Chapter 1

Modelos de IA

Objetivo

El alumno explicará cómo actúan los modelos de IA, aplicándolo en el área de la inteligencia artificial

1.1 Modelos probabilísticos

1.1.1 Modelos de Markov

Una *cadena* o *proceso de Markov*, es un proceso estocástico que satisface la *propiedad de Markov*: sin memoria (*memoryless*). Un proceso satisface esta propiedad si se pueden realizar predicciones basadas únicamente en el estado presente aún cuando sea posible conocer la historia completa del proceso; es decir, es independiente de dicha historia. Reciben el nombre en honor a Andrey Markov (https://bit.ly/293MsgP), quién a inicios del siglo XX se encontraba estudiando procesos de *caminatas aleatorias*.

♢ Un *modelo de Markov* es un modelo estocástico que sirve para representar datos secuenciales (temporales), es decir, datos que están ordenados.

♢ Provee una forma de modelar las dependencias entre información actual e información previa.

♢ Se compone de estados, transiciones entre estados y formas de obtener resultados de salida.

♢ Sirven para diversas aplicaciones:

- Inferir estadísticas a partir de datos secuenciales,
- Predicciones y/o estimaciones,
- Reconocimiento de patrones, etc.

1.1.1.1 Ejemplo

Después de recolectar múltiples datos, se obtuvo el modelo de Markov de la figura 1.1.

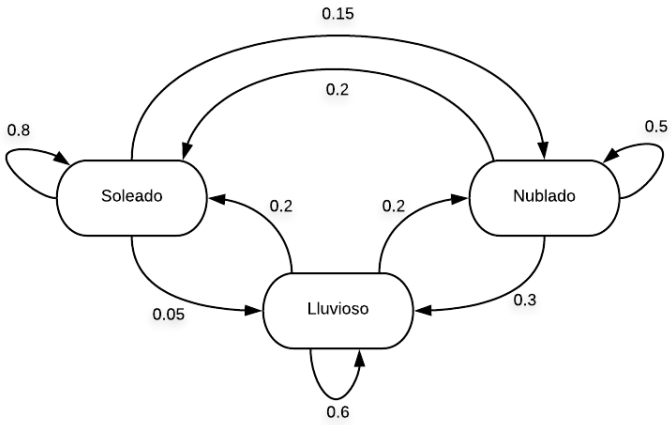

Figure 1.1: Ejemplo de modelo de Markov

Es importante notar que la suma de las transiciones de salida de cada nodo debe ser igual a 1; para el ejemplo:

$$\begin{array}{rclrclrcl} P(S|S) & = & 0.8 & P(S|Ll) & = & 0.2 & P(S|N) & = & 0.2 \\ P(Ll|S) & = & 0.05 & P(Ll|Ll) & = & 0.6 & P(Ll|N) & = & 0.3 \\ P(N|S) & = & 0.15 & P(N|Ll) & = & 0.2 & P(N|N) & = & 0.5 \end{array}$$

Para algún día en específico, el modelo puede estar en cualquiera de los tres estados y puede generarse una secuencia $q_1, q_2, q_3, q_4, q_5, \ldots$, donde $q_i \in \{Soleado, Lluvioso, Nublado\}$.

Es posible aprovechar la propiedad de Markov para calcular la probabilidad del estado del tiempo para días siguientes:

♢ Suponiendo que ayer estuvo lluvioso y hoy está nublado, ¿Cuál es la probabilidad de que mañana esté soleado?:

$$\begin{array}{rcl} P(S|Ll, N) & = & P(S|N) \\ & = & 0.2 \end{array}$$

♢ Dado que hoy está soleado ¿Cuál es la probabilidad de que mañana esté soleado y pasado mañana lluvioso?:

$$\begin{array}{rcl} P(S|S, Ll) & = & P(S|S)\, P(Ll|S) \\ & = & 0.8 \times 0.05 \\ & = & 0.04 \end{array}$$

En este sitio se pueden ver ejemplos de cadenas de Markov en ejecución: http://setosa.io/ev/markov-chains/

1.1.2 Redes Bayesianas

Una red bayesiana es un tipo de modelo gráfico probabilista que utiliza inferencia bayesiana para calcular probabilidades. Su objetivo es modelar dependencia condicional y, por tanto, causalidad utilizando para esto, aristas en una gráfica dirigida. Mediante estas relaciones, es posible realizar

inferencias eficientes sobre las variables aleatorias de la gráfica. El término fue acuñado por Judea Pearl (https://bit.ly/2IEfdTz) en 1985.

Utilizando las relaciones especificadas en nuestra red bayesiana, podemos obtener una representación compacta de la distribución de probabilidad conjunta, aprovechando la independencia condicional. Recordemos que la independencia condicional de dos variables A y B dada otra variable C, es equivalente a satisfacer la siguiente propiedad: $P(A, B|C) = P(A|C) * P(B|C)$. En otras palabras, mientras que el valor de C sea conocido y fijo, A y B son independientes. Otra formulación útil es: $P(A|B, C) = P(A|C)$.

Una red bayesiana es una gráfica acíclica dirigida (DAG por sus siglas en inglés) en la que cada arista corresponde a una dependencia condicional y cada nodo a una única variable aleatoria. Formalmente, si existe una arista (A, B) que conecta las variables aleatorias A y B en la gráfica, significa que $P(A|B)$ es un factor en la distribución de probabilidad conjunta, por lo tanto, debemos conocer $P(A|B)$ para todos los valores de B y A para poder realizar inferencias. En el ejemplo de la figura 1.2, como hay una arista que va de $Lluvia$ a $PastoMojado$, significa que $P(PastoMojado|Lluvia)$ será un factor cuyos valores de probabilidad están especificados junto al nodo $PastoMojado$ en una tabla de probabilidad condicional.

La redes bayesianas satisfacen la *propiedad local de Markov*, que establece que un nodo es condicionalmente independiente de todos sus *no-descendientes* dados sus *padres*. Para el mismo ejemplo, esto significa que $P(Aspersor|Nublado, Lluvia) = P(Aspersor|Nublado)$, debido a que $Aspersor$ es condicionalmente independiente de su no-descendiente $Lluvia$ dado $Nublado$.

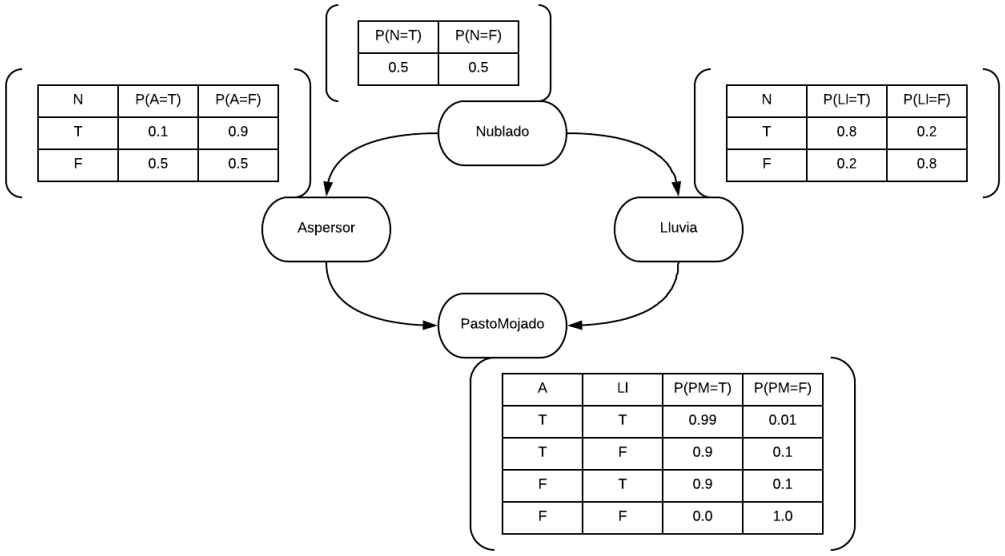

Figure 1.2: Ejemplo de red bayesiana

Esta propiedad permite simplificar la distribución conjunta obtenida: la distribución conjunta para todos los nodos de una red bayesiana se calcula como:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1}) = \prod_{i=1}^{n} P(X_i|Padres(X_i))$$

En redes grandes, esta propiedad permite reducir la cantidad de cómputo requerido, dado que en general los nodos tendrán pocos padres en relación al tamaño completo de la red.

1.1.2.1 Ejemplo

Se tiene la red bayesiana de la figura 1.3.

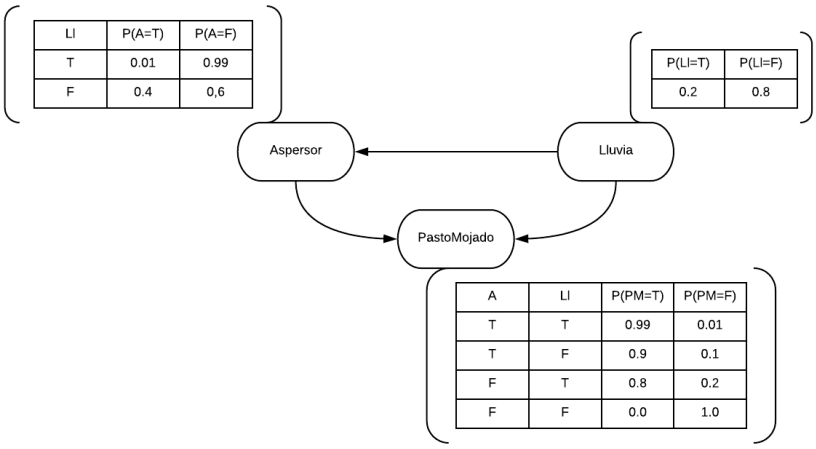

Figure 1.3: Ejemplo de red bayesiana

La función de probabilidad conjunta está dada por: $P(PM, A, Ll) = P(PM|A, Ll)\, P(A|Ll)\, P(Ll)$. Con este modelo se pueden responder preguntas del tipo: "¿Cuál es la probabilidad de que esté lloviendo, dado que el pasto está mojado?" utilizando probabilidad condicional y realizando la sumatoria sobre las variables que no interesan:

$$P(Ll = T|PM = T) = \frac{P(PM = T|Ll = T)}{P(PM = T)} = \frac{\sum_{A\in\{T,F\}} P(PM = T, A, Ll = T)}{\sum_{A,Ll\in\{T,F\}} P(PM = T, A, Ll)}$$

Con ayuda de la expansión de la función de probabilidad conjunta $P(PM, A, Ll)$ y las probabilidades conjuntas indicadas en las tablas de la figura 1.3, se pueden obtener los valores de los sumandos; por ejemplo:

$$\begin{aligned} P(PM = T, A = T, Ll = T) &= P(PM = T|A = T, Ll = T)\, P(A = T|Ll = T)\, P(Ll = T) \\ &= 0.99 \times 0.01 \times 0.2 \\ P_{TTT} &= 0.00198 \end{aligned}$$

Obteniendo los valores para las posibles combinaciones, se obtiene:

$$P(Ll = T|PM = T) = \frac{0.00198_{TTT} + 0.1584_{TFT}}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0.0_{TFF}} = \frac{891}{2491} \approx 35.77\%$$

Tarea:

♢ Para la figura 1.3, determinar:

- La probabilidad de que no esté lloviendo, dado que el pasto está mojado
- La probabilidad de que esté lloviendo, dado que el pasto está seco
- La probabilidad de que los aspersores estén encendidos, dado que el pasto está mojado
- La probabilidad de que los aspersores no estén encendidos, dado que el pasto está mojado
- La probabilidad de que los aspersores estén encendidos, dado que el pasto está seco

♢ Obtener la forma de calcular la función de probabilidad conjunta genérica para la red de la figura 1.2.

*

Programa:

Desarrollar un código que reciba la gráfica que representa un modelo de Markov y genere cadenas de forma similar a las obtenidas en el sitio mostrado en clase (http://setosa.io/ev/markov-chains/).

*

1.2 Modelos bioinspirados

1.2.1 Computación evolutiva: algoritmos genéticos

Existen problemas demasiado complejos para resolver: ejecutar un programa que los resuelva excede la memoria disponible o no terminaría en una cantidad razonable de tiempo. Para estos problemas, una solución puede encontrarse mediante procesos evolutivos que involucrean varias generaciones de soluciones de prueba. La estrategia mencionada es el fundamento de los algoritmos genéticos: esencialmente, un algoritmo genético descubre una solución utilizando comportamiento aleatorio combinado con la teoría reproductiva y el proceso de selección natural.

Un algoritmo genético comienza generando un conjunto aleatorio de soluciones de prueba, cada solución es una suposición. Cada solución de prueba se llama cromosoma y cada componente del cromosoma es un gen.

Como cada cromosoma inicial es una suposición aleatoria, es muy poco probable que represente una solución del problema a resolver. Por tanto, el algoritmo genético genera un nuevo conjunto de cromosomas (descendencia) en el que cada cromosoma es un hijo de dos cromosomas del conjunto anterior (padres). Los padres también se eligen aleatoriamente, pero se asigna mayor probabilidad a aquellos que aparentemente tienen mejor oportunidad de llevar a una solución, de esta forma emulan el principio evolutivo de supervivencia del más apto: determinar qué cromosomas son los mejores candidatos es el paso más problemático en un algoritmo genético. Cada descendiente es una combinación aleatoria de genes de los padres, además, un descendiente puede mutar de alguna forma. Eventualmente, si se repite este proceso muchas veces, se obtienen cada vez mejores soluciones hasta llegar a una suficientemente buena. Desafortunadamente no se tiene seguridad de que el algoritmo genético encontrará una solución al final de este proceso repetido,

pero investigaciones recientes han mostrado que los algoritmos genéticos pueden ser muy efectivos para resolver una gama sorprendemente amplia de problemas complejos.

Cuando se aplican al desarrollo de programas, el enfoque de algoritmos genéticos se conoce como programación evolutiva. Aquí, el objetivo es desarrollar programas permitiéndoles que evolucionen, en lugar de escibirlos. La propuesta es iniciar con una gran colección de funciones: esta colección conforma el "conjunto de genes" de los que las futuras generaciones de programas serán construidos. Después se permite que el proceso evolutivo se ejecute por varias generaciones esperando que cada generación tenga mejor desempeño que la anterior hasta llegar a la mejor solución del problema propuesto.

1.3 Clasificadores

What is the most mathematically involved machine learning algorithm?(https://bit.ly/3MySDwP)

> Over the brief time I've spent studying the mathematics of machine learning, I've come to a realization that might seem absurd to the layman:
>
> > *(Almost) every concept in machine learning gets mathematically more involved, the deeper you study it. Each concept is like a fractal, exhibiting ever increasing layers of complexity as you zoom into the specifics. [*]*

Figure 1.4: ML & Math

1.3.1 Clasificación por *k-medias*

Inventado en 1967 por James B. MacQueen, es uno de los algoritmos de aprendizaje sin supervisión más simples que resuelve el problema de clasificación. El procedimiento sigue una idea simple para clasificar datos entre un número k (predefinido) de clases. Se inicia seleccionando k centroides para cada clase de forma *arbitraria*; es recomendable que se elijan estos k puntos iniciales lo más lejano posible entre ellos. A continuación, se toman todos los datos originales y se asocian a cada clase: al centroide más cercano. Una vez agrupados todos los puntos, se recalculan los centroides entre los puntos de cada clase y se vuelven a agrupar todos los datos con los nuevos centroides. Este procedimiento se repite hasta que no existan cambios en los centroides.

Algoritmo 1.1 k-medias

Seleccionar medias iniciales: $m_1, m_2, \ldots, m_k$
Mientras haya cambio en cualquiera de las medias:
    Usar las medias actuales para clasificar las muestras
    Para $i = 1$ hasta k
        Reemplazar m_i con la media de las muestras de la clase i

Aunque el algoritmo siempre termina, no garantiza que encuentre la configuración óptima para los centroides encontrados: es muy sensible a la selección inicial de las medias estimadas. Además, este algoritmos es *greedy*; minimiza la suma de las distancias entre las clases y sus respectivos centroides.

El algoritmo tiene algunas debilidades:

♢ No se especifica cómo inicializar las medias. Existen varias opciones; una de las más comúnes, es elegir al azar los k valores iniciales; otra (discutiblemente mejor) es seleccionar el valor más cercano (o más lejano) al origen y a partir de éste, elegir a los $k-1$ puntos más lejanos.

♢ El resultado final depende de los valores iniciales y, frecuentemente devuelve soluciones subóptimas. Es recomendable ejecutar el algoritmo para diferentes inicializaciones.

♢ Puede suceder que alguna clase m_i sea vacía y no se pueda actualizar su centroide. Este problema suele ser ignorado.

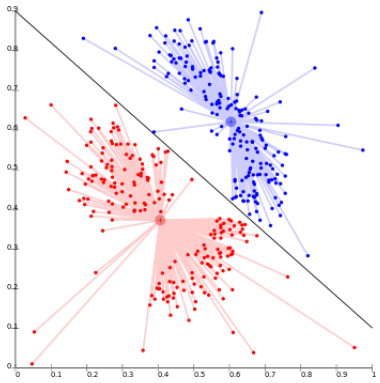

Figure 1.5: Ejemplo con $k = 2$

Ejercicio:

Clasificar las muestras siguientes utilizando $k = 2$:
$[8, 10], [3, 10.5], [7, 13.5], [5, 18], [5, 13], [6, 9], [9, 11], [3, 18], [8.5, 12], [8, 16]$
Utilizando $[8, 10]$ y $[5, 13]$ como medias iniciales.

Tarea: _____

Clasificar las muestras siguientes utilizando $k = 2$:
$[1, 12.5], [3, 10.5], [3, 12.5], [3, 14.5], [3, 18], [5, 18], [5, 16], [5, 14.5], [5, 13], [6, 9], [8, 10], [9, 11],$
$[8.5, 12], [7, 13.5], [8, 16], [0.5, 10.5]$
Utilizando $[3, 14.5]$ y $[9, 11]$ como medias iniciales.
_____ *

1.3.2 Redes neuronales: perceptrón

1.3.2.1 Neuronas artificiales y el modelo de McCulloch-Pitts

La idea original del *perceptrón* se remonta al trabajo de Warren McCulloch (http://bit.ly/1BVMCxa) y Walter Pitts (http://bit.ly/1G7QKkd) en 1943, quienes observaron una analogía entre las neuronas biológicas y compuertas lógicas con salida binaria. Intuitivamente, una neurona puede verse como una subunidad de una red neuronal en un cerebro biológico. Las señales de magnitud variable entran por las dendritas; estas señales se acumulan en el cuerpo de la célula y, si el acumulado sobrepasa el umbral establecido, se genera una señal de salida que se entrega al axón.

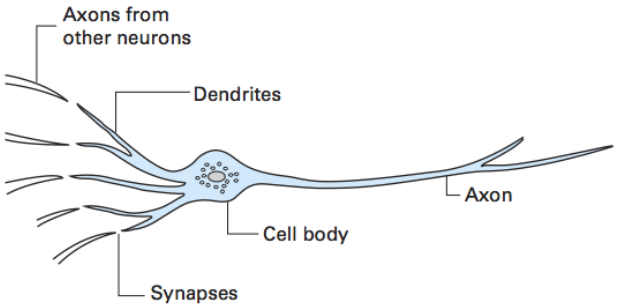

Figure 1.6: Esquema de una neurona biológica

Perceptrón de Frank Rosenblatt Posteriormente, en 1957 Frank Rosenblatt (http://bit.ly/1CBgm5d) publicó el primer algoritmo para un *perceptrón de aprendizaje*. La idea básica es definir un algoritmo que aprende los valores de los pesos w que serán multiplicados con las características de entrada para poder decidir si la neurona *dispara* o no. El perceptrón es un clasificador de aprendizaje *supervisado*, *determinista* y *a posteriori*.

Función escalón Antes de describir a detalle el algoritmo de aprendizaje, definiremos algunos conceptos auxiliares. Primero, llamaremos a las clases positiva y negativa para nuestra clasificación binaria como 1 y -1 respectivamente. A continuación, definimos una función de activación $g(z)$ que toma una combinación lineal de las entradas x y los pesos w como entrada $z = w_1x_1 + \cdots + w_nx_n$ y, si $g(z)$ es mayor que el umbral definido θ, se obtiene 1 y -1 en otro caso. Esta función de activación se conoce como "función escalón unitario" o función escalón de Heaviside.

$$g(z) = \begin{cases} 1 & \text{si } z \geq \theta \\ -1 & \text{en otro caso} \end{cases} \text{ y } z = w_1x_1 + \cdots + w_nx_n = \sum_{i=1}^{n} w_ix_i$$

Además, se suele definir $w_0 = \theta$ y $x_0 = 1$. De este modo:

$$g(z) = \begin{cases} 1 & \text{si } z \geq \theta \\ -1 & \text{en otro caso} \end{cases} \text{ y } z = w_0x_0 + w_1x_1 + \cdots + w_nx_n = \sum_{i=0}^{n} w_ix_i$$

La regla del perceptrón de aprendizaje La idea tras del preceptrón de *umbral* es simular el funcionamiento de una célula en el cerebro: *dispara* o no. En resumen: un perceptrón recibe múltiples señales de entrada y, si la suma de las señales de entrada (multiplicadas por el peso respectivo) sobrepasa cierto umbral, entrega una señal, si no pasa el umbral, queda en *silencio*.

Este es el primer algoritmo de *aprendizaje de máquina*, dada la idea de Frank Rosenblatt, conocida como *regla de aprendizaje*: el perceptrón aprenderá los pesos para cada señal de entrada para poder dibujar un límite de decisión que nos permita discriminar entre dos clases linealmente separables.

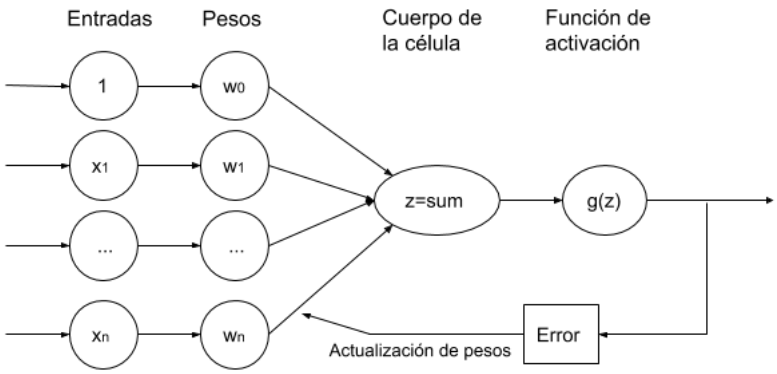

Figure 1.7: Esquema del perceptrón de Rosenblatt

La regla del perceptrón de Rosenblatt es bastante simple y puede resumirse en los pasos del algoritmo 1.2.

Algoritmo 1.2 Regla del perceptrón de Rosenblatt

inicializar los pesos a 0
para cada muestra de entrenamiento $x^{(i)}$:
    calcular el valor de salida $\hat{y}^{(i)}$
    actualizar pesos

El valor de salida es el predicho por la función escalón definida previamente y la actualización del peso se obtiene como $w_j = w_j + \Delta w_j$. El valor para actualizar los pesos en cada incremento se obtiene mediante la regla de aprendizaje:

$$\Delta w_j = \eta \left(salida_{mj} - salida_j \right) x_{ji}$$

Donde η es la tasa (razón) de aprendizaje (una constante entre 0.0 y 1.0); $salida_{mi}$ es la clase a la que pertenece la muestra y *salida* es la salida que predice el perceptrón en el paso actual. Es importante notar que el vector de pesos se actualiza *simultáneamente*.

En particular, para un conjunto de datos de 2 dimensiones, la actualización se obtiene como:

$$\begin{aligned} \Delta w_0 &= \eta \left(y^{(i)} - \hat{y}^{(i)} \right) \\ \Delta w_1 &= \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_1^{(i)} \\ \Delta w_2 &= \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_2^{(i)} \end{aligned}$$

Ejemplo, tarea

Utilizando el valor $\eta = 0.1$, aplicar el algoritmo de aprendizaje del perceptrón para una neurona artificial que calcule la función booleana NAND con 2 parámetros definida como:

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	-1

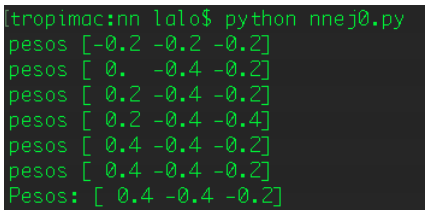

Figure 1.8: Actualizaciones de los pesos en el ejemplo

*

Problemas con el perceptrón El principal problema del perceptrón, el mismo Rosenblatt probó matemáticamente es que la regla de aprendizaje converge si las dos clases pueden separarse linealmente, pero no se puede garantizar su convergencia si no se cumple esta condición.

Programas extra: (0.5 adicional c/u)

♢ Desarollar el algoritmo de la **red bayesiana** en lenguaje de elección libre, pero el algoritmo en sí debe ser desarrollo propio.

♢ Desarollar el algoritmo de **k-medias** en lenguaje de elección libre, pero el algoritmo en sí debe ser desarrollo propio.

♢ Implementar el **algoritmo de aprendizaje del Perceptrón** en lenguaje de elección libre, pero el algoritmo en sí debe ser desarrollo propio.

Nota: Deben seguir los algoritmos presentados en clase, si usan algún otro se notará en el código.

*

1.4 Modelos basados en reglas

1.4.1 Árboles de decisión/regresión

Los árboles de decisión son un tipo importante de modelos predictivos de aprendizaje artificial (*machine learning*). Existen varios algoritmos para este tipo de árboles desde hace varias décadas y algunas variantes como *random forest* se consideran como técnicas muy poderosas en el área de minería de datos (*data mining*).

El término *Classification and Regression Trees* (CART) fue introducido por Leo Breiman (https://bit.ly/2RvIgwl) en 1984 para referirse a árboles de decisión y regresión que pueden usarse como modelos de clasificación o regresión. El algoritmo CART es el fundamento de otros importantes algoritmos tales como *bagged decision trees*, *random forest* y *boosted decision trees* (https://bit.ly/2NtIUXZ). La idea básica es dividir repetidamente los registros disponibles buscando maximizar la *homogeneidad* en los conjuntos obtenidos en dicha división.

Es un modelo jerárquico de aprendizaje que puede ser usado tanto para clasificación como para regresión: permiten explorar relaciones complejas entre entradas y salidas sin necesidad de hacer suposiciones sobre los datos. Los árboles de decisión pueden verse como una función f que realiza una estimación de un *mapeo* desde un espacio de entrada X hacia un espacio objetivo

y: $y = f(X)$. Para el caso de la clasificación, el espacio objetivo y es numérico: $y \in \mathbb{R}$; para la regresión, los valores de y son categóricos: $y \in \{C_1, C_2, \ldots\}$. Se trata de un modelo de aprendizaje supervisado; por tanto, se tienen muestras con la salida esperada. La representación más común para CART es un árbol binario.

Un árbol de decisión divide el espacio X en regiones locales utilizando alguna medida de distancia y el objetivo es determinar particiones bien separadas y homogéneas. Las regiones se dividen de acuerdo a preguntas de prueba y, con esto, se obtiene una división $n-$aria, de forma que mientras se recorre el árbol, en cada nodo se realiza toman decisiones.

Ejemplo 1

La figura 1.9 se muestra un ejemplo sencillo. Los datos se encuentran en dos dimensiones $\{x_1, x_2\}$, también llamadas características, variables ó atributos. Se trata de una clasificación binaria con clases *Redonda* y *Cuadrada*. En el nodo raíz se pregunta si $x_1 > w_{10}$ que divide los datos en un nodo hijo (derecho) con instancias de la clase R y un nodo hijo (izquierdo) que tiene instancias tanto de R como de C. En el hijo izquierdo se realiza una segunda pregunta $x_2 > w_{20}$ para obtener finalmente instancias separadas de las clases R y C. Es importante notar que los resultados obtenidos realizan particiones disjuntas en las variables de entrada.

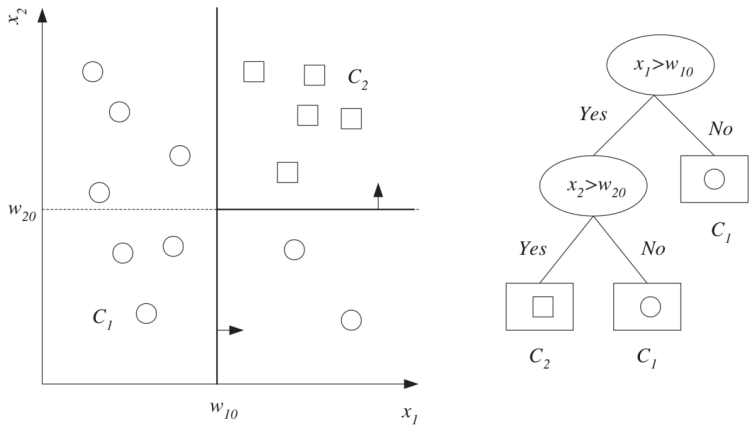

Figure 1.9: Ejemplo de árbol de decisión/regresión (derecha) y sus datos (izquierda)

Ejemplo 2

La figura 1.10 muestra otro ejemplo sencillo para determinar si se debe cargar paraguas o no, de acuerdo a la predicción del estado del tiempo.

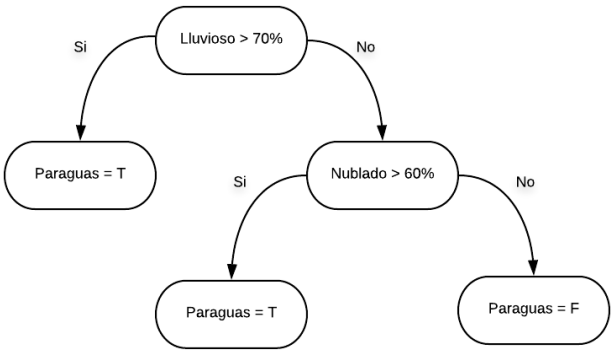

Figure 1.10: Ejemplo de árbol de decisión para llevar paraguas

El algoritmo 1.3 muestra la forma de obtener los valores para las particiones.

Algoritmo 1.3 Obtención de particiones

elegir una variable x_i

- ♢ elegir algún valor s_i para x_i que divida los datos de entrenamiento en dos particiones (no necesariamente iguales)
- ♢ medir la *pureza* (homogeneidad) resultante en cada partición
- ♢ usar otro valor s_j para x_i buscando incrementar la pureza de las particiones hasta alcanzar el umbral de *pureza aceptable*

repetir el proceso de partición con una variable diferente (posiblemente alguna usada previamente)
cada valor obtenido se convierte en un nodo en el árbol

Para determinar la homogeneidad de las particiones, se tienen dos posibilidades:

1.4.1.1 Grado de impureza de Gini

El índice de impureza en un rectángulo A que contiene m clases, se calcula como:

$$I(A) = 1 - \sum_{i=1}^{m} p_i^2$$

Con p la proporción de casos en el rectángulo A que pertenecen a la clase i. Es importante notar que:

- ♢ $I(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.
- ♢ El valor máximo sucede cuando todas las clases se encuentran igualmente representadas (0.5 para el caso binario).

1.4.1.2 Grado de entropía

El grado de entropía en un área A que contiene m clases, se calcula como:

$$E(A) = \sum_{i=1}^{m} p_i \times log_2(p_i)$$

Con p la proporción de casos en A que pertenecen a la clase i. Es importante notar:

♢ $E(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.

♢ El valor máximo $log_2(m)$ sucede cuando todas las clases se encuentran igualmente representadas.

Problemas con CART

♢ Pueden ser poco robustos: un cambio pequeño en los datos de entrenamiento generan grandes cambios en la estructura del árbol.

♢ Obtener el mejor árbol de decisión es $NP - completo$; por tanto, en la práctica para su construcción se utilizan heurísticas basadas en óptimos locales.

♢ No es tan difícil obtener modelos sobreajustados que no generalizan bien las muestras; por esto es necesario establacer un límite en la obtención de particiones.

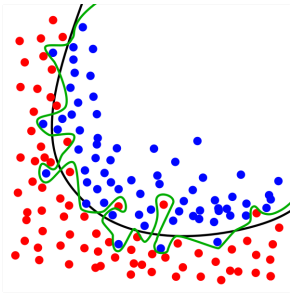

Figure 1.11: Ejemplo de sobreajuste

Ejemplo 3

Para el caso de variables categóricas el proceso es más simple. En la figura 1.12 se observa un ejemplo de datos categóricos para determinar si se debe jugar tennis o no, de acuerdo a la predicción del estado del tiempo.

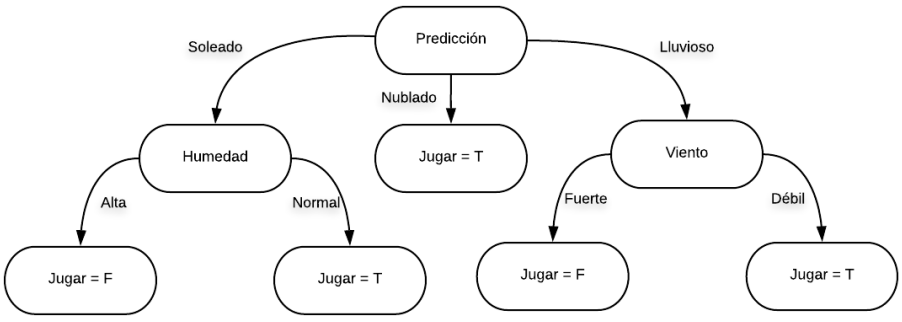

Figure 1.12: Ejemplo de árbol de decisión con variables categóricas

Extracción de reglas

Una vez constuidos, los árboles de decisión son altamente interpretables y es fácil entender la información que representan como reglas del tipo *if - then*: cada camino desde un nodo hasta una hoja es una conjunción de condiciones que deben ser satisfechas para llegar a ese nodo terminal. Para el último ejemplo, tenemos:

- ♢ **if** P=S **and** H=N **then** J=T
- ♢ **if** P=N **then** J=T
- ♢ **if** P=Ll **and** V=D **then** J=T

1.5 Modelos para toma de decisiones