1. Formulación matemática

$$\min_{(\vec{w},b)} : \quad d^{-1} = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{l} \xi_i$$

sujeto a :

$$\begin{aligned} y_i(\vec{w}^t \vec{x}_i + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

Si C es un valor **alto**:

- Más importancia a minimizar los errores ξ_i.
- Menos importancia a maximizar el margen.
- Resultado: **sobreajuste**.

Si C es un valor **bajo**:

- Más importancia a maximizar el margen.
- Menos importanca a minimizar los errores ξ_i.
- Resultado: menos sobreajuste, más **generalización**.

El **Problema Dual** tiene la forma:

$$L_P = \frac{1}{2}\|\vec{w}\|^2 - C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\Big(y_i(\vec{w}^t\vec{x}_i + b) - 1 + \xi_i\Big) - \sum_{i=1}^{n}\beta_i\xi_i$$

y debe ser minimizado con respecto a $(\vec{w}, b, \xi)$ y maximizado con respecto a (α_i, β_i).

1. Se halla el gradiente de L_P con respecto a $(\vec{w}, b, \xi)$ y se iguala a 0:

$$\begin{cases} \dfrac{\partial L_P}{\partial b} = 0 \implies \displaystyle\sum_{i=1}^{n}\alpha_i y_i = 0 \\ \dfrac{\partial L_P}{\partial \vec{w}} = 0 \implies \vec{w} = \displaystyle\sum_{i=1}^{n}\alpha_i y_i \vec{x}_i \\ \dfrac{\partial L_p}{\partial \xi} = 0 \implies \alpha_i + \beta_i = C \end{cases}$$

2. Sustituyendo en la expresión anterior:

$$L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\ y_i y_j\ \vec{x}_i^t\vec{x}_j$$

4.2.2. Formulación del Problema Dual

$$\max_{\alpha_i}: \quad L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\ y_i y_j\ \vec{x}_i^t\vec{x}_j$$

sujeto a :

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

$$0 < \alpha_i < C, \qquad i = 1, ..., n$$

Una vez obtenidos los α_i, la solución viene dada por:

$$\vec{w} = \sum_{i=1}^{n}\alpha_i y_i \vec{x}_i, \qquad b = y_k(1 - \xi_k) - \vec{w}^t\vec{x}_k, \quad \alpha_k > 0$$

Y la función de clasificación viene dada por:

$$f(\vec{x}) = \sum_{i=1}^{n}\alpha_i\ y_i\ \vec{x}_i^t x + b$$

4.2.3. Teorema de Kuhn-Tucker

Como $y_i(\vec{w}^t\vec{x}_i + b) \geq 1 - \xi_i$, si $\alpha_i \geq 0$, entonces:

$$\alpha_i\Big(y_i(\vec{w}^t\vec{x}_i + b) - 1 + \xi_i\Big) = 0, \qquad \xi_i(C - \alpha_i) = 0$$

Si $\alpha_i > 0$, el patrón $\vec{x}_i$ es un **vector de soporte**: $y_i(\vec{w}^t\vec{x}_i + b) = 1 - \xi_i$

- Si $0 < \alpha_i < C$, entonces $\xi_i = 0$ y $\vec{x}_i$ está en el límite del margen.
- Si $\alpha_i = C$, entonces $\xi_i \neq 0$:
  - $\vec{x}_i$ es un error de clasificación, pero define el margen.
  - $\vec{x}_i + \xi_i$ sí está en el margen.

4.3. SVM no lineales

Idea: Proyectar cada patrón por medio de una transformación:

$$\phi : \vec{x} \longrightarrow \phi(\vec{x})$$

a un nuevo espacio de mayor dimensionalidad de tal manera que en ese nuevo espacio los patrones sean linealmente separables o linealmente separables tolerando cierto error.

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i \ \phi(\vec{x}_i)^t\phi(\vec{x}) + b$$

4.3.1. Función Kernel

El **truco del kernel** se basa en definir la **función kernel**:

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^t\phi(\vec{x}_j)$$

que es equivalente a $\phi(\vec{x}_i)^t\phi(\vec{x})$ a partir de los vectores $(\vec{x}_i, \vec{x}_j)$ en el espacio original.

- No es necesario obtener $\phi : \vec{x} \longrightarrow \phi(\vec{x})$
- No es necesario calcular las proyecciones en el nuevo espacio.
- La función kernel implícitamente mapea los datos a un espacio de mayor dimensionalidad.

4.3.2. Métodos basados en *kernels*

Teorema de Mercer: Todas las funciones simétricas definidas semipositivas son kernels.

Las funciones simétricas definidas semipositivas se corresponden con una matriz de Gram simétrica semidefinida positiva:

$$K = \begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{pmatrix}$$

Por tanto, las funciones de kernel deben ser:

1. Continuas.
2. Simétricas.
3. Preferiblemente, tener una matriz de Gram semidefinida positiva.

Los kernels que satisfacen el teorema de Mercer son **semidefinidos positivos**:

1. Sus matrices de kernel no tienen valores propios no negativos.
2. Garantizan que el problema de optimización convergirá y la solución será única.

Aunque muchas funciones de kernel no son estrictamente definidas positivas, han demostrado tener un buen comportamiento en distintos problemas.

4.3.3. Selección del Kernel

- Kernel polinómico: Modelizar conjunciones de valores hasta el orden del polinomio.
- Función de base radial (*radial basis function*): Construir hiperesferas.
- Kernel lineal: Construir hiperplanos.

Algunos de los kernels más usados son:

- **Kernel Lineal** (equivalente a no-kernel): $K(x, y) = x^t y + c$
- **Kernel Polinómico**. $K(x, y) = (ax^t y + c)^d$
  - Son adecuados para problemas donde todos los datos de entrenamiento están normalizados.
- **Kernel Gaussiano** (base radial):

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

  - $\phi(\vec{x})$ es infinito dimensional, cada punto se mapea a una función gaussiana.
  - Si σ se sobreestima, la proyección perderá su poder no lineal.
  - Si σ se subestima, los límites de decisión se volverán demasiado sensibles al ruido en los datos de entrenamiento.
- **Kernel Exponencial** (base radial):

$$K(x, y) = \exp\left(-\frac{\|x - y\|}{2\sigma^2}\right)$$

- **Kernel Laplaciano** (base radial):

$$K(x,y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right)$$

  - Equivalente al exponencial, menos sensible a los cambios del parámetro σ.

- **Kernel Hiperbólico Tangente** (*kernel sigmoidal*, *kernel perceptrón multicapa*):

$$K(x,y) = \tanh(ax^t y + x)$$

  - Equivalnente a un perceptrón de dos capas.
  - $a \overset{\text{def}}{=}$ pendiente (comúnmente $1/N$, donde N es el número de dimensiones).

4.3.4. Formulación del problema

$$\min_{(\vec{w},b)} : \quad d^{-1} = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

sujeto a :

$$y_i\left(\vec{w}^t\phi(\vec{x}_i) + b\right) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

- ϕ realiza una transformación de los datos a un espacio de mayor dimensionalidad.
- w es un vector en el nuevo espacio de mayor dimensionalidad.

El **Problema Dual** se transformaría en encontrar $\vec{\alpha}$ tal que:

$$\max_{\alpha_i} : \quad L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\ y_i y_j\ K(x_i,\vec{x}_j)$$

sujeto a :

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

$$0 < \alpha_i < C, \qquad i = 1,...,n$$

Y la función de decisión quedaría como:

$$f(\vec{x}) = \sum_{i=1}^{n}\alpha_i\ y_i\ K(\vec{x}_i,\vec{x}) + b$$

4.4. SVM para problemas de regresión

El planteamiento para un problema de regresión es encontrar un hiperplano $\vec{w}^t\vec{x} + b$ que aproxime mejor los patrones $\vec{x}_i$, para $i = 1, ..., n$.

$$\begin{aligned} \text{mín} : \quad & z = \frac{1}{2}\|\vec{w}\|^2 \\ \text{sujeto a :} \quad & \\ & y_i - \vec{w}^t\vec{x}_i - b \leq \varepsilon \\ & \vec{w}^t\vec{x}_i + b - y_i \leq \varepsilon \end{aligned}$$

Tolerando cierto error (ξ_i y ξ_i^*):

$$\begin{aligned} \text{mín} : \quad & z = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{sujeto a :} \quad & \\ & y_i - \vec{w}^t\vec{x}_i - b \leq \varepsilon + \xi_i \quad \text{(exceso)} \\ & \vec{w}^t\vec{x}_i + b - y_i \leq \varepsilon + \xi_i^* \quad \text{(defecto)} \\ & \xi_i, \xi_i^* \geq 0 \end{aligned}$$

Aplicando multiplicadores de Lagrange, obtenemos una nueva función objetivo:

$$\begin{aligned} L = & \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ & - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \vec{w}^t\vec{x}_i + b) - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* - y_i + \vec{w}^t\vec{x}_i + b) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*) \end{aligned}$$

La ecuación de regresión resultante es:

$$y(\vec{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle\vec{x}_i, \vec{x}\rangle + b$$

4.4.1. SVMs para regresión no lineal

La solución viene dada por proyectar el conjunto de patrones a otro espacio donde sí se puedan aproximar con un hiperplano.

$$y(\vec{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle\phi(\vec{x}_i), \phi(\vec{x})\rangle + b\rangle$$

Para el caso general:

$$y(\vec{x}) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(\vec{x}_i, \vec{x}) + b$$

4.5. Conclusiones de las SVM

- Las SVM son clasificadores para 2 clases.
- Se puede cambiar la formulación del algoritmo QP para permitir clasificación multiclase.
- El modelado de la SVM no necesita de la totalidad de puntos disponibles para hallar una solución al problema de maximización de la separación entre clases.

Ventajas:

- Entrenamiento fácil.
- No hay óptimo loca, sino óptimo global.
- Escalamiento aceptable para datos en espacios dimensionales altos.
- El error puede ser controlado explícitamente.
- Datos como cadenas de caracteres y árboles pueden ser usados como entrada a la SVM.

Debilidades:

- Se necesita una "buena" función kernel.
- Ajustar todos los parámetros de las SVM es un proceso laborioso.
- Prueba y error.

Tema 5: Árboles de Decisión

Los **árboles de decisión** son una representación de los procesos de decisión involucrados en las tareas de clasificación. En su estructura diferenciamos:

- Hojas. Representan la etiqueta asociada a una clasificación.
- Nodos. Representan la pregunta acerca de un cierto atributo.
- Ramas de un nodo. Representan los diferentes valores que puede tomar un atributo respecto a lo que se pregunta en el nodo.

En el caso de clasificación binaria, el árbol representa una **disyunción de conjunciones** de restricciones sobre los valores de los atributos de las instancias.

- **Camino**: conjunción de tests de atributos.
- **Árbol**: disyunción de esas conjunciones.
- Del árbol se pueden generar una serie de **reglas** y **división de patrones**.

Generación de un árbol: Dado un conjunto de ejemplos clasificados (*aprendizaje supervisado*), el árbol óptimo es aquel que permita describir la clasificación con el menor número de cuestiones posibles.

5.1. ID3 (*Iterative Dichotomiser 3*)

El algoritmo ID3 de generación de árboles de decisión se basa en construir el árbol de arriba a abajo:

1. Se selecciona el *mejor atributo* como raíz.
2. Se abre el árbol para cada valor del atributo.
3. Según los valores de ese atributo seleccionado, los patrones se clasifican y colocan en cada rama, "partiendo" la base de datos.

Este proceso debe ser repetido iterativamente hasta que el árbol clasifique correctamente los ejemplos o se hayan usado todos los atributos. Los nodos hoja deben ser etiquetados con la clase de los ejemplos.

5.1.1. Instrucciones del algoritmo

A la hora de crear un nodo con un conjunto de instancias:

- Si todas las instancias de ese nodo son de la misma clase, entonces crear una hoja etiquetada con dicha clase.
- Si el conjunto de instancias está vacío, crear una hoja etiquetada con la clase por defecto.
- Si el conjunto de instancias no contiene ningún atributo, crear una hoja etiquetada con la clase mayoritaria.

En otro caso:

1. Elegir el atributo A con mayor ganancia de información y crear un nodo con el atributo seleccionado.
2. Para cada valor V de dicho atributo:
   - Crear una rama con el valor V.
   - Seleccionar las instancias con el valor V del atributo A.
   - Eliminar el atributo A de este conjunto de instancias.
   - Repetir el proceso con el conjunto de instancias.

5.1.2. Selección del mejor atributo

- C: conjunto de clases.
- n: número de patrones.
- n_c: número de patrones de la clase $c \in C$.
- A: conjunto de atributos.
- A_i: atributo i de A.
- V_i: conjunto de valores de A_i.
- n_{ij}: número de patrones que en el atributo A_i toma el valor $j \in V_i$.
- n_{ijc}: número de patrones que en el atributo A_i toma el valor $j \in V_i$ y pertenecen a la clase $c \in C$.

La **entropía** de un conjunto de patrones se define como:

$$I = -\sum_{c \in C} \frac{n_c}{n} \log_2 \left(\frac{n_c}{n} \right)$$

- Si $I = 0$, certidumbre total, todos los patrones son de la misma clase.
- Si $I = 1$, incertidumbre total, mismo número de patrones de cada clase.

La entropía del valor j del atributo A_i se define como:

$$I_{ij} = -\sum_{c \in C} \frac{n_{ijc}}{n_{ij}} \log_2 \left(\frac{n_{ijc}}{n_{ij}} \right)$$

La entropía del atributo A_i se define como:

$$I(A_i) = \sum_{j \in V_i} \frac{n_{ij}}{n} I_{ij}$$

- Es la media ponderada de las entropías para cada uno de los valores V_i de A_i.
- Determina el nivel de incertidumbre que se tiene al separar la base de datos con todos los valores de ese atributo.

La **ganancia de información** del atributo A_i se define como:

$$G(A_i) = I - I(A_i)$$

En el algoritmo ID3 es de vital importancia obtener el atributo que mejor separe los ejemplos (el que tenga mayor ganancia de información) con el objetivo de reducir la entropía (*incertidumbre*) en la partición de los patrones.

Para ello se escoge el **atributo con menor entropía**. Dicho atributo indica que al separar los patrones con el mismo, cada uno de los conjuntos resultantes tendrá más certidumbre en los datos.

5.1.3. ¿Cuánto hacer crecer el árbol?

Si se hace crecer el árbol hasta clasificar correctamente todos los ejemplos de entrenamiento, se produce el efecto de **sobreajuste/sobreentrenamiento**.

- Si hay ruido en los ejemplos, se aprende el ruido.
- Si hay pocos ejemplos asociados a los nodos hoja, no son representativos de la verdadera función.
- ID3 produce árboles que sobreajustan los ejemplos de entrenamiento y no funcionan adecuadamente con nuevos ejemplos.
- No es un modo capaz de generalizar.
- Soluciones: **pre-poda** y **post-poda**.

5.1.4. Pre-poda

- Detener el crecimiento del árbol antes de que alcance el punto en el que clasifica todos los ejemplos de entrenamiento.
- Aplicar un test estadístico para estimar si expandiendo un nodo particular es probable producir una mejora más allá del conjunto de entrenamiento.

5.1.5. Post-poda

1. Permitir que se genere un árbol sobreajustado (mayor coste computacional).
2. Comenzando desde abajo, y de forma iterativa, examinar los subárboles de los nodos no terminales.

3. Escoger el nodo cuya poda implique **mayor mejora** (o igualación) en los resultados de test respecto al árbol original.
4. El proceso de poda en un nodo conlleva:
   - Eliminar el subárbol correspondiente con raíz en ese nodo.
   - Convertir el nodo no terminal en nodo hoja.
   - Asignarle la clasificación más común de los patrones asociados a ese nodo.

5.1.6. ¿Es adecuada la medida de selección de los atributos?

El algoritmo ID3 favorece la elección de atributos con muchos valores, pues dan una ganancia mayor. En contrapartida, el tener tantas ramificaciones obliga a separar los ejemplos en conjuntos muy pequeños, y como resultado, el modelo de clasificaicón será muy malo con patrones de test.

El **ratio de ganancia** es una medida para evaluar la ganancia de información que penaliza los atributos con muchos valores y uniformemente distribuidos:

$$\text{Ratio}(A_i) = \frac{G(A_i)}{H(A_i)}$$

donde $H(A_i)$ es la entropía con respecto a los valores de A_i (no respecto a las clases):

$$H(A_i) = \sum_{j \in V_i} \frac{n_{ij}}{n} \log_2 \left(\frac{n_{ij}}{n} \right)$$

- No tiene en cuenta las clases, solamente los diferentes valores que toma.
- Cuando un atributo tiene muchos valores distintos, tiene una entropía grande, con lo que se divide por un valor mayor.
- Favorece aquellos atributos que, en igualdad de ganancia, separen los datos en menos subconjuntos.

5.1.7. ¿Cómo manejar los atributos *missing*?

A la hora de hacer crecer el árbol:

- Eliminar instancias completas.
- Estimar dichos valores (imputación):
  - Asignar la **moda**.
  - Asignar a cada posible valor una probabilidad de acuerdo con el resto de patrones, y repartir el ejemplo en cada uno de sus valores de acuerdo con dicha probabilidad.

A la hora de clasificar un nuevo caso:

- Los casos con valores desconocidos se clasifican de acuerdo con la mayor probabilidad que proporcione el árbol.

5.1.8. ¿Cómo manejar atributos con costes diferentes?

Introduciendo el coste en la medida de selección de atributos:

$$\frac{G(A_i)}{\text{Coste}(A_i)}, \qquad \frac{2^{G(A_j)}-1}{(\text{Coste}(A_j)+1)^w}$$

donde $w \in [0,1]$ pondera la importancia del coste frente a la ganancia.

Mediante esta nueva medida de selección de atributos se trata de situar los atributos de bajo coste arriba del árbol, acudiendo a los de alto coste sólo cuando sea necesario mejorar la clasificación.

5.1.9. ¿Cómo manejar atributos continuos?

Para manejar atributos continuos, se **discretizan**:

1. Se particionan en un conjunto discreto de intervalos.
2. Se buscan aquellos intervalos que produzcan la mayor ganancia de información.

El proceso es el siguiente:

1. Ordenar los ejemplos de acuerdo al valor continuo de A.
2. Tomar cada par de valores contiguos de A con distinto valor de C (etiqueta de clase) y hallar los promedios, que serán los candidatos a ser ℓ (límites de los intervalos).
3. Entre estos candidatos a ser ℓ, escoger el que produzca mayor cantidad de información.
4. Una vez escogido el ℓ, este nuevo atributo compite con los otros discretos para ver si es escogido.

Otra alternativa es usar combinaciones lineales de varios atributos, permitiendo que una condición sea una combinación lineal de atributos.

5.1.10. C4.5

- Algoritmo basado en ID3.
- Permite trabajar con valores continuos para los atributos.
- Escoge atributos usando ratio de ganancia.
- Incorpora postpoda de reglas.
- Introduce costes en los atributos.
- Maneja valores desconocidos.
- Ordena las reglas de clasificación por su error estimado (menor a mayor).

5.2. CART (*Classification and Regression Trees*)

1. Construcción de un árbol.
2. Parada del proceso de crecimiento cuando se constituye un árbol máximo que sobreajusta la información contenida en la base de datos.
3. Podado del árbol, haciéndolo más sencillo y dejando sólo los nodos más importantes.
4. Selección del árbol óptimo con capacidad de generalización.

5.2.1. Selección de atributos

- Se busca una medida de su *pureza*.
- Un conjunto de nodos es más puro cuando las instancais están menos repartidas entre las clases (menor entropía, mayor cerditumbre).
- La función de partición debe asegurar que la pureza en los nodos hijos sea la máxima. Es decir, que las instancias en los hijos tengan más homogeneidad de clases posible.

La **medida de impureza** debe ser máxima cuando las instancias de un nodo están divididas equitativamente entre todas las clases, y debe ser 0 cuando las instancias de un nodo son de la misma clase. Para media la impureza se puede usar:

- Tasa de instancias mal clasificadas (no usar). Se clasifican todas las instancias que llegan a un nodo con la clase mayoritaria en dicho conjunto de instancias. Después se computa la tasa de fallo.
- Entropía.
- Índice de Gini.

5.2.2. Índice de Gini

El **índice de Gini** es una medida de cuántas veces un elemento escogido aleatoriamente es etiquetado incorrectamente si fuera etiquetado aleatoriamente de acuerdo con la distribución de etiquetas en el subconjunto.

Se calcula sumando la probabilidad de escoger cada patrón por la probabilidad de equivocarse etiquetándolo de acuerdo a la distribución de etiquetas del conjunto.

Para el nodo p, el índice de Gini se define como:

$$i(p) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m}(f_i - f_i^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2$$

$$i(p) = 1 - \sum_{i=1}^{m} f_i^2$$

donde f_i es la proporción de instancias de la clase i, para $i = 1, ..., m$.

Sea un nodo a que se divide en dos partes: a_L y a_D generando dos subconjuntos con n_{aL} y n_{aL} patrones respectivamente. La **reducción de la impureza** al realizar esta división es:

$$\Delta(a) = i(a) - \frac{n_{aL}}{n_a} i(a_L) - \frac{n_{aD}}{n_a} i(a_D)$$

- La reducción de la impureza es siempre positiva.
- Se escoge la división que da la máxima reducción de impureza, decrementando así el índice de Gini.

La **impureza de un árbol** es la suma de la impureza de todos los nodos terminales por la proporción de casos que llegan a ese nodo en el árbol.

Si se tiene un árbol A con un nodo terminal a que se puede dividir en dos nodos a_L y a_D, la impureza del nuevo árbol A' al dividir el nodo a será:

$$i(A') = \underbrace{i(A)}_{\substack{\text{Impureza} \\ \text{del árbol}}} + \underbrace{\frac{n_{aL}}{n} i(a_L) + \frac{n_{aD}}{n} i(a_D)}_{\substack{\text{Impureza relativa a} \\ \text{los nodos terminales}}} \underbrace{- \frac{n_a}{n} i(a)}_{\substack{\text{Impureza relativa al} \\ \text{antiguo nodo terminal } a}}$$

Y la **reducción de la impureza** será:

$$i(A) - i(A') = \frac{n_a}{n} \Delta(a)$$

Es decir, escoger la división que maximiza la reducción de impureza del árbol es equivalente a escoger la división que maximice la reducción de impureza del nodo.

5.2.3. Crecimiento del árbol

El crecimiento del árbol continúa por los nodos terminales hasta que uno de ellos:

- Es puro (sólo hay una clase).
- No hay divisiones admisibles.
- Se puede añadir una condición sobre el tamaño mínimo del nodo.
- Se ha fijado un límite externo de la profundidad de crecimiento del árbol.

5.2.4. Poda del árbol

El árbol generado con el algoritmo CART clasifica correctamente los patrones de entrenamiento, pero produce un sobreajuste que provoca pérdida de capacidad de generalización.

Para la poda se usa una medida de **coste-complejidad** que combina los criterios de **precisión** frente a **complejidad** en el número de nodos y **velocidad** de procesamiento, buscando el árbol que obtenga menor valor en este parámetro. Los árboles más sencillos aseguran una mayor capacidad de generalización.

La propuesta es podar sucesivamente A_{max} de tal forma que entre los posibles árboles podados con el mismo número de hojas (misma complejidad) se selecciona el de menor error de test (menor coste).

- Se necesita una muestra test.
- Se suprimen las ramas menos informativas de tamaño grande (aumenta poco el coste y se reduce mucho la complejidad).

Para conseguir este objetivo (escoger el árbol de menor coste en la muestra test) hay varios métodos. Entre ellos el más común es **validación cruzada**.

5.3. MARS (*Multivariate Adaptive Regression Splines*)

El algoritmo MARS es una generalización continua de CART y sus funciones base son un producto de funciones *splines*.

- Una variable puede ser elegida varias veces.
- Los test en los nodos pueden ser multivariantes (con combinaciones lineales de variables).

5.4. Aplicaciones

Los árboles de decisión son apropiados para problemas donde:

- Las instancias están representadas como pares atributo-valor (atributos discretos o reales).
- Se necesita robustez a errores en los datos de entrenamiento (tanto en los atributos como en la variable respuesta).
- Puede haber datos perdidos (*missing*) en los atributos de algunas instancias.
- Los dominios son complejos y no existe una clara separación lineal entre clases.
- La función objetivo toma valores discretos.

Tema 6: Reglas de Clasificación

Objetivo. Dado un conjunto de ejemplos positivos y negativos, generar un conjunto de reglas que describan todos los positivos y no reconozcan ningún ejemplo negativo.

6.1. Algoritmo AQ

6.1.1. Elementos del algoritmo

- A: conjunto de atributos.
- V: conjunto de valores de los atributos.
- C: conjunto de clases.
- E: conjunto de ejemplos de entrenamiento.

LEF: Lista de criterios de preferencia de reglas. Es una función de evaluación lexicográfica que permite elegir la regla a añadir entre un conjunto de candidatas siguiendo un criterio determinado:

- **Cobertura**. Número de ejemplos positivos cubiertos.
- **Simplicidad**. Número de atributos que se estudian en el antecedente.
- **Coste**. Coste de evaluación del antecedente.
- **Generalidad**. Número de ejemplos observados entre el número de ejemplos posibles.

Selector: Permite realizar preguntas sobre un atributo.

- Sintaxis: `Atributo - Operador - Valores`.
- Operadores: $=$, $>$, $<$, $\geq$, $\leq$, $\neq$.
- Valores: valor continuo o discreto del atributo.

Complejo. Conjunción de selectores (antecedente de una regla de clasificación).

Recubrimiento: Disyunción de complejos. Permite describir conjuntos de reglas.

6.1.2. Funcionamiento del algoritmo

1. El conjunto de reglas (*recubrimiento*) está vacío.
2. Se considera el conjunto P de ejemplos positivos y el conjunto N de ejemplos negativos.
3. Mientras queden ejemplos positivos en P, repetir:

- Elegir un ejemplo de P (semilla).
- Generar complejos que cubran la semilla y excluyan a los ejemplos N (algoritmo star).
- Elegir de entre los complejos, el que optimice el criterio de selección LEF.
- Añadir el complejo elegido al recubrimiento.
- Eliminar de P todos los ejemplos cubiertos por la nueva regla.

4. El algoritmo AQ dice cuándo un ejemplo nuevo es positivo, no a la inversa.
5. No importa el orden de aplicación de las reglas.

6.2. Algoritmo CN2

El algoritmo CN2 genera una lista ordenada de reglas (el orden influye en la clasificación) donde la última regla es la regla por defecto.

Para la construcción de las relgas se utilizan todso los selectores posibles y se seleccionan atendiendo a la entropía de distribución y a la significancia estadística de los complejos estudiados.

6.3. Otros algoritmos

- REP (*Reduced Error Pruning*)
- IREP (*Incremental Reduced Error Pruning*)
- IREP*.
- RIPPER
- SLIPPER.

Tema 7: Regresión

La regresión es un método que modela la relación entre una variable dependiente Y y un conjunto de variables independientes X_i.

Objetivo. Minimizar el error entre los valores de Y que se intentan aproximar con $f(\vec{x})$ y el valor de la aproximación $f'(\vec{x})$.

Medida de error: Suma del error cuadrático medio sobre el conjunto de entrenamiento T_{train}:

$$\text{ECM} = \frac{1}{2} \sum_{\vec{x} \in T_{\text{train}}} (f(\vec{x}) - f'(\vec{x}))^2$$

7.1. Regresión lineal

En regresión lineal, se intenta aproximar la función $f(\vec{x})$ (supuestamente lineal) con una función lineal $f'(\vec{x})$:

$$f'(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

Es decir, se intenta definir el vector de pesos $w \in \mathbb{R}^m$ que minimice la función de error ECM. Para ello se conoce el método estadístico de mínimos cuadrados.

No obstante, el modelo de regresión lineal es un modelo global que calcula una ecuación aplicable en todo el espacio. Si los datos interactúan de forma no lineal y complicada, este modelo global no es adecuado.

Alternativa: Crear un nuevo modelo que particione el espacio de forma recursiva en regiones más pequeñas. En cada región las interacciones serán más manejables y será más sencillo crear en cada un **modelo local** de regresión lineal.

7.2. Árboles de regresión

Los **árboles de regresión** son una generalización de los árboles de decisión para particionar el espacio en regiones y aproximar la función objetivo mediante una función a trozos.

- Es decir, el modelo de cada región es una constante.
- Sea T_h el conjunto de n_h instancias $(\vec{x}_i, y_i)$ que llega a un nodo hoja h cualquiera. El modelo de esa hoja será el valor promedio de los valores de y que lleguen a ese nodo.

$$\frac{1}{n_h} \sum_{i=1}^{n_h} y_i$$

7.2.1. Algoritmo CART

- En vez de usar el índice de Gini como criterio de impureza para separar regiones, se utiliza la **suma de cuadrados**.
- Se escogen las divisiones que causan la mayor reducción en la suma de cuadrados.
- Al podar el árbol, la medida utilizada es el ECM en las predicciones hechas por el árbol.

7.2.2. Ventajas

- Cálculo rápido de predicciones.
- Es fácil comprender qué variables son importantes a la hora de realizar la predicción.
- Robustez ante datos perdidos. Siempre se continúa descendiendo en el árbol y la predicción se obtiene promediando todas las hojas del subárbol al que se llega.
- El modelo da una **respuesta escalonada** que se puede aplicar cuando la superficie de regresión deseada es suave. Si no es suave, la superficie constante a trozos puede aproximarla arbitrariamente bien con un número suficiente de hojas.
- Algoritmos rápidos y eficientes para desarrollar árboles de regresión.

7.3. Árboles de modelos de regresión

Los árboles de modelos de regresión son un tipo de árbol de decisión donde las hojas contienen un **modelo de regresión lineal** para predecir la respuesta de las instancias que llegan a esa hoja.

7.3.1. Algoritmo M5 y M5'

El algoritmo M5 genera árboles de decisión similares a los de ID3. Escoge el atributo que minimiza la variación en los valores de la clase que resulta por cada rama.

M5' introduce mejoras para:

- Tratar atributos discretos.
- Tratar valores perdidos.
- Reducir el tamaño del árbol.

Fase 1. Construir el árbol

Sea $S = \{(\vec{x}_1, y_1), ..., (\vec{x}_s, y_s)\}$ el conjunto de ejemplos que llega a un nodo (inicialmente, el conjunto de entrenamiento):

- Sea $\sigma(S)$ la desviación típica de los valores que llegan a dicho nodo, esta se trata como **medida de error** de ese nodo, pues se desea que los valores de la clase sean los más parecidos posible.
- Sea S_{Ai} el conjunto de ejemplos que resulta de **ramificar un nodo** de acuerdo al atributo A con el valor i-ésimo, para $i = 1, ..., n_A$.
- $\sigma(S_{Ai})$ marca el error que se cometería al ramificar un nodo de acuerdo al valor i-ésimo del atributo correspondiente A.

La reducción del error al ramificar un nodo con el atributo A es:

$$\Delta\text{Error} = \sigma(S) - \sum_{i=1}^{n_A} \frac{|S_{Ai}|}{|S|}\sigma(S_{Ai})$$

- Si $\sigma(S_{A1}), ..., \sigma(S_{An_A})$ son valores pequeños, ΔError será un valor alto, muy cercano a $\sigma(S)$.
- Se debe escoger el atributo que maximice dicha reducción, que reduzca la dispersión al bajar de nivel.

Este proceso de construcción se debe detener cuando:

- Quedan pocos ejemplos en los nodos terminales.
- Hay poca variación de los valores de la clase. Es decir, su desviación típica es una pequeña fracción de la desviación típica del conjunto total.

Faso 2. Estimar el error

Residual: Diferencia entre el valor real de una instancia y el valor predicho por el modelo.

M5 estima el error de un modelo con el conjunto de entrenamiento T (donde $|T| = n$) de la siguiente forma:

$$\text{Error} = \frac{1}{n}\sum_{i=1}^{n} |y_i - f(x_i)|$$

Generalmente, este cálculo infraestima el error en casos no vistos. Se puede compensar multiplicando por el factor:

$$\frac{n+v}{n-v}$$

donde v es el número de atributos en el modelo del nodo.

- Si n bajo y v alto $\Longrightarrow (n+v)/(n-v)$ alto:
  - El modelo caracteriza pocas instancias con muchos atributos.
  - Particulariza más en esas instancias y generaliza menos (sobreajuste).
  - E residual de test se aleja del de entrenamiento.
- Si n alto y v bajo $\Longrightarrow (n+v)/(n-v)$ bajo:
  - El modelo caracteriza muchas instancias con pocos atributos.
  - Abstracción, generalización.
  - El error de test se acerca al de entrenamiento.

Fase 3. Construcción/Simplificación de modelos lineales en los nodos intermedios

1. En cada uno de los nodos no terminales y nodos hoja se construye un modelo lineal multivariante usando técnicas estándar de regresión.
   - Los atributos a usar en el modelo de un nodo concreto se restringen a aquellos que se referencian en los tests o en los modelos del subárbol creado a partir de dicho nodo.
2. Cada uno de estos modelos lineales se **simplifica** eliminando parámetros para minimizar su error estimado.
   - En general, eliminar parámetros conlleva al aumento del error.
   - Sin embargo, se reduce el factor $(n+v)/(n-v)$.

Fase 4. Podar nodos

Una vez generado un modelo en cada nodo interior, se aplica una poda desde las hojas siempre que el **error estimado decrezca**.

Dado un nodo ya simplificado, se calcula el error de su modelo y se compara con el error del subárbol que cuelga de él (que es la ponderación de los errores de los modelos de cada rama).

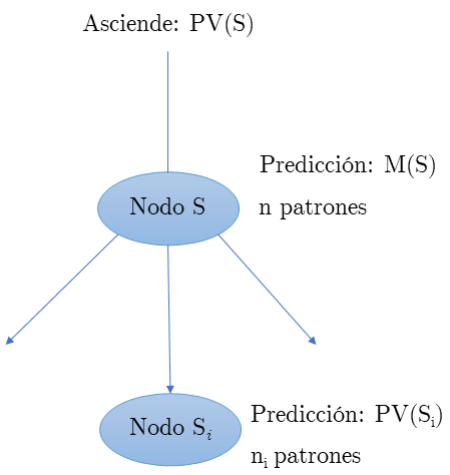
Se poda dicho nodo (y se convierte en nodo hoja) si su error es menor que el de su subárbol.

Fase 5. Suavizar las predicciones

Una vez podado el árbol, en los modelos linelaes adyacentes de las hojas puede haber grandes discontinuidades (valores predichos muy distintos). El proceso de suavización se realiza desde las hojas hasta la raíz:

1. Predecir el valor de la instancia en la hoja.
2. Pasar la predicción al siguiente nodo hacia arriba.
3. Sea:
   - S_i: rama del nodo S por la que llegó dicha predicción.
   - n_i: nº de casos de entrenamiento en S_i.
   - PV(S_i): valor predicho en S_i.
   - M(S): valor dado por el modelo en S.
   - k: constante por defecto.

   Entonces el valor predicho subido a S es:



$$\text{PV}(S) = \frac{n_i \text{PV}(S_i) + k\text{M}(S)}{n_i + k}$$

4. Se suaviza así el valor en cada nodo hasta la raíz, combinándolo con el valor que predice el nodo.

7.3.2. Atributos discretos/categóricos

Se pueden tratar transformándolos primero en varias variables binarias. Antes de empezar a construir el árbol, para cada atributo discreto A:

1. Para cada valor $v_1, ..., v_{k_A}$ que puede tomar el atributo A, promediar los valores de las salidas (y) correspondientes.
2. Si A toma k_A valores, reemplazar A por $k_A - 1$ variables binarias.
3. Para cada instancia, la i-ésima variable binaria vale 1 si su valor en A es uno de los i primeros y vale 0 en otro caso.

7.3.3. Salidas categóricas (clasificación)

Si sólo hay dos categorías para las salidas deseadas (0 o 1), el árbol dará un valor continuo que pasará por un umbral 0.5.

Si hay más de dos categorías, se cambia al salida por un conjunto de variables, tantas como categorías de salidas.

- Cada variable indica pertenencia (1) o no pertenencia (0) a su categoría correspondiente.
- Cada una de estas variables se utiliza como salida para un árbol distinto. Es decir, se entrena un árbol por salida.
- Para evaluar un patrón, se introduce en cada árbol. Aquel que de una valor de salida mayor, es el que asigna la categoría final de la salida.

Tema 8: Aprendizaje basado en Instancias

La idea del **Aprendizaje basado en Instancias** (IBL, *Instance-Based Learning*) se basa en la idea de crear un **clasificadodr perezoso** (*lazy*) que para clasificar una instancia use las que más se parecen del conjunto que conoce.

1. **Entrenamiento**. No se realiza ningún cómputo de entreno, sino que se almacena todo el conjunto de datos disponibles.

2. **Generalización**. Dado un nuevo dato, se extraen de memoria el conjunto de datos similares para clasificarlo.

Ventajas:

- Para cada nueva instancia puedo botener un clasificador diferente.
- La descripción de las instancias puede ser tan compleja como quiera.

Desventajas:

- El coste de clasificación puede ser alto.
- Atributos irrelevantes pueden afectar la medida de similitud.
- Necesidad de una métrica apropiada.
- Almacenamiento.

8.1. Algoritmo k-NN (k-Nearest Neighbour)

Idea. Un nuevo caso se clasifica en la clase más frecuente a la que pertenecen sus k vecinos más cercanos. Las fases de inducción del modelo + deducción se colapsan (**trasinducción**).

Sea $(\vec{x}_i, y_i)$ una instancia del conjunto T_{train} (normalmente normalizadas), donde $\vec{x}_i \in \mathbb{R}^m$ y $y_i \in C$ (conjunto de clases), para $i = 1, ..., n$.

Se puede calcular distancia entre dos instancais por medio de la distancia euclídea:

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{r=1}^{m} (x_{ir} - x_{jr})^2}, \qquad \vec{x}_i, \vec{x}_j \in \mathbb{R}^m$$

8.1.1. Otras medidas de distancias

- Distancia de Manhattan:
$$d(\vec{x}_i, \vec{x}_j) = \sum_{r=1}^{m} |x_{ir} - x_{jr}|$$
- Distancia de Minkowski:
$$d(\vec{x}_i, \vec{x}_j) = \left(\sum_{r=1}^{m} |x_{ir} - x_{jr}|^q \right)^{1/q}$$
- Distancia coseno:
$$d(\vec{x}_i, \vec{x}_j) = 1 - \cos \angle(\vec{x}_i, \vec{x}_j)$$
- Correlación como distancia:
$$d(\vec{x}_i, \vec{x}_j) = 1 - \text{corr}(\vec{x}_i, \vec{x}_j)$$
- Distancia de Hamming (variables categóricas):
$$D_H(\vec{x}_i, \vec{x}_j) = \sum_{r=1}^{m} |x_{ir} - x_{jr}|$$
  - Si $\vec{x}_i = \vec{x}_j$, entonces $D_H(\vec{x}_i, \vec{x}_j) = 0$.
  - Si $\vec{x}_i \neq \vec{x}_j$, entonces $D_H(\vec{x}_i, \vec{x}_j) = 1$.

8.1.2. Función objetivo

Dada una nueva instancia $\vec{x}_0$ y sus k vecinos: $\vec{x}_1, ..., \vec{x}_k$ de los cuales se sabe su clase de pertenencia: $y_1, ..., y_k$. Se asigna a $\vec{x}_0$ la clase más común entre los k vecinos más cercanos:

$$f : \mathbb{R}^m \longrightarrow C \text{ (conjunto de clases)}$$

$$f(\vec{x}_0) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta(c, y_i), \qquad \text{donde } \delta(c, y) = \begin{cases} 1 & c = y \\ 0 & c \neq y \end{cases}$$

Es decir, la clase de la nueva instancia $\vec{x}_0$ será $y_0 \in C$ tal que maximice $\sum_{i=1}^{k} \delta(y_0, y_i)$.

8.1.3. Funcionamiento del algoritmo

Sea $T_{\text{train}} = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ el conjunto de entrenamiento y $\vec{x}_0$ una nueva instancia a clasificar:

1. Calcular todas las distancias de cada patrón de T_{train} a la nueva instancia.
2. Ordenar dichas distancias en orden ascendente.
3. Quedarse con las k instancias más cercanas (conjunto $D_{\vec{x}_0}^{(k)}$).
4. Asginar $\vec{x}_0$ a la clase más frecuente del conjunto $D_{\vec{x}_0}^{(k)}$.

8.1.4. Valor de k

- Si se elige un k muy bajo, el resultado es muy sensible al ruido.
- Si se elige un k muy alto, las zonas con muchos ejemplos acaparan a zonas que tienen menos ejemplos.
- Para estimar k probamos distintos valores, midienod los resultados dejando un elemento del conjunto fuera y clasificando al resto (LOO-CV).
- En general se elige k impar para no tener problemas de empate (caso de clasificación binaria).
- En caso de empate entre dos o más clases es conveniente tener una **regla heurística para su ruptura**:
  - Seleccionar la clase que contenga el vecino más próximo.
  - Seleccionar la clase con distancia media menor.

8.2. Variantes del k-NN

8.2.1. k-NN con rechazo

Se seleccionan una garantías para clasificar una nueva instancia.

- El número de votos obtenidos por la clase escogida tendrá que superar un umbral prefijado.
- El número de votos obtenidos por la clase más votada deberá superar a la segunda clase más votada en cierta cantidad.
- Para asignar una clase, debe haber mayoría absoluta.

8.2.2. k-NN con distancia media

Idea: Asignar un nuevo caso a la clase cuya **distancia media** con los k patrones más cercanos sea la menor (muy sensible al ruido).

Es decir, sea n_c el número de instancias de la clase $c \in C$ en el conjunto de las k instancias más cercanas:

$$f(\vec{x}_0) = \arg\min_{c \in C} \frac{1}{n_c} \sum_{i=1}^{k} \delta(c, y_i) d(\vec{x}_0, \vec{x}_i)$$

8.2.3. k-NN con ponderación de vecinos (*Distance-Weighted k-NN*)

Idea: Ponderar la importancia que aporta cada vecino al valor de la función objetivo en función de su distancia.

- Inverso de la distancia.
- Inverso de la distancia al cuadrado.
- Si $\exists \vec{x}_i = \vec{x}_0$, entones $y_0 = y_i$.

Para el caso discreto:

$$f(\vec{x}_0) = \arg\max_{c \in C} \sum_{i=1}^{k} w_i \delta(c, y_i), \qquad \text{donde} \quad w_i = \frac{1}{d(\vec{x}_0, \vec{x}_i)^2}$$

Para el caso continuo:

$$f(\vec{x}_0) = \arg\max_{c \in C} \frac{\sum_{i=1}^{k} w_i \delta(c, y_i)}{\sum_{i=1}^{k} w_i}$$

8.2.4. k-NN con distancia mínima

1. Se comienza seleccionando de T_{train} **una instancia por clase** (generalmente la instancia más cercana el centroide) que funcionará como **representante**.
2. Dado un nuevo caso a clasificar, se le asigna la clase del representante más cercano.
   - Ventaja: Menor coste computacional al kNN genérico.
   - Desventaja: Su efectividad está condicionada a la homogeneidad dentro de las clases.
   - Cuanto más homogéneas sean las clases, más efectivo.

8.2.5. kNN con ponderación de variables

La función que calcula la distancia cambia para ponderar algunas variables. Sea $\vec{w} \in \mathbb{R}^m$ el vector de ponderación de las m variables. En el caso de la distancia euclídea:

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{r=1}^{m} w_r (x_{ir} - x_{jr})^2}$$

Para calcular $\vec{w}$ podríamos usar la **medida de información mínima** entre una variable X (del conjunto de m variables) y la variable que indica la clase Y:

$$I(X, Y) = \sum_{x_i, y_i} p_{XY}(x_i, y_i) \ \log \frac{p_{X,Y}(x_i, y_i)}{p_X(x_i) \cdot p_Y(y_i)}$$

- Reducción en la incertidumbre sobre Y cuando se conoce el valor de X.
- Cuanto mayor sea $I(X, Y)$, mayor será la dependencia entre (X, Y).
- El peso w_r asociado a la variable X_r será proporcional a $I(X_r, Y)$s

8.3. Soluciones al problema del tamaño del conjunto de datos

8.3.1. Indexación

- Árboles KD (*Locally Weighted Regressión*).
- Agrupación o clustering.

8.3.2. Selección de instancias

Elegir un grupo reducido de instancias S (prototipos) que mantenga la misma información que el conjunto total T.

- Método incremental.
  1. Iniciar S vacío.
  2. Se añaden instancias a S a partir de T siempre que se cumpla un determinado criterio.
  3. Ejemplo de criterio: Cuando al intentar clasificar dicha instancia, se clasifica de forma incorrecta.
- Método decremental.
  1. Se comienza con $T = S$.
  2. Se van eliminando instancias de S cuando se cumpla un determinado criterio.
  3. Ejemplo de criterio: Que al extraer dicha instancia de S, se sigue clasificando correctamente.

8.3.3. Reemplazo de instancias

Learning Vector Quantization (LVQ): Calcular prototipos a partir del conjunto de entrenmiento.

1. Comenzar con un conjunto de prototipos $S = \{(\vec{s}_1, s_1), ..., (\vec{s}_p, s_p)\}$.
2. Para cada nueva instancia $(\vec{x}_0, y_0)$ del conjunto de entrenamiento, obtener el prototipo más cercano:

$$\vec{s}_0 = \arg\min_i d(\vec{x}_0, \vec{s}_i)$$

3. Actualizar la posición $\vec{s}_0$:

$$\vec{s}_0 = \begin{cases} \vec{s}_0 + \alpha(\vec{x}_0 - \vec{s}_0) & y_0 = s_0 \\ \vec{s}_0 - \alpha(\vec{x}_0 - \vec{s}_0) & y_0 \neq s_0 \end{cases}$$

8.4. Métodos relacionados

8.4.1. Regresión local ponderada (*Locally Weighted Regression*)

- Se construyen modelos lineales de forma local.

- Construcción de un modelo cuando se realiza la consulta de un nuevo caso con los k vecinos más cercanos.
- Se devuelve esa salida y se elimina el modelo.

8.4.2. Razonamiento basado en casos (*Case-Based Reasoning*, CBR)

El CBR busca en una base de datos (Base de Casos) problemas similares que hayan sido anteriormente resueltos con éxito (casos) y adapta las soluciones para dar una solución el nuevo problema.

1. Recuperar (*retrieve*). Dado un problema, se recuperan los casos más similares de la Base de Casos.
2. Reutilizar (*reuse*): Extraer la solución del caso seleccionado y adaptarla si es necesario.
3. Revisar (*revise*). Analizar si la nueva solución es aceptable y si es necesario revisarla.
4. Retener (*retain*). Después de haber aplicado la nueva solución con éxito, almacenarla en la experiencia como un nuevo caso de la Base de Casos.

Para recuperar los casos más similares se debe buscar una métrica de similitud que depende del dominio de trabajo.

Los casos en vez de ser instancias representadas en un espacio euclídeo pueden ser representaciones más complejas que se puedan comparar entre ellas.

8.4.3. Algoritmos perezosos vs Algoritmo voraces

Los algorimos perezosos:

- Retrasan el cálculo de una hipótesis hasta la llegada de una nueva consulta.
- Computan una aproximación local de la función objetivo para una nueva consulta.
- Mayor adaptación a nuevas consultas.

Los algoritmos voraces:

- Se elabora un modelo.
- Puede utilizar aproximaciones, pero estas quedan fijas en el conjunto de entrenamiento.

Bloque III
Computación Evolutiva

Tema 9: Computación Evolutiva

9.1. Algoritmos Genéticos

Motivación. Crear un algoritmo con la misma filosofía que emplea la naturaleza.

Los **algoritmos genéticos** (AG) son algoritmos de búsqueda inspirados en proesos de selección natural. Establecen una analogía entre el conjunto de posibles soluciones de un problema y el conjunto de individuos de una población natural.

- Aplicados principalmente en problemas de optimización.
- Comportamiento eficaz en problemas de superficie compleja, con múltiples mínimos locales y grandes espacios de búsqueda.
- Aplicado a problemas no resolubles polinomialmente (NP-duros).

9.1.1. Componentes de un algoritmos genético

1. Codificación del material genético de un individuo/solución como un vector de valores.
2. Generación aleatoria de una población inicial.
3. Función de evaluación en términos de conveniencia/adaptación a los individuos de la población (**función Fitness**).
    - Debe "penalizar" las malas soluciones y "premiar" las buenas, de forma que estas se propaguen con mayor rapidez.
    - Determina que soluciones tienen mayor o menor probabilidad de sobrevivir.
4. Operadores genéticos que cambien la composición de los descendientes.
5. Valores de los parámetros utilizados.

9.1.2. Esquema de un algoritmo genético

1. Generar la población inicial aleatoriamente (Generación 0).
2. Evaluar la población actual.

3. ¿Se ha satisfecho el criterio de parada?
   - Sí: Finaliza el algoritmo.
   - No: Construir la siguiente generación (ciclo) a partir de la anterior combinando los individuos mediante operadores genéticos.

9.1.3. Criterio de Selección

Objetivo. Seleccionar individuos de la población actual que generarán descendientes para una nueva población que reemplazará a la actual.

- **Ruleta**. Sea $\vec{x}$ un individuo de los n totales en la población y $f(\vec{x})$ su valor según la función Fitness. Entonces su probabilidad de ser seleccionado por ruleta es:

$$\mathbb{P}(\vec{x}) = \frac{f(\vec{x})}{\sum_{i=1}^{n} f(\vec{x}_i)}$$

- **Torneo**. Se selecciona al azar un número fijo de individuos y se escoge el que tenga mayor valor de Fitness.

Observaciones:

- En ocasiones se mantienen los k mejores individuos de una población para la siguiente (estrategia elitista).
- El torneo reparte la búsqueda en el espacio de estados (**exploración**).
- La ruleta pone más presión de búsqueda sobre el mejor individuo (**explotación**).

9.1.4. Operador de cruce

- Forma de calcular el genoma de un nuevo individuo en función del genoma de sus progenitores.
- Es responsable de las propiedades del algoritmo genético y determinará la evolución de la población.

Posibles técnicas de cruce:

- Cruce básico de un punto.
- Cruce multipunto.
- Cruce uniforme.
- Cruce segmentado.
- Cruces por permutación.
  - Cruce de mapeamiento parcial.
  - Cruce de orden.
  - Cruce de ciclo.

9.1.5. Operador de mutación

- La operación se realiza sobre un único individuo. Determina un gen (posición en el vector) de forma aleatoria y lo invierte con cierta probabilidad.

- Permite salir de máximos/mínimos locales.
- Contribuye a la diversidad genética de la población.

Para aplicar la mutación se genera un número aleatorio en cada operación de mutuación. Si dicho número está por debajo de una probabilidad, se cambia el gen.

Técnicas de mutación:

- Mutación de bit.
- Mutación multibit.
- Mutación de gen.
- Mutación multigen.
- Mutación de intercambio.
- Mutación de barajado.

9.1.6. Cruce vs mutación

El cruce busca combinaciones, mientras que la mutación busca variabilidad. Se debe buscar un compromiso entre las dos operaciones:

- El cruce favorece a la explotación.
- La mutación favorece a la exploración.

9.1.7. Sustitución

Objetivo. Mantener el tamaño de la población.

- Generacional. Generar una nueva población.
- Steady-state. No se sustituye una población por otra. Una vez nace un nuevo individuo, se sustituye por otro de la misma población:
  - Por los peores (homogeneización).
  - Por los padres.
  - Por los parecidos.
- Elitismo. Mantener siempre a los k mejores individuos.

9.1.8. Función Fitness

Las técnicas de CE no garantizan la consecución del óptimo global.

- Comienzan con una distribución inicial uniforme sobre el espacio de soluciones.
- Consiguen una convergencia muy rápida a la zona del óptimo global.
- Se pueden combinar con técnicas de optimización local como el escalado de colinas (*Hill Climbing*).

9.1.9. Criterio de paro

Normalmente el AG se detiene cuando un porcentaje alto de la población converge a un valor. Si con ese valor no se llega a la medida esperada, entonces tenemos dos opciones:

- Se toma una pequeña proporción y se inyecta diversidad genética.
- Se reemplaza completamente la población.

9.1.10. Criterios sobre parámetros

- **Tamaño de la población**. Normalmente se elige un tamaño de población fijo.
- **Probabilidad de cruce** (90 %).
- **Probabilidad de mutación** (5-10 %).
- Se pueden variar según las fases del algoritmo (al inicio cruce alto y mutación bajo).

9.1.11. Conclusiones

Diferencias con métodos tradicionales de búsqueda y optimización:

- Trabajan con un conjunto de parámetros codificados y no con los parámetros mismos.
- Inician la búsqueda desde un conjunto de puntos, no desde uno solo.
- Usan una función a optimizar en lugar de derivadas u otros conocimientos adiciones difíciles de conseguir.
- Usan reglas de transición probabilísticas no determinísticas.

Ventajas:

- No necesitan conocimientos específicos sobre el problema a resolver.
- Operan de forma simultánea con varias soluciones.
- No necesitan recorrer todo el espacio de búsqueda.
- **Robustez**. El AG encuentra de forma eficiente la solución del problema aunque varíen sus parámetros.
- AG es independiente al problema original.
- Se usan para problemas de optimización-maximización de una función objetivo.
- Se ven menos afectados por los máximos locales.

Desventajas:

- Usan operadores probabilísticos.
- Pueden tardar mucho en converger, no converger o converger prematuramente.

Cuando NO usar AG:

- Si se requiere forzosamente llegar al óptimo global.
- Si conocemos la función de optimización.

- Si el problema está muy demilitado y se presta a un tratamiento analístico (funciones de una variable).
- Si la función es suave y convexa.
- Si el espacio es limitado entonces mejor enumerar todas las soluciones.

9.2. Programación Genética

La programación genética surge como una evolución de los algoritmos genéticos tradicionales, manteniendo el mismo principio de selección natural.

La mayor diferencia entre un AG y la programación genética es la forma de **codificación** de la solución al problema. En programación genética:

1. La codificación se realiza en forma de **árbol**.
2. El **operador de cruce** entre dos individuos (árboles) consiste en la selección de un nodo al azar y el intercambio de los subárboles debajo de dicho nodo para generar los dos hijos.ç
3. El **operador de mutación** consiste en tomar nodos al azar del árbol y cambiar los valores que se alojan dentro.

9.3. Aprendizaje Evolutivo

El aprendizaje evolutivo es una rama de la Inteligencia Artificial aplicable en problemas de optimización combinatoria.

- Inspirada en mecanismos de Evolución Biológica.
- Técnicas de *Soft Computing*.
- Aplicaciones: clasificación, predicción, modelado, clustering...

Técnicas de Computación Evolutiva:

- Optimización de Enjambre de Partículas
- Programación Evolutiva
- Colonias de Hormigas
- Estrategias Evolutivas
- Evolución Diferencial
- *Coral Reefs Optimization*
- *Grey Wolf Optimization*

9.4. Optimización de Enjambres de Partículas (PSO)

Utiliza un número de partículas que constituyen un enjambre que se mueve alrededor del espacio de búsqueda buscando la mejor solución.

Cada partícula es una solución al problema:

- Ajusta su "vuelo" según su propia experiencia de vuelo y la experiencia de otras partículas.
- Tiene en cuenta su posición y velocidad actual.
- Tiene en cuenta la posición de su mejor solucion: `pbest` (personal best).
- Tiene en cuenta la posición de la mejor solución del vecindario: `gbest` (global best).
- Tiene un vecindario fijo.

9.4.1. Reglas de actualización de las partículas

Sea x una partícula del enjambre:

- p: posición de la partícula.
- v: velocidad, dirección.
- c_1: peso de la información local.
- c_2: peso de la información global.
- pBest: mejor posición de la partícula.
- gBest: mejor posición del enjambre.
- randn: variable aleatoria.

La actualización de dicha partícula en el instante t obedece la siguiente regla:

$$p(t+1) = p(t) + v + \underbrace{c_1 \text{rand}(\text{pBest} - p)}_{\text{Influencia personal}} + \underbrace{c_2 \text{randn}(\text{gBest} - p)}_{\text{Influencia social}}$$

- v es la inercia. Hace que la partícula se mueva en la misma dirección, con la misma velocidad.
- La **influencia personal** hace que la partícula vuelva a una posición previa mejor que la actual.
  - Mejora el individuo.
  - Medida conservadora.
- La **influencia social** hace que la partícula siga la dirección del mejor vecino.

El parámetro v incentiva la **diversificación** (exploración) de nuevas soluciones, encontrando regiones con soluciones potencialmente mejores.

La influencia personal y social incentiva la **intensificación** (explotación) de soluciones previas, encontrando la mejor solución en una región determinada.

- c_1 dicta la importancia del mejor valor personal.
- c_2 dicta la importancia del mejor valor del vecindario.
- Normalmente $c_1 + c_2 = 4$, y $c_1 = c_2 = 2$.

9.4.2. Conclusiones

Ventajas:

- Implementación sencilla.
- Fácilmente paralelizable para el procesado concurrente.
- Pocos parámetros que ajustar.
- Algoritmo de búsqueda global muy eficiente.

Desventajas:

- Tendencia a una convergencia rápida y prematura.
- Convergencia lenta en una etapa de búsqueda refinada.
- Baja habilidad de búsqueda local.

Bloque IV
Metodologías de Análisis de Datos

Tema 10: Metodología CRISP-DM

La metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*) proporciona una descripción normalizada del ciclo de vida de un proyecto estándar de análisis de datos.

Propone una secuencia de fases **no rígida**.

- Se permite el movimiento hacia delante y hacia atrás entre diferentes fases.
- El resultado de cada fase determina la siguiente.

Fases de CRISP-DM:

1. Definición de las necesidades del cliente.
2. Estudio y comprensión de los datos.
3. Análisis de los datos y selección de características.
4. Modelado.
5. Evaluación (obtención de los resultados)
6. Despliegue (puesta en producción).

Tema 11: Evaluación en CRISP-DM

- Comparación de los resultados de varios algoritmos:
  - Algoritmos distintos.
  - Mismo algoritmo con distintos parámetros.
- Si los algoritmos son estocásticos, es necesario minimizar la componente aleatoria repitiendo el experimento un número elevado de veces y promediar los resultados.

11.1. Conjunto de test

La evaluación siempre se realiza en el conjunto de test, que si se escoge de forma aleatoria:

- Es necesario minimizar dicha componente aleatoria repitiendo el experimento y promediando los resultados.

Por tanto tenemos dos fuentes de variabilidad:

- Elección del conjunto de entrenamiento y test.
- Variabilidad propia del algoritmo.

Además, para evaluar los resultados en el conjunto de test, no basta con la **media**. Hay que promediar también la **desviación típica**.

11.1.1. Técnicas para la obtención del conjunto de test

- HoldOut.
- Random Subsampling (submuestreo aleatorio),
- Validación Cruzada.
- Validación Cruzada estratificada.
- Leave-One-Out Cross Validation.

11.2. Comparación de dos modelos

Debemos definir un contraste de hipótesis:

$$\begin{cases} H_0 : \mu_1 = \mu_2 \\ H_1 : \mu_1 \neq \mu_2 \end{cases}$$

1. Seleccionar el estadístico de contraste.
2. Elegir el nivel de significación α del test. Esto es, la probabilidad de rechazar H_0 cuando es cierta.
3. Usamos la **región de rechazo**:
   - Determinar la región de rechazo C.
   - Calcular el estadístico de contraste para la muestra particular d.
   - Si $d \in C$, rechazamos H_0.
   - Si $d \notin C$, aceptamos H_0.
4. Usamos el p-valor: Probabilidad de obtener una discrepancia mayor de la observada bajo H_0.
   - Si $p \leq \alpha$, rechazamos H_0.
   - Si $p > \alpha$, aceptamos H_0.

   El p-valor informa sobre cuál sería el nivel de significación más pequeño que nos permitiría rechazar H_0.

11.2.1. T-test

$$\begin{cases} H_0 : \mu_1 = \mu_2 \\ H_1 : \mu_1 \neq \mu_2 \end{cases}$$

Suposiciones: Distribución normal con igualdad de varianzas.

Intervalo de confianza:

$$(\overline{X}_1 - \overline{X}_2) \pm t_{\alpha/2} S \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}, \qquad t_{\alpha/2} \in t_{n_1+n_2-2}$$

$$S = \sqrt{\frac{(n_1-1)S_1^2 + (n_2-1)S_2^2}{n_1+n_2-2}}$$

11.2.2. Test Wilcoxon

$$\begin{cases} H_0 : \mu_1 = \mu_2 \\ H_1 : \mu_1 \neq \mu_2 \end{cases}$$

- Test no paramétrico.
- Ninguna suposición.

Calculando el valor absoluto de diferencias entre las observaciones (x, y) ordenadas:

$$|z_1|, ..., |z_n| \qquad z_i = y_i - x_i$$

Se obtiene el rango de dichos valores: R_i.

El estadístico de prueba es:

$$W = \left\| \sum_{i=1}^{n} \text{sign}(y_i - x_i) R_i \right\|$$

11.2.3. ¿Qué test usar?

Si se cumplen las suposiciones, el t-test es más potente. Es decir, hay más probabilidad de rechazar H_0 cuando es falsa.

Si no se cumplen las suposiciones del t-test, el test de Wilcoxon es más potente y fiable. En general, el test de Wilcoxon es más robusto frente a casos atípicos.

Si se desconoce la distrivución de los errores de cada método, es mejor usar el test de Wilcoxon.

11.3. Comparar varios modelos

11.3.1. ANOVA

$$\begin{cases} H_0 : \mu_1 = \cdots = \mu_k \\ H_1 : \exists (i,j) \text{tal que } \mu_i \neq \mu_j \end{cases}$$

Suposiciones del test:

- Distribución normal de las muestras.
- Misma varianza de las diferentes poblaciones.
- Todas las observaciones son mutuamente independientes.

El test sigue siendo robusto para observaciones que no cumplan "ligeramente" las dos primeras suposiciones.

Si el test ANOVA concluye rechazando H_0, debemos investigar cuáles son las medias diferentes empleando un **método de comparación múltiple**.

Si el test ANOVA concluye aceptando H_0, se escoge el modelo más sencillo.

11.3.2. Kruskal-Wallis

$$\begin{cases} H_0 : \mu_1 = \cdots = \mu_k \\ H_1 : \exists (i,j) \text{tal que } \mu_i \neq \mu_j \end{cases}$$

Suposiciones del test:

- Todas las poblaciones provienen de una población continua.
- Todas las observaciones son mutuamente independientes.

11.3.3. Métodos de comparación múltiple

Comparación de las diferencias entre cada par de medias con ajustes apropiados a la comparación múltiple.

- Método de Tukey.
- Método de Scheffé.
- Método de Bonferroni.