
Inteligencia Artificial

Aprendizaje basado en ejemplos.

In whitch we describe agents that can improve their behavior through diligent study of their own experiences.

¿Porqué queremos que un agente aprenda?

Si es posible un mejor diseño, **¿porqué no lo diseñamos mejor desde el principio?**

Aprendizaje basado en ejemplos.

In whitch we describe agents that can improve their behavior through diligent study of their own experiences.

¿Porqué queremos que un agente aprenda?

Si es posible un mejor diseño, **¿porqué no lo diseñamos mejor desde el principio?**

1. No es posible anticipar todas las situaciones
2. No se puede anticipar todos los cambios
3. El programador no tiene ni idea de como programar una solución

Componentes a aprender.

1. Mapeo de condiciones de un estado a acciones
2. Inferir propiedades relevantes del mundo desde la secuencia de percepciones
3. Como impactan las acciones
4. Utilidad de los estados
5. Información acerca de las preferencias de las acciones

Representación del conocimiento.

- **Representación factorizada** como entradas (un vector de atributos)
- **Aprendizaje inductivo** es aprender una función general desde casos específicos.

Feedback. ¿es necesario?

Feedback. ¿es necesario?

3 tipos de feedback:

1. **Aprendizaje no supervisado:** aprende sin feedback. Lo más común son algoritmos de **clustering**
2. **Aprendizaje por refuerzo:** el agente aprende a través de una serie de refuerzos (recompensas o castigos)
3. **Aprendizaje supervisado:** el agente cuenta con entradas y las salidas esperadas y aprende una función para realizar el mapeo.

Aprendizaje semi-supervisado es un gris entre 1 y 3. Se debe al ruido o a la falta de etiquetas.

Aprendizaje supervisado.

- Dado un **Conjunto de entrenamiento** *(x1, y1), (x2, y2),..., (xn, yn)* donde cada *yj* es generado mediante **y = f(x)**, donde *f* es desconocida; se trata de encontrar un *h(x)* que aproxime a *f*.
- *x* e *y* pueden adoptar cualquier tipo de datos, *xj* es un vector de atributos
- **h** es una **hipótesis**. El aprendizaje consiste en buscar una hipótesis que se ajuste bien a los datos.
- Un **conjunto de test** es usado para medir la **precisión** de una hipótesis.
- Decimos que una hipótesis **generaliza** bien, cuando predice correctamente con entradas no conocidas.
- Cuando la salida es un valor de un conjunto finito, el problema es llamado **clasificación**.
- Cuando la salida es un número el problema es llamado **regresión**.

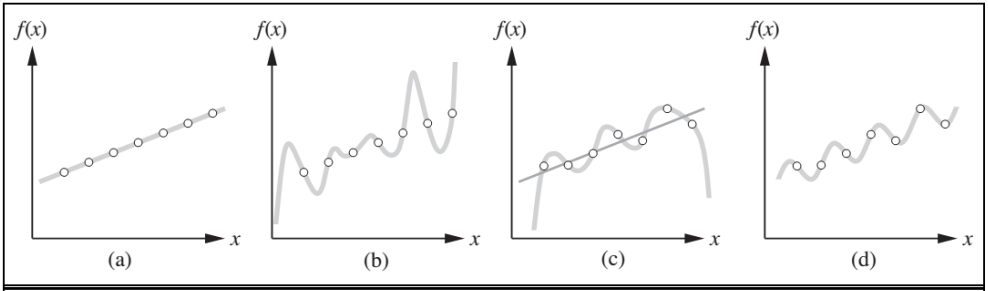


Figure 18.1 **FILES: figures/xy-plot.eps (Tue Nov 3 16:24:13 2009).** (a) Example $(x, f(x))$ pairs and a consistent, linear hypothesis. (b) A consistent, degree-7 polynomial hypothesis for the same data set. (c) A different data set, which admits an exact degree-6 polynomial fit or an approximate linear fit. (d) A simple, exact sinusoidal fit to the same data set.

- *h* es seleccionada desde el **espacio de hipótesis H**
- una hipótesis es consistente si se ajusta a **todos** los datos (18.1 a, b y d)
- **¿Qué hipótesis usamos?** *La navaja de Ockham*

- Punto de equilibrio entre hipótesis complejas que se ajustan a los datos e hipótesis simples que generalizan mejor (18.1 c)
- Un problema es **realizable** si el espacio de hipótesis contiene a la función real
- Hay un compromiso entre la expresividad del espacio de hipótesis y la complejidad de hallar una buena hipótesis.

Arboles de decisión.

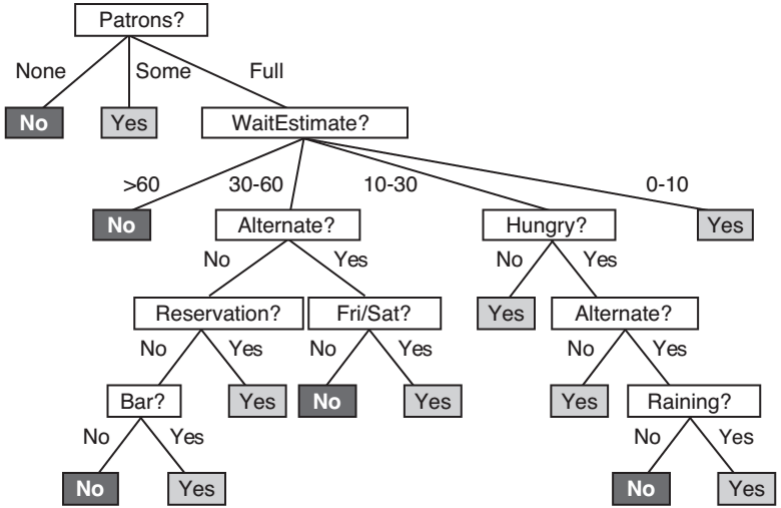
- Representa una función que toma como entradas un vector de atributos y devuelve un único valor.
- Las entradas y salida pueden ser discretas o continuas. Si solo 2 posibilidades para la salida se denomina **clasificación binaria**
- El árbol determina su decisión realizando una serie de tests.
    - Cada nodo interno es un test sobre los valores de uno de los atributos.
    - Cada arista tiene las distintas opciones de la condición
    - Cada nodo hoja es el valor que retorna la función.

Ejemplo.

- El objetivo es aprender como decidir el predicado *WillWait*.
- WillWait se denomina **predicado meta**

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range ($, $$, $$$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
10. *WaitEstimate*: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

Un posible árbol de decisión para este ejemplo.



Expresividad de los árboles de decisión.

- Cualquier función en lógica proposicional se puede expresar como un arbol de decisión booleano

Goal <=> (path1 v path2 v ...)

- Algunas funciones se pueden representar más fácil que otras (ej. mayoría)
- Si tenemos un conjunto de funciones booleanas con n atributos (booleanos tb); ¿Cuántas hipótesis tiene el espacio de hipótesis?

Expresividad de los árboles de decisión.

- Cualquier función en lógica proposicional se puede expresar como un arbol de decisión booleano

Goal <=> (path1 v path2 v ...)

- Algunas funciones se pueden representar más fácil que otras (ej. mayoría)
- Si tenemos un conjunto de funciones booleanas con n atributos (booleanos tb); ¿Cuántas hipótesis tiene el espacio de hipótesis?
- Cada función es una tabla de verdad.
- Una tabla de verdad tiene 2 ^ n filas; donde cada una de estas filas puede tener 2 valores para la misma
- Nos quedan 2 ^ 2 ^ n tablas de verdad distintas

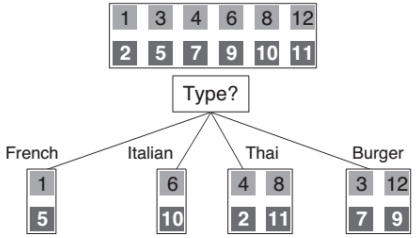
Induciendo áboles de decisión.

- Queremos encontrar un arbol de decisión que sea consistente con el conjunto de entrenamiento y lo más chico posible
- Con algunas heurísticas se pueden encontrar árboles chicos (no el **más chico**)
- DECISION-TREE-LEARNING usa un enfoque **divide y venceras**. Selecciona el atributo más importante y genera subarboles.
- Los ejemplos son cruciales para la construcción del árbol, pero los mismos no aparecen en el mismo.

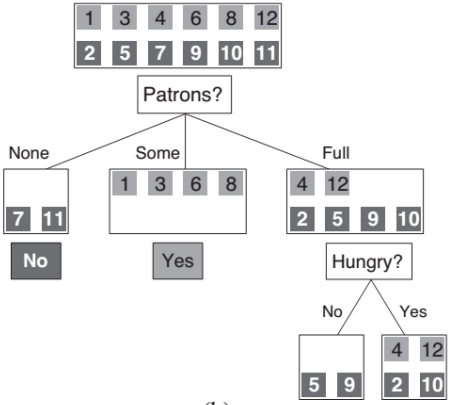
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

Se considera **más importante** a un atributo que hace la mayor diferencia de clasificación.



(a)

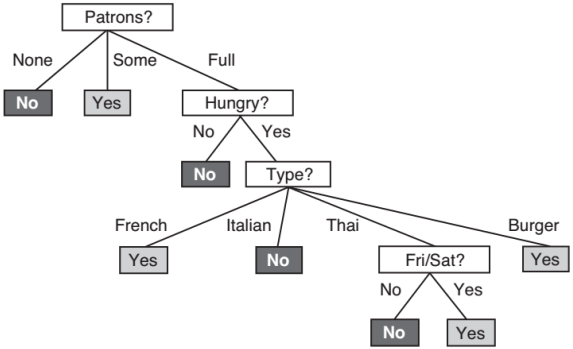


(b)

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) **returns** tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v_k\}$
    $subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
    add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
  return $tree$

El resultado del algoritmo.



Midiendo la precisión del algoritmo. Curvas de aprendizaje

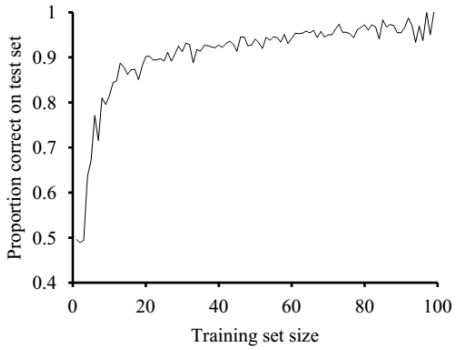


Figure 18.7 **FILES: .** A learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.

Selección de atributos.

- La idea es elegir un atributo que **mejor clasifique** los ejemplos.
  - Atributo perfecto: divide los ejemplos en conjuntos donde todos los ejemplos son positivos o negativos
  - Atributo inútil: divide en conjuntos donde los ejemplos se clasifican en igual proporción. (*Type*)
- *Patrons* no es perfecto pero es bastante bueno
- Una medida formal se puede obtener mediante la noción de **Ganancia de información**, la cual se define en terminos de entropía
- **Entropía** es la medida de incertidumbre de una variable aleatoria. La ganancia de información reduce la entropía.
  - Una moneda que cae siempre de cara, tiene entropía 0

- Una moneda común tiene 1 bit de entropía
- Un dado de cuatro caras tiene 2 bits de entropía
- Una moneda que cae de cara el 99% de las veces tiene menos entropía que una moneda común (cercana de 0)

La entropía sobre una variable **V** con valores **vk** la podemos calcular:

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k) .$$

$$H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$$

$$H(Loaded) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits.}$$

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

$$H(Goal) = B\left(\frac{p}{p+n}\right)$$

El remanente de entropía luego de testear un atributo **A** con **d** valores es:

$$remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B(\frac{p_k}{p_k + n_k})$$

La ganancia de información por testear **A** es la reducción de entropía.

$$Gain(A) = B(\frac{p}{p+n}) - remainder(A)$$

$$Gain(Patrons) = 1 - [\frac{2}{12} B(\frac{0}{2}) + \frac{4}{12} B(\frac{4}{4}) + \frac{6}{12} B(\frac{2}{6})] = .0541 \text{bits}$$

$$Gain(Type) = 1 - [\frac{2}{12} B(\frac{1}{2}) + \frac{2}{12} B(\frac{1}{2}) + \frac{4}{12} B(\frac{2}{4}) + \frac{4}{12} B(\frac{2}{4})] = 0 \text{ bits}$$

Generalización y sobreentrenamiento.

- **Sobreentrenamiento:** es un fenómeno que se da en cualquier algoritmo de aprendizaje.

Se produce cuando el modelo se adapta perfectamente a los patrones de entrada pero es incapaz de generalizar.

- Posibles soluciones:
  - Poda del árbol: elimina test irrelevantes
  - Early stop: Corta la ejecución del algoritmo cuando quedan solo atributos irrelevantes.

Este último implica menos trabajo pero no permite reconocer situaciones donde la combinación de atributos es relevante. (XOR)

Ampliando la aplicabilidad de los AD.

Se necesita poder manejar estos aspectos para poder aplicarlos en aplicaciones reales:

- **Datos faltantes**
- **Atributos multivaluados** problemas con la ganancia de información
- **Atributos enteros o continuos:** puntos de división
- **Atributos de salida continuos:** árboles de regresión

Los AD es generalmente el primer método que se trata de aplicar a un dataset. Una propiedad importante de los AD es que son comprensibles por un ser humano

Seleccionando la mejor hipótesis

Queremos una hipótesis que se comporte **bien** con **datos futuros**.

- **Datos futuros** -> Asumsión de **estacionaridad**
- **bien** -> definimos **error rate** como la proporción de errores que comete.
  - **Holdout cross-validation**: separa los datos en training y test. No se aprende de todos los ejemplos
  - **k-fold cross-validation**: separa los datos en *k* subconjuntos. Se hacen *k* entrenamientos, siempre tomando 1 conjunto para test.
  - **leave-one-out cross-validation**: k = 1
- Los datos se entrenan con el conjunto de training; se elige la mejor hipótesis con los datos de test y se informa la tasa de error del conjunto de validación.

Selección del modelo.

La selección de la mejor hipótesis se divide en 2 tareas:

- Elegir el espacio de estados.
- Encontrar la mejor hipótesis dentro del espacio.

Una manera es parametrizar los modelos de acuerdo al *tamaño*.

function CROSS-VALIDATION-WRAPPER($Learner, k, examples$) **returns** a hypothesis

  local variables: $errT$, an array, indexed by $size$, storing training-set error rates
  $errV$, an array, indexed by $size$, storing validation-set error rates
  for $size = 1$ **to** ∞ **do**
    $errT[size], errV[size] \leftarrow$ CROSS-VALIDATION($Learner, size, k, examples$)
    if $errT$ has converged **then do**
      $best\_size \leftarrow$ the value of $size$ with minimum $errV[size]$
      return $Learner(best\_size, examples)$

---

function CROSS-VALIDATION($Learner, size, k, examples$) **returns** two values:
  average training set error rate, average validation set error rate

  $fold\_errT \leftarrow 0$; $fold\_errV \leftarrow 0$
  for $fold = 1$ *to* k *do*
    $training\_set, validation\_set \leftarrow$ PARTITION($examples, fold, k$)
    $h \leftarrow Learner(size, training\_set)$
    $fold\_errT \leftarrow fold\_errT +$ ERROR-RATE($h, training\_set$)
    $fold\_errV \leftarrow fold\_errV +$ ERROR-RATE($h, validation\_set$)
  return $fold\_errT/k, fold\_errV/k$

From error rate to loss.

• función loss: define la pérdida de utilidad debido a un error

L(x, y, y') = Utilidad(x, y) - Utilidad(x, y')

• L1(y, y') = |y - y'|

• L2(y, y') = (y - y')^2

• L0/1 (y, y´) = 0 if y == y' else 1

$$EmpLoss_{L,E}(h) = \frac{1}{N} \sum_{(x,y) \in E} L(y, h(x)) \ .$$

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} EmpLoss_{L,E}(h) \ .$$

Regularización.

- Selección del modelo penalizando las hipótesis complejas

$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h)$$
$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} Cost(h)\,.$$

Bibliografía y enlaces útiles.

- Russell S., Norvig P.: Artificial Intelligence: A modern Approach. Third Edition. Chapter 18.