Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Resúmen II

Introducción a la Inteligencia Artificial

Autor:
Arroyo, Joaquín

8 de junio de 2023

Índice

1. Razonamiento aproximado

El conocimiento que necesitamos para desarrollar un Sistema basado en Conocimiento(KBS) tiene muchas veces las siguientes características:

- No es del todo confiable: Falta de evidencias, excepciones
- Impreciso: El lenguaje usado para transmitirla es inherentemente impreciso, vago
- Contradictorio: Diferentes fuentes pueden ser conflictivas, redundantes, subsumidas
- Incompleto: Faltan datos provenientes de mediciones, análisis

El problema esta en como modelizamos estas caracteristicas del conocimiento, de modo de poder representarlo y utilizarlo.

Por ejemplo, a lógica clásica es un buen modelo para formalizar cualquier razonamiento basado en información certera (V o F), asi que necesitamos otros formalismos.

1.1. Conocimiento incierto

El conocimiento se expresa mediante predicados precisos pero no podemos establecer el valor de verdad de la expresión, por ejemplo:

Es posible que A pese más de 10kg.
Creo que el auto era rojo.

Cuando no podemos establecer la verdad o falsedad de la información, debemos evaluar la: PROBABILIDAD, POSIBILIDAD, NECESIDAD/PLAUSIBILIDAD, GRADO DE CERTEZA, de que la información sea verdadera

1.2. Conocimiento impreciso

El conocimiento cuenta con predicados o cuantificadores vagos (no precisos), por ejemplo:
Pedro tiene *entre 20 y 25 años.*
Juan es *joven.*
Mucha gente juega al fútbol.
El espectáculo es para *gente grande.*

Si la variable X toma valores en S:

- Proposiciones precisas: $\{p : X\ es\ s \mid s \in S\}$
- Proposiciones imprecisas: $\{p : X\ es\ r \mid r \subset S\}$

Imprecisa - no borrosa: Si r es un conjunto clásico
Imprecisa - borrosa(fuzzy): Si r es un conjunto borroso (fuzzy)

1.3. Conocimiento incompleto

Se debe tomar decisiones a partir de información incompleta o parcial. Esto se suele manejar a través de supuestos o valores por defecto, por ejemplo:

Si el paciente tiene S1, S2 y S3 entonces tiene una infección a Bacteria S3.

1.4. Conocimiento no-monótono

La información recibida a partir de distintas fuentes o en diferentes momentos es conflictiva y cambiante, por ejemplo:

Si el vuelo 1340 sale en forma puntual y no tiene escalas técnicas arribará a Madrid a las 8 hs:
1 Supongo no-escala técnica y concluyo arribará a Madrid a las 8 hs.
2 Aviso de escala técnica, debo revisar la conclusión del horario de arribo.

1.5. Lógicas no-monótonas

A partir de esto, viene la definición de las lógicas no-monótonas, que a diferencia de las lógicas clásicas, las cuales siempre creces a partir de teoremas y axiomas, en estas lógicas, se pueden 'volver a atrás' teoremas.

1.6. Conclusión

El Razonamiento Aproximado(RA) trata como representar, combinar e inferir con conocimiento impreciso y/o incierto.

Esquema general en sistemas basados en reglas de producción:
Hipótesis:
- Si X es A entonces Y es B (λ)
- X es A*
Conclusión:
- Y es B* ?

donde A y B son imprecisos, λ es una regla incierta y también pueden existir reglas hibridas(problema complejo).

2. Redes Bayesianas

Para representar la dependencia que existe entre determinadas variables, en aplicaciones complejas, se utiliza una estructura de datos conocida como Red Bayesiana, Red de creencias, Red Probabilística o Red causal.

Esta estructura sirve para especificar de manera concisa la distribución de probabilidad conjunta.

Las Redes Baysesianas son redes de relaciones probabilisticas entre proposiciones(variables aleatorias), relacionadas semánticamente(relaciones causales) y representadas por un grafo(GDA). Donde el GDA esta compuesto por:

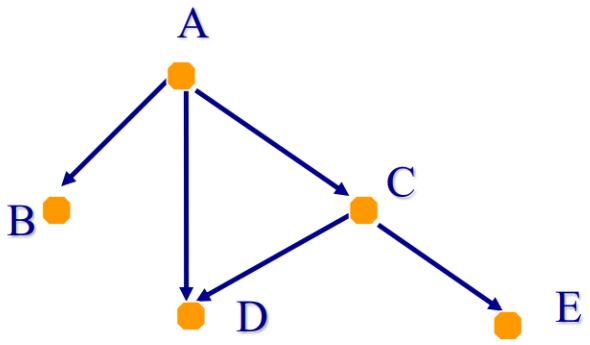
- Nodos: Proposiciones
- Arcos: Relaciones causales
- Peso de arcos: Probabilidad condicional

En estas redes hay que establecer, la topología de la redy las probabilidades condicionales, donde esto es una tarea más compleja(datos estadísticos, subjetivos, utilizar otras técnicas aprendizaje automatizado)

La incertidumbre inherente a los distintos enlaces (relaciones causales) representan las situaciones no representadas explícitamente.

Las probabilidades resumen un conjunto de posibles circunstancias en que pueden ser verdaderas (falsas) las variables de un nodo.

Ejemplo:



Del grafo, que representa las relaciones causales, se puede sacar la distribución conjunta:
$P(A, B, C, D, E) = P(E/C)P(D/A, C)P(C/A)P(B/A)P(A)$

En general, es posible calcular cada una de las entradas de la distribución conjunta desde la información de la red:

$$\prod_{i=1}^{n} P(x_i)/Padres(x_i)$$

Estas redes presentan las siguientes caracteristicas:

- Independencia: Se hace explícita mediante la separación de grafos.
- Escalabilidad: Se construye incrementalmente por el experto agregando objetos y relaciones.

Los arcos no deben considerarse estáticos, representan restricciones sobre la certeza de los nodos que unen. $P(B/A)$ cuantifica la certeza de $B \implies A$, y si lo que se conoce es una evidencia e de que B es cierto, entonces $P(A/B, e)$

2.1. Inferencias

Belief revision: Consiste en encontrar la asignación global que maximice cierta probabilidad
Puede usarse para tareas explicatorias/diagnóstico

Básicamente a partir de cierta evidencia E, nuestra tarea es encontrar un conjunto de hipótesis que constituyan la mejor explicación de las evidencias

Encontrar asignaciones a los nodos $N_1, ..., N_j|(P(E, N_1, ..., N_j))$ sea máxima.

Belief updating: Consiste en determinar la mejor instanciación de una variable, dada una evidencia.

Es la actualización de probabilidades de un nodo dadas un conjunto de evidencias: $(P(Ni/E1, \ldots, En))$

2.2. Conclusión

Las Redes Bayesianas son modelos más cercanos a un modelo probabilístico puro y permite la representación explícitas de las dependencias del dominio en la red.

3. Fuzzy Systems

3.1. Fuzzy Logic

Es una rama de AI que permite trabajar con razonamientos imprecisos partiendo de conocimiento impreciso que se representa mediante conjuntos borrosos (fuzzy sets).

3.2. Conjuntos borrosos

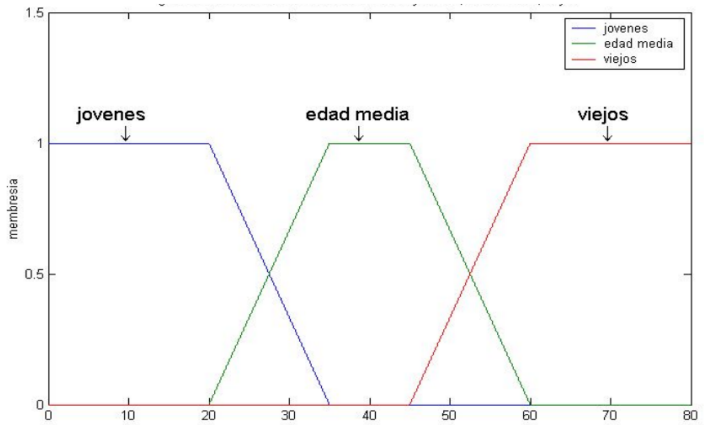
Definición formal: Sea X un espacio de objetos, un conjunto borroso A en x se caracteriza por una función de pertenencia $f_A(x)$ que asocia a cada elemento de X un número real en $[0, 1]$, donde el valor $f_A(x)$ representa el 'grado de pertenencia de x en A'.

$A = \{x, f_A(x)\}$

Ejemplo: $A = (a_1/x_1, .., a_n/x_n)$ considerando el conjunto difuso $Alta$ asociado a la variable lingüística estatura:

$Alta = (0/1{,}65, 0{,}8/1{,}75, 0{,}9/1{,}85, 1/1{,}95)$

Más ejemplos de conjuntos borrosos:



3.2.1. Funciones de pertenencia

Las funciones de pertenencia a los fuzzy sets mapean cada elemento del conjunto borroso a un valor de pertenencia.

$f_A(x) : X \implies [0, 1]$ donde X es el espacio de entrada o Universo.

Los distintos tipos de funciones de pertenencia son:

Triangular, Trapezoidal, Gaussianas y Sigmoides.

Para elegir una funcion de pertenencia:

Evaluación subjetiva Individuos asignan un grado de pertenencia subjetivo a cada elemento

Frecuencias o probabilidades Estadísticas basadas en histogramas o el porcentaje de respuestas afirmativas y negativas sobre la pertenencia de un elemento al conjunto.

Funciones ad-hoc En los sistemas borrosos de control se suele utilizar funciones de pertenencia sencillas (triangulares o trapezoidales). Ajustes mediante experimentación.

3.2.2. Propiedades

Normalidad: A está normalizado si su supremo es 1
Igualdad: $A = B sii f_A(x) = f_B(x) \forall x \in X$
Inclusión: $A \subseteq B sii f_A(x) \leq f_B(x) \forall x \in X$

3.3. Operaciones

3.3.1. Operaciones básicas

Unión: $f_{A \cup B}(x) = max\{f_A(x), f_B(x)\}$
Intersección: $f_{A \cap B}(x) = min\{f_A(x), f_B(x)\}$
Complemento: $f_{\overline{A}}(x) = 1 - f_A(x)$

3.3.2. T-Conormas y T-Normas

las operaciones básicas no son las únicas. En las lógicas multivaluadas se han estuidiado y definido distintas formas de: Complementos, intersecciones y uniones.

Complementos: Se usa la misma: $f_{\overline{A}}(x) = 1 - f_A(x)$

T-Normas:

Intersección estándar: $Min(a, b) = min(a, b)$

Suma algebraica: $Prod(a, b) = a.b$

Suma acotada: $W(a,b) = max(0, a+b-1)$ (Dual de Lukasiewicz)

Unión drástica: $Z(a,1) = a$, $Z(1,b) = b$, $Z(a,b) = 0$

Luego, $Z \leq W \leq Prod \leq Min$

T-Conormas:

Unión estándar: $Max(a,b) = max(a,b)$

Suma algebraica: $Prod*(a,b) = a+b-a.b$

Suma acotada: $W^*(a,b) = min(1, a+b)$ (Dual de Lukasiewicz)

Unión drástica: $Z^*(a,0) = a$, $Z(0,b) = b$, $Z(a,b) = 1$

Luego, $Max \leq Prod* \leq W* \leq Z*$

3.3.3. Propiedades

Hay propiedades que son válidas en la teoría de conjuntos, que no lo son en la teoría de conjuntos borrosos, como las siguientes:

$A \cap \overline{A} = \emptyset$

$A \cup \overline{A} = U$

Todo lo realizado en conjuntos borrosos se puede transladar a la lógica borrosa.

$\cap =$ **and**, $\cup =$ **or** y $\overline{A} =$ **not**A

3.4. Razonamiento borroso

La lógica borrosa trata con proposiciones borrosas que asignan un valor a una variable lingüística, por ejemplo "estatura", el valor "estatura es alta", mediante un conjunto difuso A definido sobre el universo de discurso X de la variable lingüística: [0, 2.5mts].

Una variable linguística es caracterizada por:

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

x : Nombre de la variable base

$T(x)$: Conjunto de términos linguísticos de x que refieren a la variable base

X : Conjunto Universo

G : regla sintáctica (gramática) para generar términos linguísticos

M : regla semántica que asigna a cada término un significado

Definir una implicación es asignar una función de pertenencia a una agrupación antecedente-consecuente del tipo $P \implies Q$, esto nos permite razones afirmaciones como SI "la velocidad es normal" ENTONCES "la fuerza de frenado debe ser moderada"

Se puede definir de dos formas:

Teórica: Darle el significado de la lógica clásica

Práctica: Darle un significado causa-efecto, como la Implicación de Mamdani.

$P \implies Q = P \wedge Q$

$M_{P \implies Q}(u,v) = min(m_p(u), m_q(v))$

Los operadores de Mamdani (Min) y Larsen (Prod) son útiles para hacer un modelo de implicación como relación de causa-efecto. Son ampliamente utilizados en ingeniería.

3.5. Razonamiento aproximado

Razonamiento con información imprecisa o incierta. La inferencia difusa de la implicación está basada en la regla composicional de inferencia.

Premisa: Si u está en P entonces v está en Q

Hecho: u está $P*$

Consecuencia: c está $Q*$

Donde $Q*$ está determinado por la composición $Q* = P*.(P \implies Q)$ $(P \implies QmatrizasociativaM)$

En casos prácticos se utiliza la composición max-T-norma $Q* = P*.(P \implies Q)$.

Sean dos conjuntos P y Q definidos en U y V respectivamente:

$Max\{T[P*(u),(P \implies Q)(u,v)]\}, u \in U, v \in V$

T: si existe un solo camino de conexión entre $P_i * y(P \implies Q)_{ij}$, tomamos "el menor" de los grados de pertenencia asociados de cada tramo.

Max: si existe más de un camino de conexión, tomamos "el mayor".

3.5.1. Inferencias borrosas

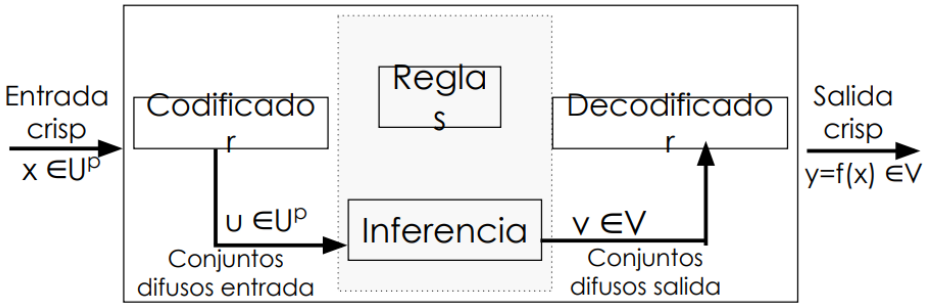
max-min: REGLA: IF A THEN B

Cuando $A*$ tiene un solo valor de pertenencia distinto de 0, por ejemplo x_k se puede utilizar solo $\mu_A(x_k)$ directamente con la representación de Q, $\mu_Q(y)$ para inducir $B*$ como $B* = \mu_A(x_k) \wedge \mu_B(y)$

max-T: En el caso que la entrada a la regla sea una lectura difusa, nosotros podemos considerar el max de la intersección de A y $A*$, es decir: $max(min(a_i, a*_i))$ para inducir el $B*$

3.6. Arquitectura de FuzzySystems

La salida de un proceso de inferencia es un conjunto difuso, en muchos procesos se requieren valores crisp.



3.7. Defusificadores

La defusificación es el proceso de convertir una salida difusa, que representa una distribución de pertenencia en un conjunto difuso, en un valor numérico o una decisión concreta.

Centroide: El centroide se calcula tomando el centro geométrico ponderado de la distribución de pertenencia. Cada valor de pertenencia se multiplica por su correspondiente valor de entrada y se divide por la suma de todos los valores de pertenencia. El resultado es el punto central ponderado y representa el valor de salida defusificado.

Valor max: En el método de defusificación del valor máximo, se selecciona el valor máximo de la distribución de pertenencia y se toma como valor de salida defusificado. Básicamente, se toma el punto en el que la pertenencia alcanza su valor máximo y se utiliza ese valor como resultado.

Promedio de max: El método de defusificación del promedio de máximo también encuentra el valor máximo de la distribución de pertenencia. Sin embargo, en lugar de tomar ese valor directamente como resultado, se calcula el promedio de todos los puntos donde la pertenencia alcanza su máximo valor. Este promedio se toma como valor de salida defusificado.

4. Aprendizaje Automatizado

Trata problemas que pueden ser simples de 'entender' o 'reconocer' pero muy difíciles de 'definir' y convertir en algoritmo. Por ejemplo, detectar una sonrisa en una cara, reconocer un gato, interpretar una secuencia de sonidos para traducirlo en palabras.

El Aprendizaje Automatizado introduce métodos que pueden resolver alguna de esas tareas 'aprendiendo' la solución a partir de ejemplos de cómo se realizan.

Dentro del AA se tratan los distintos problemas, clasificación, regresión, Búsqueda-recomendación, ranking-retrieval, detección de novedades, clustering, etc

4.1. Clasificación

Problema: Dado un objeto (descripto por un conjunto de características medidas de alguna forma) asignarle una (o varias) etiqueta/s de un conjunto finito. Por ejemplo, asignar un símbolo alfanumérico a una secuencia demovimientos del lápiz en la pantalla táctil.

4.2. Regresión

Problema: Dado un objeto asignarle un número real. Por ejemplo, predecir la relación euro-dolar de mañana, niveles de stock/ventas a futuro, cuestiones climáticas, etc.

4.3. Búsqueda-recomendación

Problema: Dado un objeto (necesidad de información/query), asignarle y ordenar las respuestas más adecuadas dentro de una base de datos. Por ejemplo, nuscadores en Internet, sistemas de recomendación.

4.4. Detección de novedades

Problema: Detectar outliers, objetos que son diferentes a los demás. Por ejemplo, alarmas de comportamiento en compras con tarjeta, detección de fallas en equipos críticos.

4.5. Cuándo un programa aprende?

Se dice que un programa aprende si mejora su performance en una cierta tarea al incorporar experiencia. Memorizar no es aprender, generalizar es aprender.

Para generalizar incorporamos "algo" a los datos: un bias, n general usamos la "navaja de Occam": La respuesta más simple que explica las observaciones es la válida.

4.6. Tipos de AA

Los programas de AA son programas que mejoran "su comportamiento" con la experiencia.

Hay dos formas de adquirir experiencia:

Aprendizaje supervisado: A partir de ejemplos suministrados por un usuario (un conjunto de ejemplos clasificados o etiquetados).

Aprendizaje NO supervisado: Mediante exploración autónoma (ej. software que aprende a jugar al ajedrez mediante la realización de miles de partidas contra sí mismo).

Dentro de estos, hay distintos tipos de aprendizaje:

Aprendizaje inductivo: Datos de entrada específicos: un usuario provee un subconjunto de todas las posibles situaciones. Datos de salida generales: regla o modelo que puede ser aplicada a una nueva situación.

Aprendizaje por refuerzo: Sistemas que aprenden mediante prueba y error. Exploración autónoma para inferir reglas de comportamiento.

Otros: Aprendizaje deductivo (EBL), Razonamiento basado en casos (CBR)

4.7. Aprendizaje por refuerzo

No hay fuente de información (no hay datos de entrada), el sistema aprende mediante prueba y error. Se realiza una exploración autónoma para inferir reglas de comportamiento (aprendizaje no supervisado). El sistema realiza una determinada tarea repetidamente, para adquirir experiencia y mejorar su comportamiento, y se requiere un número de repeticiones muy elevado. EXPLORACIÓN AUTÓNOMA $\implies$ MODELOS.

Tiene aplicaciones en procesos que se realizan como una secuencia de acciones como

Robots móviles: aprendizaje de la forma de escapar de un laberinto.

Juego de ajedrez: aprendizaje de la mejor secuencia de movimientos para ganar un juego.

Brazo robot: aprendizaje de la secuencia de acciones a aplicar a las articulaciones para conseguir un cierto movimiento.

Separación de imágenes de grandes grupos: caras de personas – animales – ninguno de ellos

4.8. Aprendizaje inductivo

El objetivo es generar un modelo a partir de ejemplos. El conjunto de ejemplos usados se llama conjunto de entrenamiento, y tiene cuatro elementos fundamentales: modelo resultante (hipótesis), instancias, atributos y clases. EJEMPLOS ESPECÍFICOS $\implies$ MODELO GENERAL.

4.9. Problemas de Machine Learning

Estos problemas llevan las siguientes definiciones:
Resultado: modelo que se infiere a partir de los ejemplos.
Instancia: cada uno de los ejemplos.
Atributo: cada una de las propiedades que se miden (observan) de un ejemplo.
Clase: el atributo que debe ser deducido a partir de los demás.

¿Como se resuelven este tipo de problemas? Identificar el problema, conseguir datos(muchos), elegir un método adecuado (o varios), entrenar varios modelos con el conjunto de entrenamiento, evaluarlos con el conjunto de validación, estimar el error con el conjunto de testeo.

Existen varios criterios para la selección del modelo:

- El tipo de modelo (árboles de decisión, redes neuronales, modelos probabilísticos, etc.)
- El algoritmo utilizado para construir o ajustar el modelo a partir de las instancias de entrenamiento (existen varias maneras de construir árboles de decisión, varias maneras de construir redes neuronales, etc.)

4.10. Árboles de Decisión

Están compuestos de nodos y ramas.Representan reglas lógicas (if - then).

- Nodos internos = atributos (atributo-valor)
- Nodos hoja = clases
- Nodo raíz = nodo superior del árbol

Su objetivo es obtener un árbol de decisión(resultado) a partir de un conjunto de instancias o ejemplos. El bias utilizado: árbol mínimo.

4.10.1. Aprendizaje(DTL)

Método para aproximar funciones de clasificación(variable objetivo toma valores discretos). Uno de los métodos de aprendizaje inductivos más conocidos. Actualmente se utilizan ensambles de árboles de decisión (conjuntos de árboles-bosques).

Se utilian algoritmos para la creación de árboles a partir de datos, generalmente se procede: Selección de nodo raíz(a partir de la garantía de info.) y luego se sigue con un proceso recursivo.

4.10.2. Algoritmo

Se realiza una búsqueda local, en profundidad(de arriba hacia abajo), a través del espacio de posibles árboles de decisión (ID3: Iterative Dichotomiser 3 y C4.5). Raíz: el atributo que mejor clasifica los datos

¿Cuál atributo es el mejor clasificador? respuesta basada en la ganancia de información, el atributo que permite obtener mayor ganancia de información es el seleccionado para dividir el nodo.

Hay ganancia de información cuando la división envía instancias con clases distintas a los distintos nodos. Para esto se utiliza como medida a la Entropía(Mide incertidumbre): $E(S) = -p_+ log_2(p_+) - p_- log_2(p_-)$, donde S : conjunto de datos actual, p_+ : proporción de ejemplos positivos y p_- :proporción de ejemplos negativos.

Luego, con esto se calcula: $Gain(S, A) = E(S) - \sum(|S_v|/|S|)E(S_v) \ \ \forall V \in Valores(A)$

Existen otas medidas para decidir, como la impureza de Gini y la reducción de la varianza.

La entrada: conjunto de entrenamiento, objetos caracterizables mediante propiedades (pares atributos-valor), la función objetivo toma valores discretos y los datos de entrenamiento pueden contener errores.
El espacio+proceso: El espacio de hipótesis (modelo) para ID3 es el conjunto de todos los árboles posibles. ID3 realiza una búsqueda hill-climbing de lo simple a complejo, no hace backtracking Comienza con el árbol vacío y utiliza para la evaluación la ganancia de información
La salida: un árbol de clasificación (nodos hojas en una clase), en árboles de decisión: una decisión (sí o no).

El algoritmo ID3 se aplica a atributos distretos, mientras que el C4.5 además se puede aplicar a atributos continuos (se discretizan), requiere un mínimo de ganancia en cada nodo.
ID3 nunca produce árboles demasiado grandes mientras que C4.5 sí, pues puede repetir atributos(temp ¡26, temp ¿24, temp ¡25, etc). Por esto último se puede generar *Overfitting* y por lo tanto hay que podar(*Pruning*)

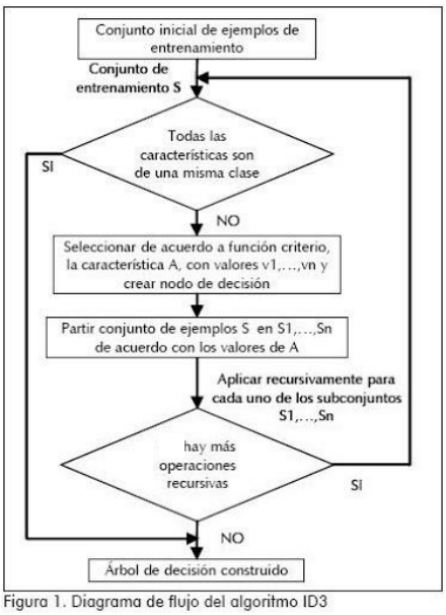
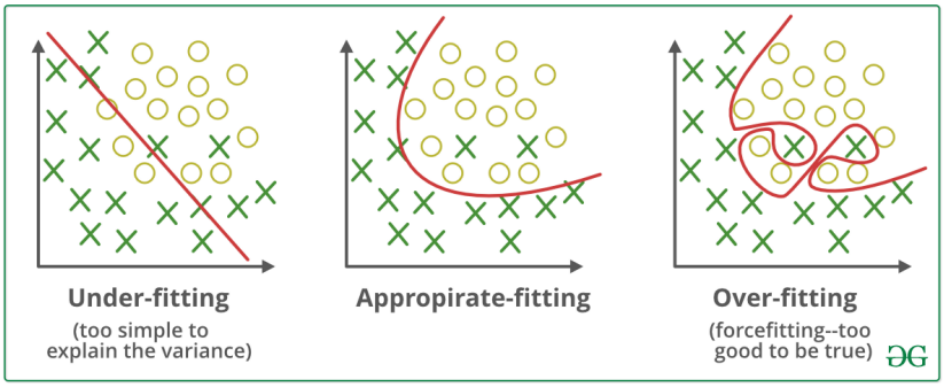


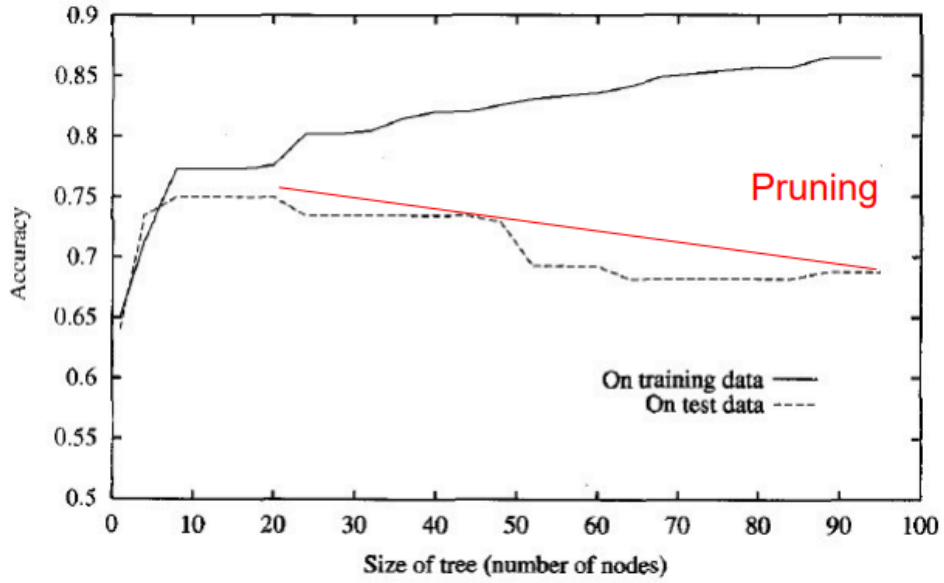
Figura 1. Diagrama de flujo del algoritmo ID3

4.10.3. Overfitting y Pruning

En una hipótesis(modelo) se dice que existe sobreentrenamiento si existe alguna otra hipótesis que tiene mayor error sobre los datos de entrenamiento pero menor error sobre todos los datos.



Se debe evitar el sobreentrenamiento: parar el crecimiento del árbol o post-procesamiento del árbol (poda)



4.10.4. Árboles de Clasificación

Un árbol de clasificación es un tipo específico de árbol de decisión que se utiliza para la clasificación. En lugar de predecir valores continuos, como en la regresión, un árbol de clasificación asigna instancias a clases o categorías discretas. Por lo tanto, los árboles de clasificación son un caso especializado de árboles de decisión que se enfocan en el problema de clasificación.

4.10.5. Conclusiones

- Legibilidad: muy alta. Uno de los mejores modelos en este sentido
- Tiempo de cómputo on-line: muy rápido. Clasificar un nuevo ejemplo es recorrer el árbol hasta alcanzar un nodo hoja
- Tiempo de cómputo off-line: rápido. Los algoritmos son simples
- Robustez ante instancias de entrenamiento ruidosas: robusto
- Sobreentrenamiento o sobreajuste: Se controla a través de una poda

4.11. Evaluando hipótesis

¿Qué y Cómo medimos? Existen dos dificultades clave cuando la información disponible es limitada: Bias en lo estimado, y varianza en lo estimado.

Para estimar el *accuracy* de una hipótesis: Dada una hipótesis h y un dataset que contiene n ejemplos extraidos al azar de D:

¿Cuál es la mejor estimación del *accuracy* de h sobre futuras instancias?

¿Cuál es el error probable en esta estimación?

Para esto tenemos: Sample error y True error

- Sample error de la hipótesis h con respecto a la función objetivo f y un dataset S:
  $error_S(h) = (1/n) \sum_{x \in S} \{\delta_k[f(x), h(x)]\}$
- True error de la hipótesis h con respecto a la función objetivo f y distribución D:
  $error_D(h) = Pr_{x \in D}[f(x) \neq h(x)]$

¿Qué tan buena es la estimación de $error_D(h)$ proporcionada por $error_S(h)$?

Intervalos de confianza: Para hipótesis con valores discretos, $n \geq 30$ y $error_S(h) = r/n$, el valor más probable de $error_D(h)$ es $error_S(h)$.

Con aproximadamente $N\,\%$ de probabilidad, $error_D(h)$ miente en el intervalo:
$error_S(h) + -z_n[error_S(h)(1 - error_S(h))/n]^{1/2}$

Regla práctica: $n\ error_s(h)[1 - error_s(h)] \geq 5$

En general, podemos estimar el error sobre datasets finitos, sin Bias y con una varianza que decrementa con el tamaño del dataset.

4.11.1. Regresión: RMSE

RMSE(root-mean squared error) es una función de costo por regresión. Su fórmula es:

$$RMSE(f) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f(\mathbf{x}_i) - y_i)^2}$$

donde m es el número de ejemplos de test, $f(x_i)$, la salida predecida sobre x_i e y_i los valores actuales.

RMSE no tiene escala, por lo que valores altos en algún escenario, pueden ser bajos en otros.

Se puede normalizar el valor obtenido, esto es NRMSE(normalized root-mean squared error).

4.11.2. Matriz de confusión

True class-> Hypothesized class	Pos	Neg
Yes	TP	FP
No	FN	TN
	P=TP+FN	N=FP+TN

Confusion Matrix

- Multi-Class Focus:
  - **Accuracy** = (TP+TN)/(P+N)
- Single-Class Focus:
  - **Precision** = TP/(TP+FP)
  - **Recall** = TP/P
  - **Fallout** = FP/N
  - **Sensitivity** = TP/(TP+FN)
  - **Specificity** = TN/(FP+TN)

4.11.3. Curvas ROC

Medidas de rendimiento para clasificadores basados en puntuaciones:

- La mayoría de los clasificadores son, de hecho, clasificadores basados en puntuaciones.
- Las puntuaciones no necesariamente deben estar en intervalos predefinidos, ni siquiera en probabilidades o verosimilitudes.
- El análisis ROC tiene sus orígenes en la teoría de detección de señales para establecer un punto de operación para la tasa deseada de detección de señales.
- Se asume que la señal está corrompida por ruido (distribuido normalmente).
- El análisis ROC mapea la tasa de falsos positivos (FPR) en el eje horizontal y la tasa de verdaderos positivos (TPR) en el eje vertical; recordemos que: $FPR = FP(FP + TN) = 1 - Specifity$ y $TPR = TP(TP + TN) = Specifity$

El análisis ROC permite a un usuario visualizar el rendimiento de los clasificadores en sus rangos de operación. Sin embargo, no permite la cuantificación de este análisis, lo cual dificulta la comparación entre clasificadores.

El Área Bajo la Curva ROC (AUC) permite dicha cuantificación: representa el rendimiento del clasificador promediado sobre todas las posibles relaciones de costos.

4.11.4. Medida de rendimiento

Si decidimos una medida de rendimiento, ¿cómo la estimamos de manera imparcial? ¿Qué sucede si usamos todos los datos?.

Re-sustitución: Demasiado optimista (mejor rendimiento logrado con un sobreajuste completo).

Enfoque de retención: Reserve un conjunto de prueba separado T. Evalúe su medida en este conjunto.

Pros: Independencia del conjunto de entrenamiento. El comportamiento de generalización se puede caracterizar. Se pueden obtener estimaciones para cualquier clasificador.

Contras: Perdemos datos para el aprendizaje.

4.11.5. Re-muestreo

La necesidad de re-muestreo:

Pocos ejemplos de entrenamiento -¿aprendizaje de un clasificador deficiente. Tener muy pocos ejemplos en el conjunto de entrenamiento afecta el sesgo del algoritmo al hacer que su predicción promedio sea poco confiable.

Pocos ejemplos de prueba -¿estimaciones de error falsas. Tener muy pocos ejemplos en el conjunto de prueba resulta en una alta variabilidad en la estimación.

Por lo tanto, el re-muestreo: Proporciona estimaciones precisas del rendimiento al tiempo que permite que el algoritmo se entrene con la mayoría de los ejemplos de datos.

Re-muestreos simples:

Repetición de retención:

1. Reserve un conjunto de prueba aleatorio separado T.
2. Evalúe su medida en este conjunto.
3. Repita n veces.
4. Calcule el promedio de sus estimaciones.

Los conjuntos de prueba y entrenamiento tienen una superposición parcial. No hay independencia entre ellos.

Validación cruzada k-fold: Divide el conjunto de datos en k pliegues o partes aproximadamente iguales. Luego, se realiza el siguiente proceso:

1. Se selecciona uno de los pliegues como conjunto de prueba y los k-1 pliegues restantes se utilizan como conjunto de entrenamiento.
2. Se entrena el modelo utilizando el conjunto de entrenamiento y se evalúa su rendimiento utilizando el conjunto de prueba.
3. Se repiten los pasos 1 y 2 k veces, de modo que cada pliegue se utilice una vez como conjunto de prueba.
4. Se promedian las métricas de rendimiento obtenidas en cada iteración para obtener una estimación más robusta del rendimiento del modelo.

Tiene la ventaja de utilizar todos los datos para entrenamiento y evaluación, a diferencia del enfoque de retención donde se pierden datos para el aprendizaje. Además, proporciona una evaluación más confiable del rendimiento del modelo al promediar las métricas de rendimiento de múltiples iteraciones.

4.12. Redes neuronales

Una aproximación robusta para aproximar funciones objetivos a valores reales y discretos.

Inspiraciones biológicas: Usar RN para modelar y estudiar procesos de aprendizaje. Obtener algoritmos de ML muy efectivos, inspirados en el cerebro humano, mediante modelos artificiales.

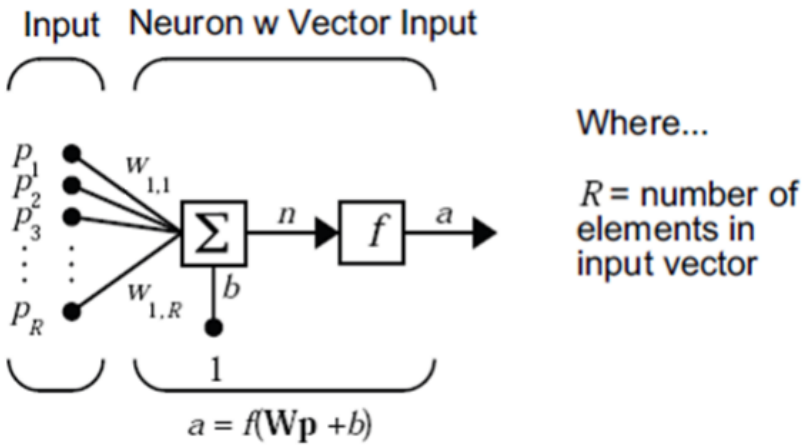
Una Red Neuronal Artificial (RNA) es un sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la neurona.

Las neuronas son un componente relativamente simple pero que conectadas de a muchas, forman un poderoso sistema.

Se basan en una red de unidades de procesamiento que intercambian datos o información. Tienen la capacidad de aprender y mejorar su funcionamiento. Se utilizan para reconocer patrones, incluyendo imágenes, manuscritos, tendencias financieras, etc.

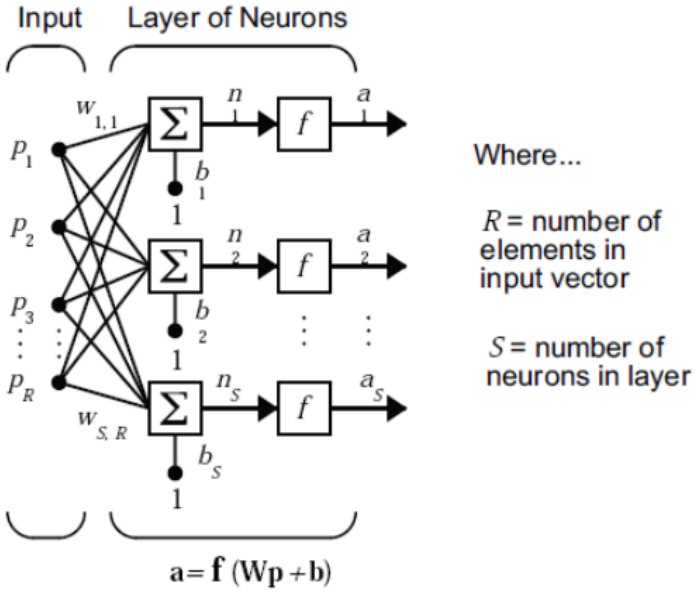
4.12.1. Estructura

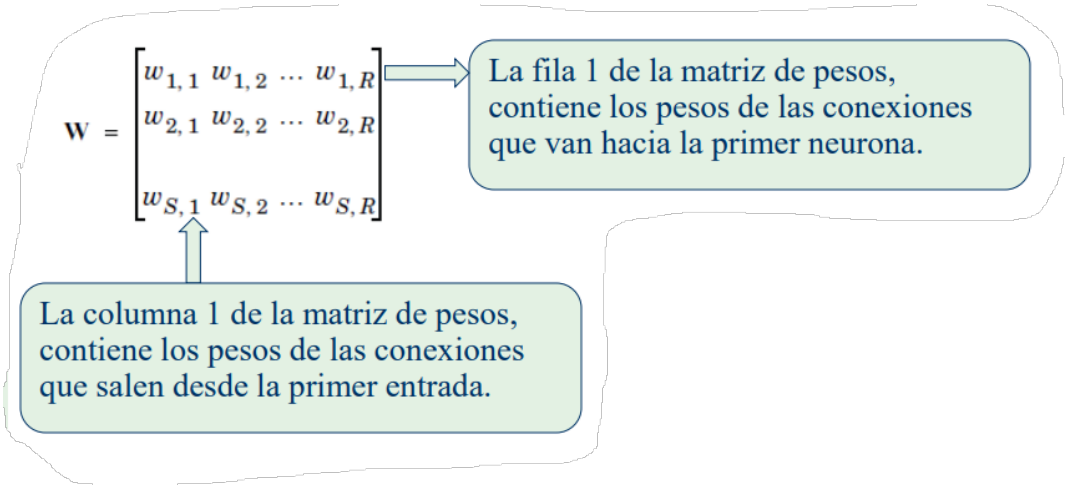
Neurona:



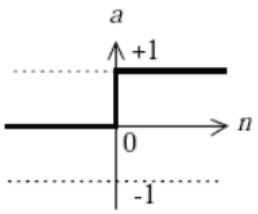
Estructura general: Se interconectan neuronas en tres tipos de capas:

- De entrada: reciben estímulos externos.
- Oculta: elementos internos de procesamiento (se pueden estructurar en varias capas).
- De salida: reciben la información procesada y retornan la respuesta del sistema al exterior.

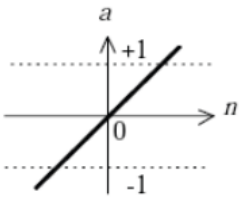




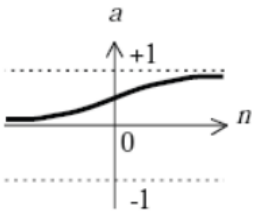
4.12.2. Funciones de transferencia



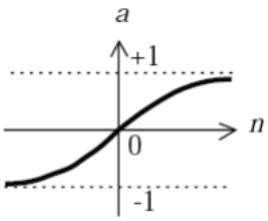
a = hardlim(n)



a = purelin(n)



a = logsig(n)



a = tansig(n)

$$a = logsig(n) = 1/(1+exp(-n))$$

$$a = tansig(n) = 2/(1+exp(-2n)) -1$$

4.12.3. Workflow

El diseño general del trabajo con RNA tiene los siguientes pasos básicos:

- Collect data
- Create the network
- Configure the network
- Initialize the weights and biases
- Train the network
- Validate the network (post-training analysis)
- Use the network

4.12.4. Redes *feedforward*

Las neuronas de una capa se conectan con todas las neuronas de la capa siguiente(hacia adelante).
Las más conocidas son: Perceptrón, Adaline y Backpropagation
Son muy útiles en aplicaciones de reconocimiento o clasificación de patrones.
También las redes *feedforward* pueden tener múltiples capas, estas son llamadas *feedforward-multiple-layer*, y son muy potentes.

4.12.5. Mecanismo y reglas de aprendizaje

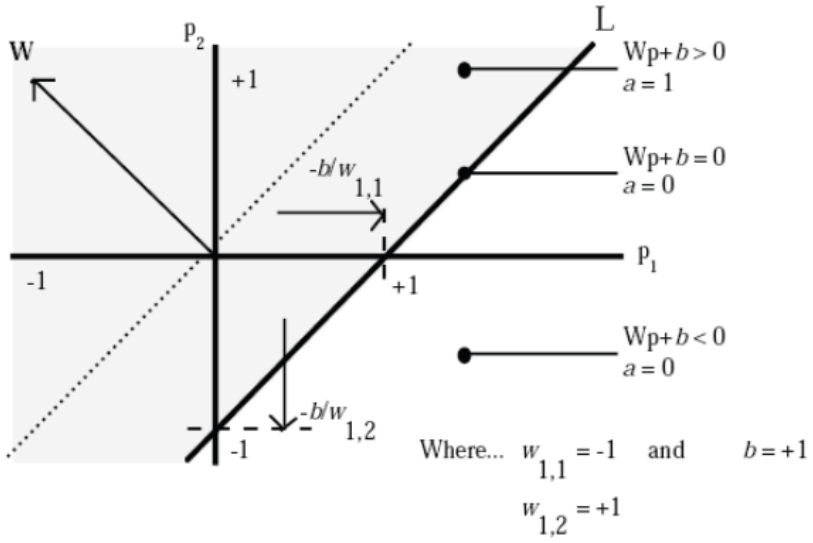
En las RNA se considera que el conocimiento se encuentra representado en los pesos de las conexiones. El proceso de aprendizaje se basa en cambios en estos pesos.

Los cambios en el proceso de aprendizaje se reducen a destrucción, modificación y creación de conexiones entre las neuronas. La creación de una conexión implica que el peso de la misma pasa a tener un valor distinto de cero. Una conexión se destruye cuando su valor pasa a ser cero.

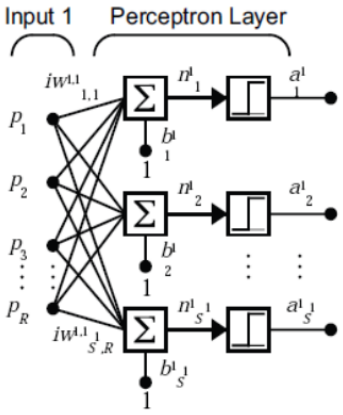
Una regla de aprendizaje es un procedimiento para modificar los pesos y umbrales de una red. Este proceso también puede ser llamado algoritmo de entrenamiento. El aprendizaje puede ser supervisado o no supervisado.

4.12.6. Red Perceptrón

Esta red utiliza la función transferencia escalón. Produce un 1 si la entrada a la función es mayor o igual que 0 y produce un 0 en caso contrario, esto permite clasificar vectores de entrada dividiendo el espacio de entrada en dos regiones.



La red Perceptrón consiste de una única capa oculta de S neuronas perceptrón conectadas a R entradas a través de conexiones con pesos w_{ij}. Los índices i y j indican que w_{ij} es el peso de la conexión desde la j-ésima entrada hacia la i-ésima neurona.



El perceptrón es entrenado con aprendizaje supervisado.

Un conjunto de pares de E/S (patrón/target) representan el comportamiento deseado, el objetivo es reducir el error e, que es la diferencia entre el valor deseado t y el resultante a: $e = t - a$

Se puede probar que la regla de aprendizaje del pereptrón converge hacia una solución en un número finito de pasos.

La estrategia consiste en modificar el vector w haciendo que éste apunte hacia los patrones cuya salida es 1 y en contra de los patrones cuya salida es 0.

Hay una función que realiza el cálculo de la variación de w.

CASO 1: $a = t$ ($e = 0$). El vector w no se altera.

CASO 2: $a = 0$ y $t = 1$ ($e = 1$). El vector de entrada p se suma a w. Esto provoca que w se cierre hacia p, incrementando la posibilidad de que p sea clasificado como 1 en el .

CASO 3: $a = 1$ y $t = 0$ ($e = -1$). A w se le resta el vector de entrada p. Esto provoca que w se aleje de p, incrementando la posibilidad de que p sea clasificado como 0 en el futuro.

En consecuencia, la regla de aprendizaje es: $W_i = W_i + \Delta W_i$ donde $\Delta W_i = \eta(t - a)p = \eta ep$, η es un learning rate pequeño (para dar pasos chicos).

Si consideramos el umbral como un peso cuya entrada siempre es 1 tenemos que los b se actualizan de igual forma.

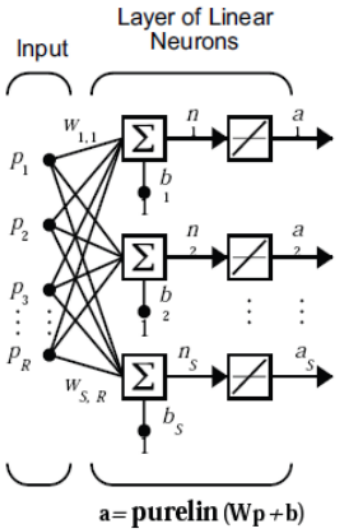
El proceso de encontrar nuevos pesos se repite hasta que el error sea aceptable. Se garantiza la convergencia en un número finito de pasos para todos los problemas que pueden ser resueltos por un perceptrón. Estos son todos los problemas de clasificación "linealmente separables".

La función de entrenamiento (train) realiza una pasada por todas las entradas calculando para cada una la salida y el error. Con cada entrada al perceptrón se van actualizando sus pesos y se introduce la entrada siguiente. Se necesita una pasada posterior por todas las entradas para evaluar si se ha alcanzado el objetivo.

4.12.7. Red Adaline

Son similares al perceptrón pero su función transferencia es lineal.

Al igual que el perceptrón estas redes resuelven problemas linealmente separables, y en este caso se busca minimiza el error cuadrático medio sobre todos los patrones de entrenamiento.



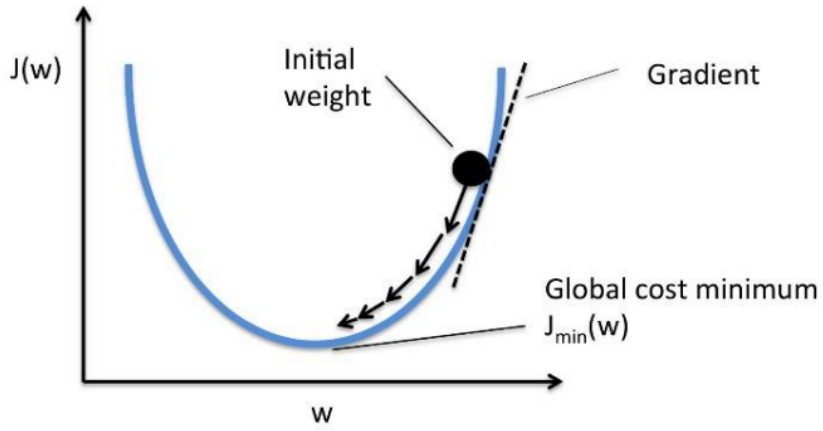
Error cuadrático medio (LMS): Sea un conjunto de Q entradas y targets: $\{p_1, t_1\}, ..., \{p_Q, t_Q\}$

Se aplican las Q entradas calculando cada error como la diferencia entre el objetivo y la salida. Se desea minimizar el promedio de los cuadrados de estos errores.

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q} (t(k) - a(k))^2$$

Algoritmo LMS (least mean square): También llamado algoritmo de Widrow-Hoff. La idea es ajustar los pesos para minimizar el mse (mean square error).

El gradiente apunta hacia la dirección de crecimiento más rápido: $e^2(w) = (t - a(w))^2$

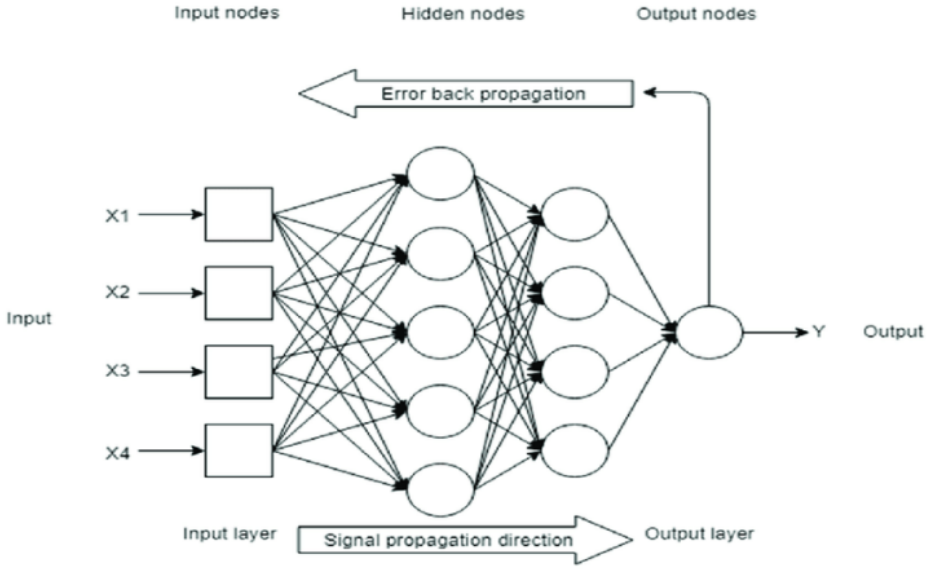


4.12.8. Redes Backpropagation

Son una generalización de las redes lineales.

Admiten múltiples capas y funciones derivables no lineales. Con una capa sigmoidea y una capa lineal son capaces de aproximar cualquier función con un número finito de discontinuidades.

Utilizan la regla de aprendizaje de Widrow-Hoff con una serie de variaciones. El término backpropagation se refiere a cómo se aplica el cálculo del gradiente hacia capas anteriores (el nuevo peso deseado en la capa s es el target de la capa $s-1$).



Las múltiples capas con funciones no lineales permiten que estas redes puedan resolver problemas "no linealmente separables".

Una capa de salida con función transferencia lineal permite que la red produzca valores más allá del (-1, 1).

Si se quiere restringir la salida, por ejemplo al (0, 1) se puede utilizar la función sigmoidea para la capa de salida.

Reconocimiento de patrones: Se puede entrenar redes para reconocimiento de patrones.

Son redes feedforward que pueden clasificar entradas en clases (conjuntos categóricos). Los datos objetivo (target) se presentan como vectores cuyos valores son todos cero excepto un 1 en la posición i, donde i es la clase que representan.

El dataset es dividido aleatoriamente en tres conjuntos: 60 % de los datos se utilizan para entrenamiento (ajuste de pesos y umbrales). 20 % de los datos se utilizan para validación (ajuste de otros parámetros - learning rate, etc). 20 % de los datos se utilizan para test (valoración del modelo obtenido).

4.13. Selección de modelo a utilizar

Para la selección del modelo tenemos dos decisiones fundamentales:

El tipo de modelo(árboles de decisión, redes neuronales, modelos probabilísticos, etc.)

El algoritmo utilizado para construir o ajustar el modelo a partir de las instancias de entrenamiento (existen varias maneras de construir árboles de decisión, varias maneras de construir redes neuronales, etc.)

Además se tienen en cuenta los siguientes criterios:

Fronteras de decisión: Separación de clases distintas. Cada modelo crea diferentes fronteras.

- **Árboles de decisión**: Fronteras perpendiculares a los ejes.

- **Redes Neuronales**: Fronteras no lineales: Mayor capacidad de representación. Permiten representar conceptos más complejos que los árboles de decisión.

*Legibilidad:*Capacidad de ser interpretado por un humano.

- **Árboles de decisión**: Fáciles de entender e interpretar: Conjunto de reglas. En los niveles más altos están los atributos más importantes.
- **Redes Neuronales**: Difíciles (o imposibles) de interpretar: Avances en redes profundas, interpretación de capas/unidades.

Un modelo legible puede ofrecer información sobre el problema que se estudia (ej. indicar qué atributos afectan la probabilidad de fallo de una máquina y cómo). Un modelo no legible sólo puede ser usado como un clasificador (ej. permite predecir si una máquina fallará o no aplicando el modelo).

Tiempo de cómputo on-line: Es el tiempo necesario para clasificar una instancia:

- **Árboles de decisión**: Tiempo necesario para recorrer el árbol, evaluando las funciones lógicas de cada nodo.
- **Redes Neuronales**: Tiempo necesario para realizar las operaciones (sumas, productos, sigmoides) incluidas en la red.

Este tiempo se consume cada vez que se debe clasificar una nueva instancia. Algunas aplicaciones requieren clasificar miles de instancias. Ejemplo: clasificación de cada uno de los píxeles de una imagen área de un cultivo, río, ruta, etc. Se requiere clasificar millones de instancias $\implies$ el tiempo de cómputo es muy importante.

Tiempo de cómputo off-line: Es el tiempo necesario para construir o ajustar el modelo a partir de los ejemplos de entrenamiento.

- **Árboles de decisión**: Tiempo necesario para elegir la estructura del árbol, los atributos a situar en cada nodo y la optimización mediante la poda.
- **Redes Neuronales**: Tiempo necesario para ajustar los pesos de las conexiones (puede tomar valores muy grandes).

Sólo se consume una vez, cuando mediante la utilización de los ejemplos de entrenamiento se genera y selecciona el resultado (modelo)más adecuado. Dependiendo de la aplicación no es un problema que el tiempo de cómputo off-line sea elevado (se deja una computadora procesando de uno a tres días enteros).

Robustez ante el ruido: Etiquetada incorrectamente (ejemplo: una máquina que no falló, etiquetada como que sí falló). Algún atributo no está valorizado.

Algunos algoritmos pueden funcionar adecuadamente aunque haya instancias ruidosas en el conjunto de entrenamiento (ej. árboles de decisión, redes neuronales).

Otros algoritmos no ofrecen buenos resultados (ej. kvecinos más cercanos).

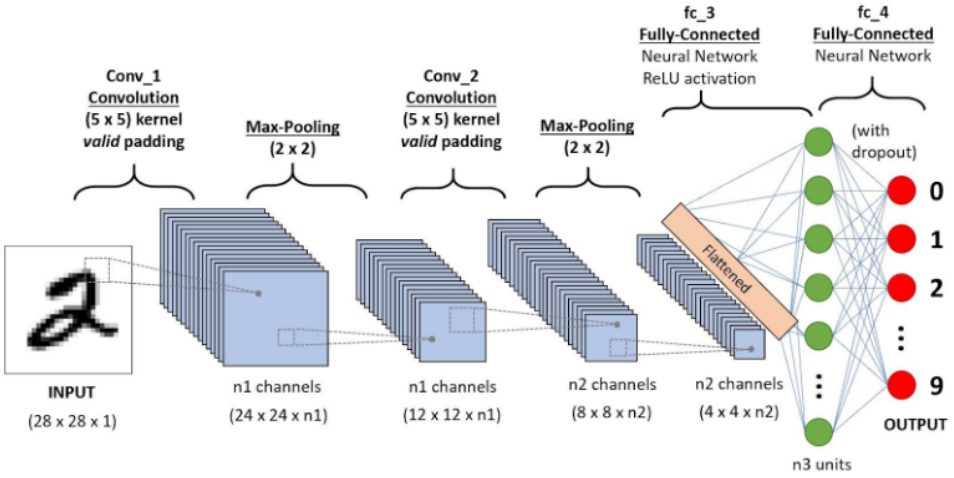
Overfitting: Problema muy común. El modelo está demasiado ajustado a las instancias y no funciona adecuadamente con nuevos casos. El modelo no es capaz de generalizar. Normalmente, fronteras de decisión muy complejas producen sobreajuste.

Además de estos puntos, también se tienen en cuenta: Minimización del error y dificultad de ajuste de par ametros

4.13.1. Redes convolucionales

Las características de los tipos de capas que conforman una red neuronal convolucional nos permiten extraer características (o features) de los datos de entrada, de forma más localizada (conexiones esparsas), con más robustez frente a las variaciones en los datos y reduciendo el número de parámetros necesarios. Tenemos: Capas Convolucionales(CONV) y Capas de Pooling (POOL).

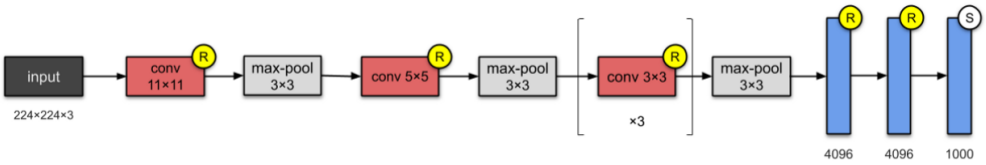
Estas redes son utilizadas en por ejemplo, la clasificación de imágenes. El esquema de las redes para clasificación y, en general, para procesamiento de imágenes, consta de una primera etapa con capas convolucionales y de pooling para la extracción de características, seguida de capas densas para la clasificación final.



Algunas arquitecturas clásicas de las redes convolucionales son:

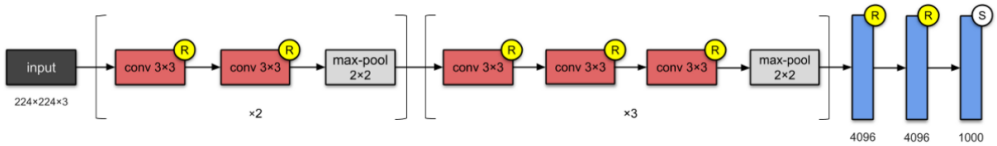
- AlexNet: En 2012 causó un gran impacto por obtener un puntaje significativamente mayor que el segundo puesto en ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), basando su modelo en capas convolucionales y de pooling. A partir de ese momento todos los ganadores comenzaron a ser redes convolucionales profundas.

  Entre las novedades introducidas en esta arquitectura encontramos: Activaciones ReLU (Rectified Linear Units), uso de múltiples GPUs para entrenar el modelo, Dropout, Local Response Normalization(no tan usado hoy en día), capas Pool con ventanas superpuestas, data Augmentation, 60 M de parámetros.



- VGGNet: Las redes VGG-16 y VGG-19 fueron presentadas en 2014 obteniendo el primer y segundo puesto en las tareas de localización y clasificación de la competencia ILSVRC, respectivamente. Estas redes, si bien están basadas en muchos de los principios introducidos por AlexNet, cuentan con algunas características destacables: Utiliza tamaños de filtro de 3x3 a lo largo de toda la red, agrega más capas convolucionales, para reducir la dimensionalidad solo se emplearon capas de Max-Pooling, todas las convolucionales son con stride igual a 1, entrenamiento con multi escalado de imágenes (Multi-Scale Training) y 138M/144M de parámetros.

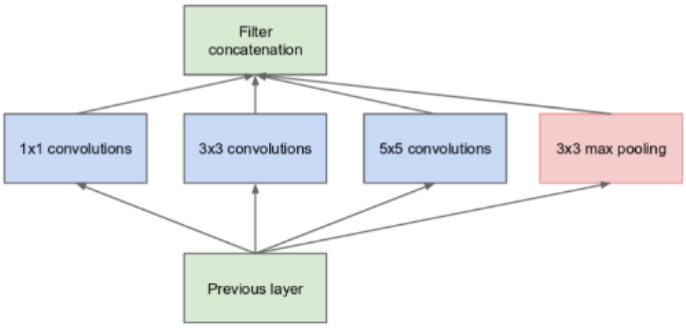
  VGG16:



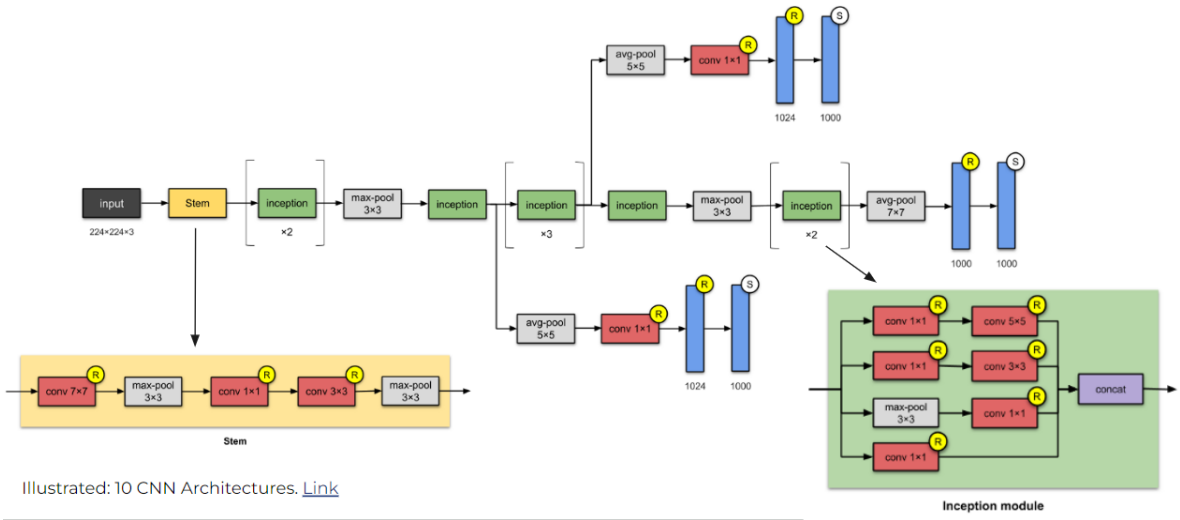
  Para el caso de VGG-19, los bloques de 3 capas convolucionales tenían 4 capas, seguidas de un max-pooling.

- Inception: En 2014 un equipo de Google presentó la primera versión de la serie de redes Inception en la competencia ImageNet, ganando en la tarea de clasificación. Como en todos los casos, la motivación principal estaba en obtener redes cada vez más profundas (más capas y más neuronas), que puedan mejorar las métricas actuales. Sin embargo, esto implicaba más probabilidad de sobreentrenamiento y más costo computacional.

  Bajo estas premisas se creó lo que se conoce como un bloque Inception:

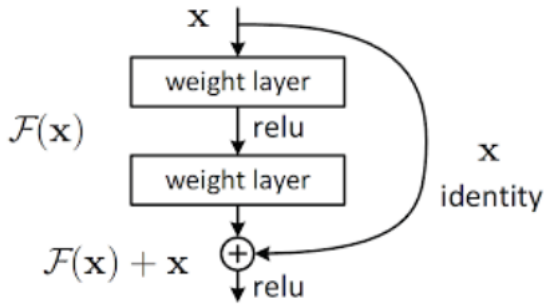


  InceptionV1:

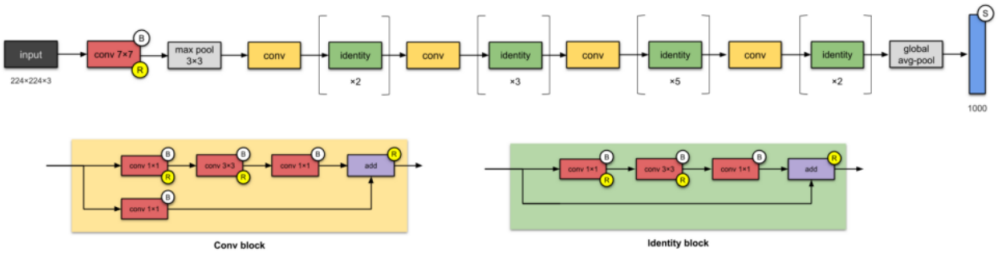


- ResNet: Para 2015 agregar más capas a una red convolucional ya no garantizaba un modelo más preciso. Ese año un equipo de Microsoft introdujo las conexiones residuales en una red neuronal convolucional a través de las redes ResNet y obtuvo el primer lugar en la competencia ImageNet. En su paper se presentan redes de hasta 150 capas que mejoran en las métricas de error a cualquiera de las anteriores.

  Las conexiones residuales son, hoy en día, uno de los conceptos que se siguen aplicando en el desarrollo de redes neuronales y que marcaron un antes y un después en su desempeño.



ResNet-50:



4.13.2. Data Augmentation

Consiste en generar "nuevos" ejemplos de datos de entrenamiento, para darle más variabilidad a nuestro conjunto de datos. Los nuevos datos pueden generarse a partir de los ya existentes, mediante distintos tipos de transformaciones, o pueden sintetizarse para crear ejemplos completamente nuevos. No es una herramienta exclusiva de computer vision, se puede aplicar en otros dominios!

Se utiliza para:

- Incrementar el tamaño del conjunto de datos: Indispensable cuando el conjunto de datos es pequeño y cuando es costoso obtener o etiquetar nuevos datos.
- Evitar el sobreentrenamiento: Busca mejorar la capacidad de los modelos de generalizar sobre los datos.
- Permite balancear las clases del conjunto de datos: Compensa las variaciones en la distribución de los datos.
- Mejorar las métricas del modelo: Enrobustece al modelo frente a variaciones en la distribución de los datos de validación o testeo.

Transformaciones básicas:

También existen transformaciones de color.

Mezcla de imágenes: Esta resulta ser una práctica contraintuitiva desde el punto de vista humano pero hay estudios que demuestran su funcionamiento en la clasificación de imágenes.

Transformaciones basadas en Deep Learning: Una posible técnica es utilizar redes neuronales para generar imágenes que formen parte del conjunto de entrenamiento. Utilizando Cycle-GAN se pueden generar imágenes de tomografías realistas.

Ataques a las redes neuronales: En muchos casos, es factible engañar a este tipo de redes neuronales, obteniendo predicciones erróneas sobre imágenes que, a simple vista, tienen otro contenido.

Ataques adversarios: El entrenamiento adversario puede ayudar a encontrar posibles técnicas de aumentación que ayuden a mejorar los puntos débiles de las redes convolucionales. Son muy importantes cuando las aplicaciones de estos modelos están relacionadas con identificación de personas, por ejemplo.

Precauciones a tener en cuenta:

- Se debe tener cuidado con las transformaciones elegidas porque pueden modificar el valor de las etiquetas según sea el conjunto de datos que estamos utilizando.
- Si nuestro dataset original contiene biases, podemos llegar a arrastrar dichos biases con las modificaciones implementadas.
- Las mejores transformaciones serán aquellas que modifiquen características que no queremos que nuestro modelo aprenda, pero hay que tener cuidado de no romper las etiquetas.
- En general, utilizar Data Augmentation hace que el entrenamiento sea más lento dado que nuestro dataset es más grande. Por lo tanto, siempre se recomienda aumentar las iteraciones de entrenamiento cuando se utilizan este tipo de técnicas.
- Además, algunos métodos de Data Augmentation presentan un tiempo de cómputo considerable, enlenteciendo aún más dicho entrenamiento.
- Utilizar métodos de Data Augmentation basados en Deep Learning requiere un trabajo extra de generación y evaluación de dichos modelos.