Clasificación de patrones: Métodos supervisados

Jordi Porta Zamorano

Escuela Politécnica Superior — **Dept. de Lingüística Computacional**

Universidad Autónoma de Madrid — **Real Academia Española**

jordi.porta@uam.es — **porta@rae.es**

abril de 2005

Máxima verosimilitud (ZeroR)

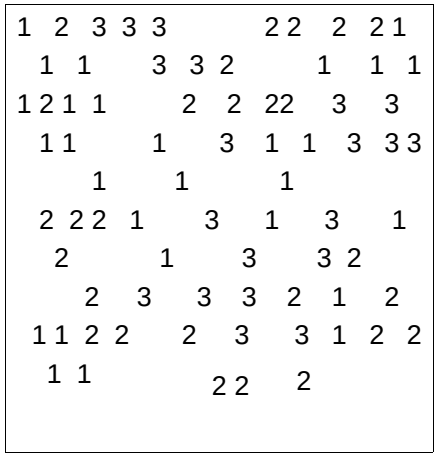
- es el método más simple
  - cuando la clase es numérica se predice la *media*
  - cuando la clase es nominal se predice la *moda* (el valor más frec.)
- sirve como *baseline* para comparar con otros métodos
  - si un método es peor que este, posiblemente haya *sobreajuste*

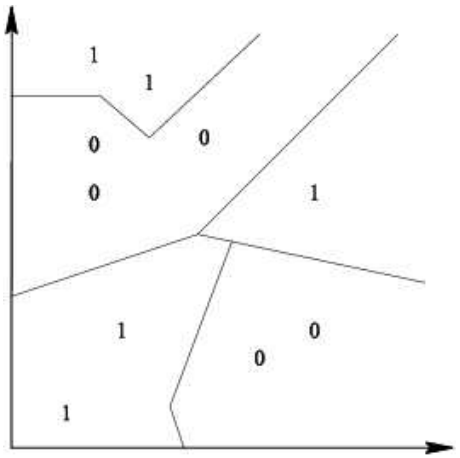
Vecinos próximos (IB1/IBk)

- métodos de *aprendizaje basados en ejemplos* (*instance-based learning*)
- los ejemplos de entrenamiento se almacenan *tal cual*
- se usa una función de distancia para determinar que patrón del conjunto de entrenamiento está más cerca el patrón a clasifi car:
  - para valores numéricos
    - distancia euclídea: $d(\vec{x},\vec{y}) = \sqrt{\sum_i (x_i - y_i)^2}$
    - distancia de Manhattan: $d(\vec{x},\vec{y}) = \sum_i |x_i - y_i|$ (*)
    - el efecto de las diferentes escalas se elimina normalizando: $v' = \frac{v - \text{mín}\, v_i}{\text{máx}\, v_i - \text{mín}\, v_i}$ (*)
    - desconocidos (?): $d(?,?) = 1$ / $d(?,v) = v$ ó $1 - v$
  - para valores nominales:
    - distancia de Hamming: 0 para valores iguales / 1 para valores distintos
    - desconocidos: 1
- la clase del patrón más cercano será asignado a la clase del patrón desconocido (IB1)
- es fácil que se degrade si hay patrones erróneos (ruido)
- podéis ver una demo en http://ilk.uvt.nl/~antalb/ltuia/

Vecinos próximos

- ejemplo / *diagrama de Voronoi*:





Vecinos próximos

- El método se puede ampliar a k vecinos (IBk) y la clase asignada puede obtenerse
  - por mayoría
  - asignando una probabilidad a posteriori:

    $$p(C_i|\vec{x}) = \frac{N_i}{k}$$

    donde C_i es el valor de la clase i y N_i el número de patrones vecinos con clase C_i.
- utilizar todos los patrones equivale a calcular la probabilidad a priori ($\equiv$ método máx. verosimilitud)
- puede ser difícil encontrar la k óptima
- cuantos menos vecinos, más sobreajuste
- el mejor método de validación es *leave-one-out*

Naïve Bayes

Preliminares: probabilidad y estadística descriptiva

- probabilidad: $\frac{\text{casos favorables}}{\text{casos posibles}}$
- probabilidad conjunta: $p(A \cap B) = p(A) \cdot p(B|A) = p(B) \cdot p(A|B)$
- probabilidad condicionada (*a posteriori*): $p(A|B) = \frac{p(A \cap B)}{p(B)}$
- si A y B son independientes $\implies p(A \cap B) = p(A) \cdot p(B)$
- teorema de Bayes: $p(B|A) = \frac{p(A|B) \cdot p(B)}{p(A)}$
- moda: el valor más frecuente
- media y varianza: $\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$, $\sigma_x^2 = var(x) = \frac{\sqrt{\sum_{i=1}^{N} (\bar{x} - x_i)^2}}{N}$

Naïve Bayes

- es un *gold-standard* de los métodos de aprendizaje
- basado en *regla de Bayes*:

  $$p(H|E) = \frac{p(E|H) \cdot p(H)}{p(E)}$$

  donde
  - $p(H)$ es la *probabilidad a priori* de la hipótesis H
  - $p(E)$ es la probabilidad de la evidencia E
  - $p(E|H)$ es la *probabilidad condicionada* de la evidencia E dada la hipótesis H
- supone que el efecto del valor de un atributo sobre una clase es independiente de los valores de los otros atributos
- la suposición de independencia, aunque es bastante fuerte (*naïve*) y a menudo no pueda aplicarse, permite simplifi car el modelo para que no requiera una enorme cantidad de observaciones para cada combinación de atributos
- dado un $\vec{x}$ se clasifi cará en la clase C_i que maximize:

$$p(C_i|\vec{x}) = \frac{p(\vec{x}|C_i) \cdot p(C_i)}{p(\vec{x})} = \frac{\prod_{j=1}^{n} p(x_j|C_i) \cdot p(C_i)}{\prod_{j=1}^{n} p(x_j)} = \frac{p(x_1|C_i) \cdot \ldots \cdot p(x_n|C_i) \cdot p(C_i)}{p(x_1) \cdot \ldots \cdot p(x_n)}$$

Naïve Bayes

```
@relation contact-lenses-recomendation
@attribute age                  young, pre-presbyopic, presbyopic
@attribute spectacle-prescrip   myope, hypermetrope
@attribute astigmatism          no, yes
@attribute tear-prod-rate       reduced, normal
@attribute contact-lenses       soft, hard, none
@data
young,myope,no,reduced,none
young,myope,no,normal,soft
young,myope,yes,reduced,none
young,myope,yes,normal,hard
...
```

age	soft	hard	none
young	2	2	3
pre-presbyopic	1	1	5
presbyopic	1	1	4

spectacle-pr.	soft	hard	none
myope	1	3	5
hypermetrope	3	1	7

astigmatism	soft	hard	none
yes	0	4	6
no	4	0	6

tear-prod-rate	soft	hard	none
reduced	0	0	10
normal	4	4	2

$\vec{x}$ =(young,hypermetrope,no,normal)

$p(\text{age=young}\|\text{cl=soft}) = 2/4$	$p(\text{age=young}\|\text{hard}) = 2/4$	$p(\text{age=young}\|\text{none}) = 3/12$
$p(\text{sp=hyper}\|\text{cl=soft}) = 2/4$	$p(\text{sp=hyper}\|\text{hard}) = 1/4$	$p(\text{sp=hyper}\|\text{none}) = 5/12$
$p(\text{ast=no}\|\text{cl=soft}) = 2/4$	$p(\text{ast=no}\|\text{hard}) = 0/4$	$p(\text{ast=no}\|\text{none}) = 6/12$
$p(\text{tpr=normal}\|\text{cl=soft}) = 2/4$	$p(\text{tpr=normal}\|\text{hard}) = 4/4$	$p(\text{tpr=normal}\|\text{none}) = 2/12$
$p(\text{cl=soft}) = 4/20$	$p(\text{hard}) = 4/20$	$p(\text{none}) = 12/20$

$$p(\text{soft}|\vec{x}) = \frac{p(\text{age=young}|\text{soft}) \times p(\text{sp=hyper}|\text{soft}) \times p(\text{ast=no}|\text{soft}) \times p(\text{tpr=normal}|\text{soft}) \times p(\text{soft})}{p(\vec{x})}$$
$$= \frac{2/4 \times 2/4 \times 2/4 \times 2/4 \times 4/20}{p(\vec{x})}$$
$$= \frac{0{,}0125}{p(\vec{x})}$$

$$p(\text{none}|\vec{x}) = \frac{p(\text{age=young}|\text{none}) \times p(\text{sp=hyper}|\text{none}) \times p(\text{ast=no}|\text{none}) \times p(\text{tpr=normal}|\text{none}) \times p(\text{none})}{p(\vec{x})}$$
$$= \frac{3/12 \times 5/12 \times 6/12 \times 2/12 \times 12/20}{p(\vec{x})}$$
$$= \frac{0{,}0052}{p(\vec{x})}$$

$$p(\text{hard}|\vec{x}) = \frac{p(\text{age=young}|\text{hard}) \times p(\text{sp=hyper}|\text{hard}) \times p(\text{ast=no}|\text{hard}) \times p(\text{tpr=normal}|\text{hard}) \times p(\text{hard})}{p(\vec{x})}$$
$$= \frac{2/4 \times 1/4 \times 0/4 \times 4/4 \times 4/20}{p(\vec{x})}$$
$$= \frac{0}{p(\vec{x})} = 0!!!!$$

Naïve Bayes

- podría haber *ceros* en todas los cálculos $\Longrightarrow$ *suavizado* (*smoothing*)
  - *estimador de Laplace* (o *adding one*):

0	1	4	$\rightarrow$	0+1	1+1	4+1
4	5	0		4+1	5+1	0+1

  - ...
- cuando en valor de un atributo es desconocido, no interviene en los cálculos
- para atributos numéricos, se asume (de manera naïve) que su *distribución de probabilidad* es una *normal*: ...

Modelos lineales

- Los modelos lineales han sido usados durante décadas tanto intensiva como extensivamente en aplicaciones estadísticas y son los más naturales a considerar cuando tanto la clase, como los atributos son numéricos. La clase se expresa como una combinación lineal de atributos:

$$\hat{y} = w_0 + w_1 x_1 + \ldots + w_n x_n$$

- Dado un conjunto de parámetros (w_j) se calcula el error del modelo como la suma de los cuadrados de la diferencia entre el valor predicho ($\hat{y}$) y el observado (y) sobre el conjunto de test:

$$\sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

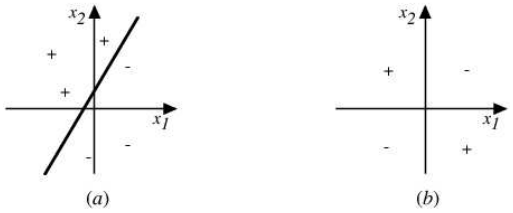
- La estimación del valor óptimo de los parámetros w_j del modelo lineal, se obtiene por minimización de la función de error anterior usando métodos para regresión lineal y puede encontrarse en cualquier manual de estadística.

Modelos lineales

- El modelo lineal puede ser aplicado a problemas de clasificación en que los valores de las clases sean categoriales mediante *regresión lineal multirespuesta* discretizando los valores de la clase y construyendo un modelo lineal para cada componente de vector característico.
- Suponiendo un problema con tres valores c_1, c_2 y c_3 para la clase, los vectores característicos de cada uno serían respectivamente: (1,0,0), (0,1,0) y (0,0,1). Se construirían tres modelos lineales que al aplicarse darían para cada patrón un predicción ($\hat{c_1}$, $\hat{c_2}$, $\hat{c_3}$) cuya componente mayor indicaría la clase del patrón que se clasifica.
- Problemas:
  - el número de patrones debe ser mucho mayor que el número de atributos
  - la naturaleza lineal de las dependencias entre la clase y los atributos

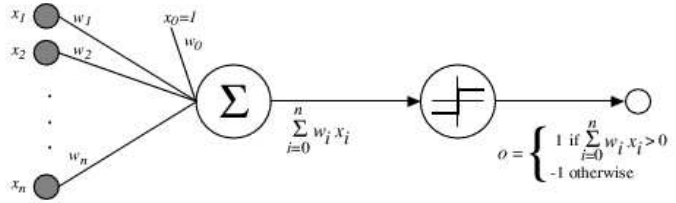
Perceptrón y Winnow

- algoritmos eficientes que manejen:
  - millones de patrones de entrenamiento
  - miles de atributos
- trabajan con clases binarias
- los algoritmos de entrenamiento:
  - son *online*: procesan ejemplo a ejemplo, la clase puede cambiar
  - están dirigidos por el error
  - convergen para problemas *linealmente separables* (a):

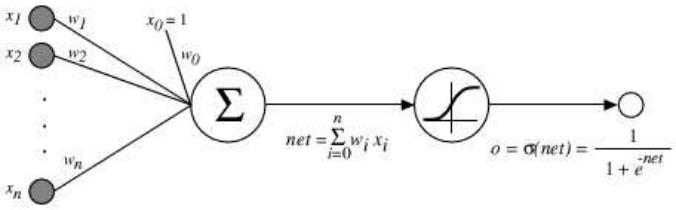


Perceptrón

- activación por la función *escalón* o la función *signo*:



- activación por la función *sigmoide*:



Perceptrón

- Entrenamiento:

$\vec{w} = \vec{0}$; {pesos}
$\theta = 0$; {umbral (threshold)}
while haya errores **do**
  for all $\vec{x}_j \in$ *training set* **do**
    $\hat{c} =$ *classify*$(\vec{x}_j, \vec{w}, \theta)$;
    if *class*$(\vec{x}_j) =$ *yes* $\wedge\ \hat{c} =$ *no* **then**
      $\theta \leftarrow \theta - 1$;
      $\vec{w} = \vec{w} + \vec{x}_j$;
    else if *class*$(\vec{x}_j) =$ *no* $\wedge\ \hat{c} =$ *yes* **then**
      $\theta \leftarrow \theta + 1$;
      $\vec{w} = \vec{w} - \vec{x}_j$;
    end if
  end for
end while

Perceptrón

- Clasificación:

  if $\vec{w} \cdot \vec{x} > \theta$ **then**
    return yes
  else
    return no
  end if

- producto escalar de vectores (*inner product*):

$$\vec{w} \cdot \vec{x} =< \vec{w}, \vec{x} >= \sum_{i=1}^{m} w_i \cdot x_i = w_1 \cdot x_1 + \ldots + w_m \cdot x_m$$

Winnow

- Entrenamiento:

  $\vec{w} = \vec{1}$;
  $\theta = \frac{n}{2}$;
  while haya errores **do**
    for all $\vec{x}_j \in$ *training set* **do**
      $\hat{c} =$ *classify*$(\vec{x}_j, \vec{w}, \theta)$;
      if *class*$(\vec{x}_j) =$ *yes* $\wedge\ \hat{c} =$ *no* **then**
        for all i tal que $x_i = 1$ **do**
          $w_i = w_i \cdot 2$;
        end for
      else if *class*$(\vec{x}_j) =$ *no* $\wedge\ \hat{c} =$ *yes* **then**
        for all i tal que $x_i = 1$ **do**
          $w_i = w_i/2$;
        end for
      end if
    end for
  end while

- Clasificación: igual que para el perceptrón

Perceptrón y Winnow

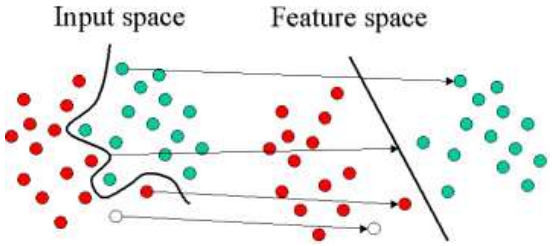
- Winnow funciona bien bajo la presencia de atributos irrelevantes, ruido y clases cambiantes:
- Winnow es mejor cuando el vector de pesos es *disperso* (*sparse*)
- el perceptrón es mejor cuando los ejemplos son dispersos en el espacio de atributos

SNoW

- SNoW (Sparse Network of Winnow-based classifiers)
- extiende el modelo Winnow a problemas de clasificación con más de dos clases
- se usa un modelo Winnow para cade clase y se combinan las decisiones los modelos
- un ejemplo de entrenamiento de clase C es:
  - un ejemplo positivo para el modelo Winnow asociado a la clase C
  - un ejemplo negativo para el resto de modelos Winnow
- todos los atributos deben binarizarse

Máquinas de soporte vectorial (SVM)

- son una extensión de los modelos lineales que se aplican a problemas con dos clases
- el espacio de entrada se transforma en un nuevo espacio con nuevos atributos obtenidos por combinación no lineal de los atributos originales
- el modelo lineal obtenido en el nuevo espacio representa un límite de decisión no lineal en el espacio original
- el modelo lineal buscado es el *hiperplano de máximo margen*: el hiperplano que da la máxima separación entre clases (evita el sobreajuste)



Máquinas de soporte vectorial (SVM)

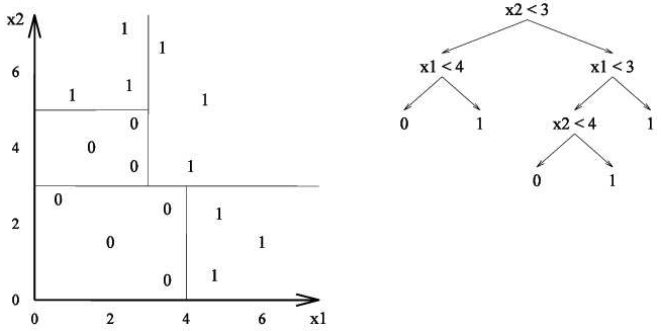
- para transformar el espacio de entrada se utilizan *kernels*:
  - kernel polinómico: $(x \cdot y)^n$
    - el conjunto original de atributos x_1 y x_2, puede ser reemplazados por todos los productos de tres factores (n=3): x_1^3, $x_1^2 x_2$, $x_1 x_2^2$ y x_2^3 en el modelo lineal:

$$\hat{y} = w_1 x_1^3 + w_2 x_1^2 x_2 + w_3 x_1 x_2^2 + w_4 x_2^3$$

    - con diez atributos, si se incluyen todos los productos con 5 factores, hay que determinar más de 2000 coefi cientes
    - los polinomios de un grado sufi ciente pueden aproximar cualquier límite
    - se empieza con $n = 1$ y se incrementa hasta que el error deja de disminuir
  - kernel de base radial: implementa un tipo de red neuronal
  - kernel sigmoide

Árboles y reglas de decisión

La contrucción de un árbol de decisión a partir de un conjunto de entrenamiento puede expresarse de manera recursiva: Se elije un atributo como nodo raíz y se crean tantas ramas como valores tenga ese atributo. Se divide el conjunto de entrenamiento en distintos subconjuntos según sea el valor del atributo elegido y se repite el proceso para cada uno de ellos.



Árboles de decisión

función *inducir-árbol-de-decisión*(*ejemplos*, *atributos*, *clase-mayoritaria*);
if no hay ejemplos **then**
  return hoja con la *clase-por-defecto*;
else if todos los ejemplo tienen la misma clase **then**
  return hoja con la clase que tienen todos;
else
  mejor ← *elegir-mejor-atributo*(*atributos*, *ejemplos*);
  árbol ← nodo con atributo mejor;
  for all valor v del atributo *mejor* **do**
    ejemplos-con-v ← $\{$*ejemplos* con *mejor*=$v\}$;
    subárbol ← *inducir-árbol-de-decisión*(*ejemplos-con-v*, *atributos*$-\{$*mejor*$\}$, *clase-mayoritaria*(*ejemplos*));
    añadir una rama con etiqueta v que conecte el nodo *árbol* con el *subárbol*;
  end for
  return árbol
end if

Árboles de decisión

- elegir mejor atributo: el que la división de los ejemplos según sus valores proporcione la máxima ganancia de información (máxima reducción de la incertidumbre)
- criterio para detener el crecimiento: se puede modificar para que el proceso de particionado según un atributo no se realize cuando sea poco útil:
  - a esta técnica se la denomina *poda* (*pruning*)
  - puede usar un *conjunto de validación* para decidir a posteriori que nodos podar (*post-pruning*)
  - puede decidirlo a priori utilizando la significación de una partición proporcionada por tests estadísticos (χ^2)
  - ventajas potenciales:
    - el árbol es más compacto
    - se gana tolerancia al ruido y se elimina sobreajuste

Árboles de decisión

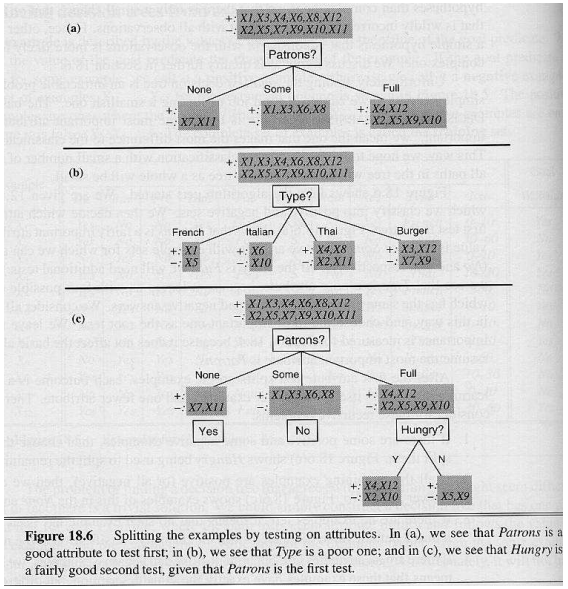


Figure 18.6 Splitting the examples by testing on attributes. In (a), we see that *Patrons* is a good attribute to test first; in (b), we see that *Type* is a poor one; and in (c), we see that *Hungry* is a fairly good second test, given that *Patrons* is the first test.

Árboles de decisión: Valores desconocidos

- cuando un ejemplo llega a un nodo con un atributo A cuyo valor es desconocido:
  - durante en el entrenamiento:
    1. se les asigna el valor más frecuente de los ejemplos con el misma valor de clase
    2. se les asigna el valor más frecuente de los ejemplos con que han llegado (o llegarán) a ese nodo
    3. se asigna una probabilidad p_i por cada valor posible v_i de A y se fracciona el ejemplo
  - durante la clasificación: como (2) o (3) de arriba

Metamétodos

- las decisiones de distintos modelos se pueden combinar para dar una única predicción
- la decisiones de cada modelo puede tener el mismo peso o no
- en el caso de clases cualitativas se puede tomar cada decisión como un voto
- en el caso de clases numéricas se puede calcular la media

Metamétodos: Bagging

- cada modelo tiene el mismo peso y por lo tanto su decisión es igual de importante
- cada modelo se obtiene con un subconjunto aleatorio del conjunto de entrenamiento
- la clase fi nal es la más frecuente de las predichas por cada modelo

Los algoritmos son:

- Aprendizaje:

  for all $i = 1 \ldots t$ **do**
  tomar n ejemplos con reemplazo del *training-set*
  construir un modelo M_i con un método de aprendizaje
  end for

- Clasifi cación:

  for all $i = 1 \ldots t$ **do**
  predecir con el modelo M_i
  end for
  return la clase predichas más frecuente

- *Bagging* intenta neutralizar la inestabilidad del método de aprendizaje.
- el modelo combinado de esta manera se comportan generalmente mejor que cada modelo.

Metamétodos: *Boosting* (*Bootstrap Aggregrating*)

- objetivo: mejorar la precisión de un modelo
- es iterativo: cada modelo construido tiene en cuenta los errores de los anteriores
- cada modelo tiene un peso y se usa para que las decisiones de los modelos más exitosos cuenten más en la decisión fi nal
- una de las variantes más conocidas de *boosting* es *AdaBoost*:

  asignar el mismo peso a todos los patrones del conjunto de entrenamiento
  for all $i = 1 \ldots t$ **do**
  construir un modelo M_i con el método de aprendizaje con el cj. de entrenamiento con pesos
  calcular el error e del modelo sobre el cj. de datos con pesos
  if $e = 0$ o $e = 0{,}5$ **then**
  acabar con la generación del modelo
  end if
  for all patrón del conjunto de datos **do**
  multiplicar el peso de cada patrón por $e/(1-e)$
  end for
  normalizar el peso de todos los patrones
  end for

Metamétodos: *Boosting*

- clasifi cación:

  asignar un peso de cero a todas las clases
  for all $i = 1 \ldots t$ **do**
  añadir $-\log(e/(1-e))$ al peso de la clase asignada por el modelo M_i
  end for
  devolver la clase con mayor peso

- cuando el modelo no admite pesos en los ejemplos se usa el muestreo con reemplazo en el que la probabilidad de ser elegido es proporcional al peso del ejemplo
- se aplica principalmente a *weak learners* como modelos subjecentes

Metamétodos: *Stacking*

- A diferencia de *bagging* o *boosting*, que combinan modelos obtenidos con el mismo método de aprendizaje, *stacking* combina modelos construidos con distintos métodos de aprendizaje.
- A los n modelos se le añade un *metalearner* que hace de árbitro y sustituye el procedimiento de votación simple.