Métodos de Remuestreo en Aprendizaje Automático

… en datos, en hipótesis,… y algunos otros trucos:
Cross-validation, Bootstrap, Bagging,
Boosting, Random Subspaces

Lo que sabemos hasta ahora:

- Hemos visto que hay 3 elecciones esenciales a la hora de ML:
  - Disponer de datos suficientemente buenos, y en suficiente cantidad: S
  - Disponer de un indicio que permita presuponer características sobre el modelo a aprender: $\mathcal{H}$
  - Disponer de un método de entrenamiento que permita obtener un buen representante del conjunto de hipótesis: $\mathcal{A}$
- Esta terna de elecciones se traduce en una descomposición del error del aprendizaje:

$$\text{Err} = \text{ruido} + \text{bias} + \text{varianza}$$

Qué pasaría si…

- … pudiéramos tener tantos datos como quisiéramos?
- … pudiéramos hacer muchas pruebas con nuestro algoritmo de entrenamiento?
- … pudiéramos usar varios modelos ?

La importancia del Remuestreo

- Lo más normal: Tenemos que aprender un modelo de un conjunto insuficiente de datos (razones: imposible/caro obtener más).
- … así que se pierde la opción de tener tantos datos como quisiéramos.
- Es un problema antiguo que ya fue "parcheado" por la estadística:

Remuestreo

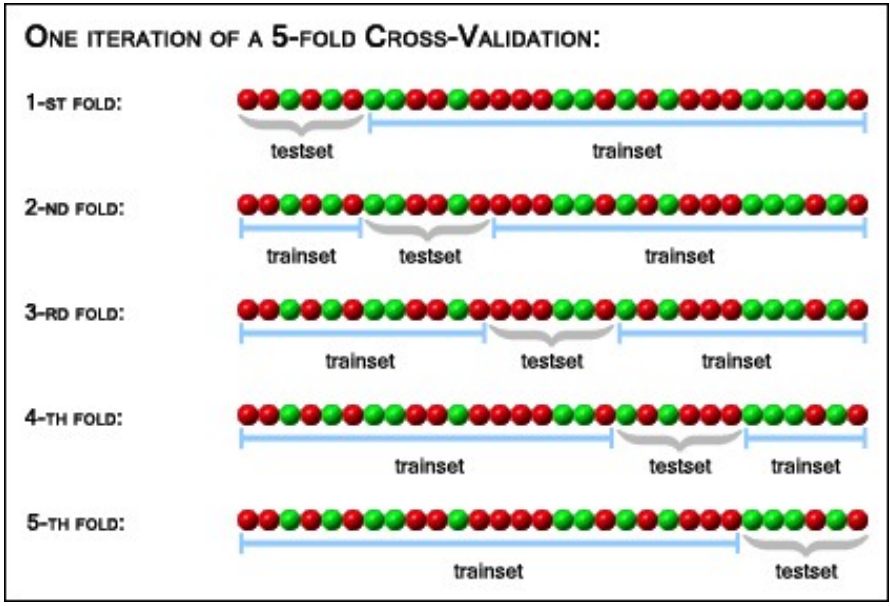
- … esencialmente, consiste en tomar muestras de forma repetida del conjunto completo de muestras que tenemos, *S*.
- Tradicionalmente muy costoso… hoy en día ya no tanto.
- Dos métodos principales: **Cross-validation**, **Bootstrap**.

Cross-validation

- Proviene de una generalización de la división:

$$S = S_{test} \cup S_{train}$$

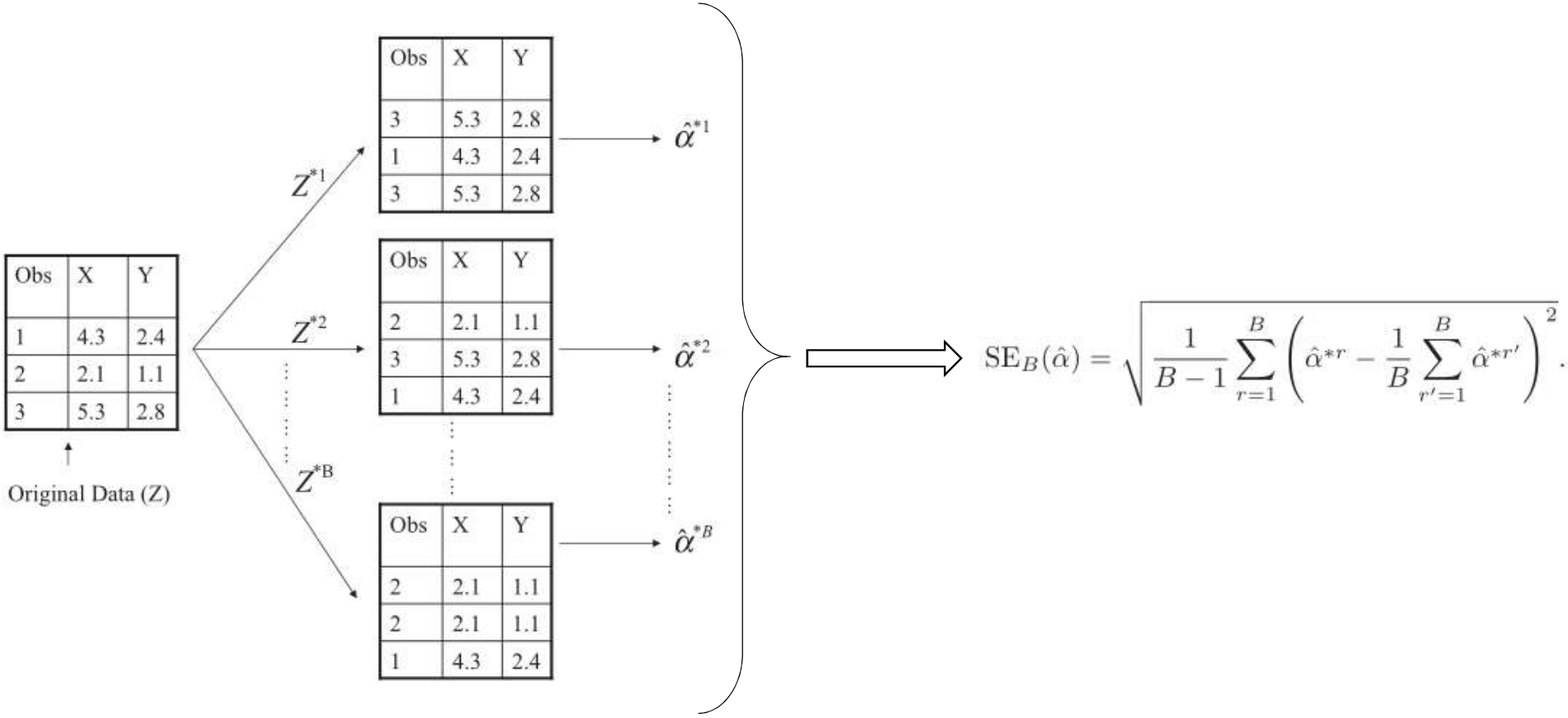
- De esta forma podemos calcular el error empírico del modelo con independencia del entrenamiento.
- El más común es el **k-fold cross-validation**:
  - Se divide S en k conjuntos de, aproximadamente, el mismo tamaño: $S_1,\ldots,S_k$.
  - Cada S_i se usa como validación mientras el resto se usa como entrenamiento. Obteniendo $\mathcal{E}rr_i$.
  - $\mathcal{E}rr$ se calcula como la media de los $\mathcal{E}rr_i$.
  - Valores normales de k : 5…10
- No genera un modelo, aproxima mejor el error.



Bootstrap

- Tampoco devuelve modelos más depurados.
- En su formato habitual se usa para estimar coeficientes estadísticos de distribuciones.
- Normalmente, no podemos acceder a la distribución real de un proceso (es lo que queremos aprender), pero disponemos de algunas muestras (conjunto de entrenamiento):
- De la muestra original, de tamaño N, se toman N' muestras al azar con reemplazamiento.
- Con cada muestra se calcula el coeficiente estadístico buscado, y se espera que la media de todos ellos sea similar al coeficiente de la distribución real que generó el conjunto de entrenamiento original.

Bootstrap

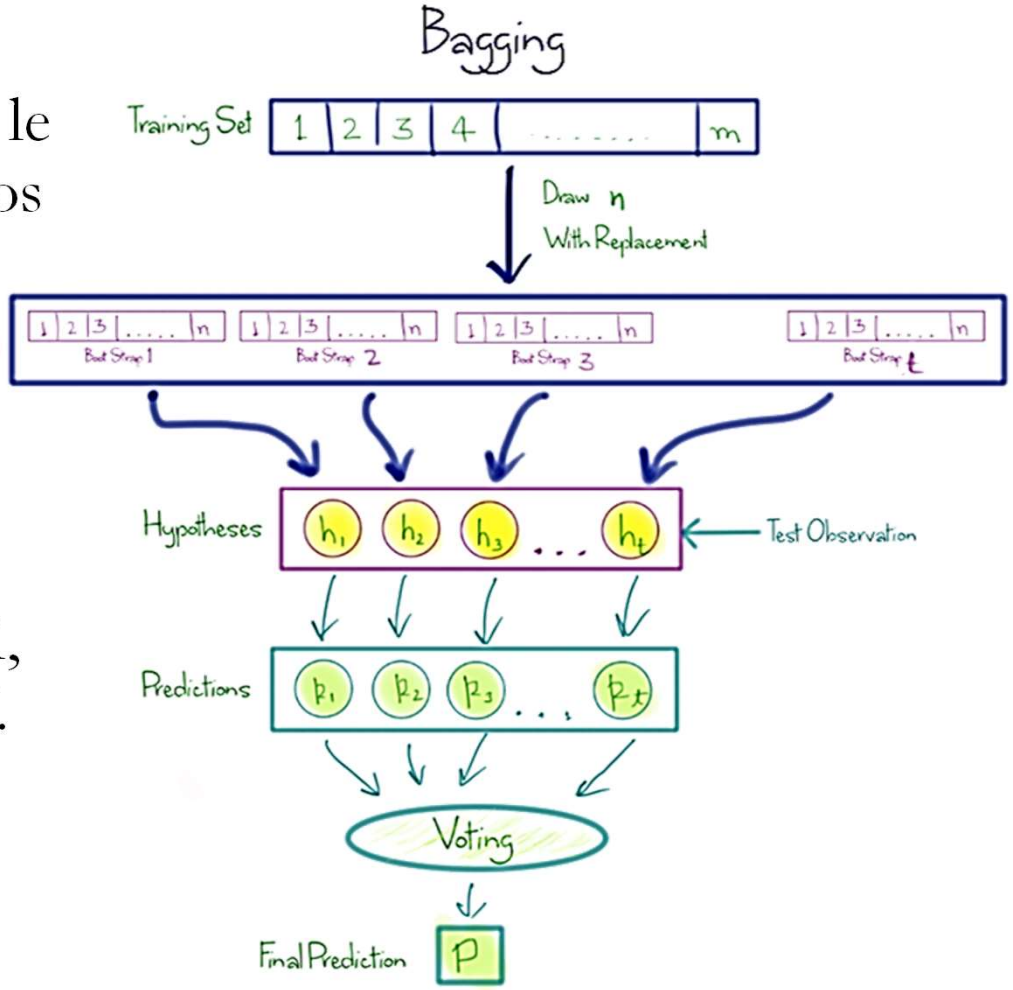

Obs X Y
1 4.3 2.4
2 2.1 1.1
3 5.3 2.8
Original Data (Z)
Z^{*1}
Z^{*2}
Z^{*B}
Obs X Y
3 5.3 2.8
1 4.3 2.4
3 5.3 2.8
Obs X Y
2 2.1 1.1
3 5.3 2.8
1 4.3 2.4
Obs X Y
2 2.1 1.1
2 2.1 1.1
1 4.3 2.4
$\hat{\alpha}^{*1}$
$\hat{\alpha}^{*2}$
$\hat{\alpha}^{*B}$
$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'}\right)^2}.$

¿Esto es todo?

- Hasta ahora solo hemos remuestreado datos, con cada uno de ellos hemos aprendido un modelo, y a partir de ahí hemos visto si podíamos tener una idea más real del Error cometido.
- Todo esto es simplemente repetir lo que ya se venía haciendo en Estadística desde hacía un tiempo (de hecho, es la base de la Estadística Clásica).
- Pero estamos interesados en la parte algorítmica del aprendizaje…
- ¿Hay otras opciones?, ¿cómo puede usarse una idea similar para obtener un algoritmo que, esperamos, devuelva mejores modelos?

Bagging: **B**ootstrap **Aggregating**

- No será hasta el año 96 que a alguien se le ocurre hacer algo similar con los modelos de aprendizaje: Breiman (el que creó los Random Forest), y lo llamó *Bagging Predictors*.
- Los árboles de decisión sufren de una **alta varianza**: si dividimos S en dos partes y con nada una hacemos un árbol, los resultados son muy distintos entre sí.



Bagging: **B**ootstrap **Agg**regat**ing**

- Resultado fundamental:

  Si $Z_1,\ldots,Z_n$ *son observaciones independientes de un mismo proceso,* Z*, que tiene varianza* σ^2*, entonces la varianza de la media de ellas:* $\frac{1}{n}\sum_{i=1}^{n} Z_i$*, es* (σ^2/n).

- Es decir:

  La media de un conjunto de observaciones reduce la varianza.

- Por tanto, una forma natural de reducir la varianza y mejorar la predicción es tomar muchos conjuntos de entrenamiento, construir con ellos modelos de predicción **independientes**, y calcular la media de las predicciones.

Bagging: **B**ootstrap **Agg**regat**ing**

- Como no suele ser habitual tener muchos conjuntos de entrenamiento independientes, se usa Bootstrap para conseguirlos a partir de uno solo original.
- En el caso de los árboles de decisión, este procedimiento hace que los árboles crezcan hasta profundidad máxima y no se podan. De esta forma, cada árbol individual tiene una alta varianza, pero bajo sesgo. La media de ellos disminuye la varianza.
- Para hacer clasificación, se toma la regla de la mayoría (la clasificación con mayor número de votos).

Bagging: **B**ootstrap **Agg**regat**ing**

- Se puede probar que en caso de estar trabajando con modelos individuales con mucha varianza (por ejemplo, árboles de decisión, redes neuronales) entonces el método funciona muy bien.
- Si los modelos individuales no presentan mucha varianza (k-NN, regresores lineales), en el caso de buscar un regresor apenas se gana nada con el bagging, y en el caso de clasificador, incluso podemos perder.
- Posibilidad de medir el error con Out-of-Bag (OOB) que, para cada muestra, considera únicamente los árboles que no lo han tomado en cuenta (así hay menos sobreajuste).
- En el caso de los árboles, empeora la interpretación, obviamente.

Random Subspaces

- La disminución de variación por Bagging depende de la independencia entre hipótesis.
- Si hay una característica muy potente, el Bagging presenta problemas cuando se aplica a árboles de decisión.
- En 2001, Breiman presenta una variación específica para árboles de decisión (aunque se puede usar para cualquier otro método de aprendizaje), y que llamó **Random Forest**.
- Al método general, cuando se independiza de la familia de hipótesis específica, se le denomina **Random Subspaces**.
- Funciona exactamente igual que el Bagging, pero en vez de muestrear S, se toma un subconjunto aleatorio de características de los datos de entrada.

Random Subspaces

Más concretamente, si $X = X_1 \times \cdots \times X_D$, para cada sucesión $p = (i_1, \ldots, i_k) \in \mathbb{N}^k$ con $1 \leq i_1 < \cdots < i_k \leq D$ podemos considerar la proyección:

$$\pi_p : X \to X_{i_1} \times \cdots \times X_{i_k}$$
$$(x_1, \ldots, x_D) \mapsto (x_{i_1}, \ldots, x_{i_k})$$

Notaremos $S_p = \{(\pi_p(x), y) : \ (x, y) \in S\}$.

Si L es el número de aproximadores individuales que se desea obtener, la combinación se construye siguiendo el siguiente algoritmo:

1. Para cada $1 \leq i \leq L$:
   - Se selecciona al azar $1 \leq k \leq D$, y $p_i = (i_1, \ldots, i_k) \in \mathbb{N}^k$ con $1 \leq i_1 < \cdots < i_k \leq D$.
   - Se entrena h_i en S_{p_i} (siguiendo el algoritmo $\mathcal{A}_i$ elegido).

Random Subspaces

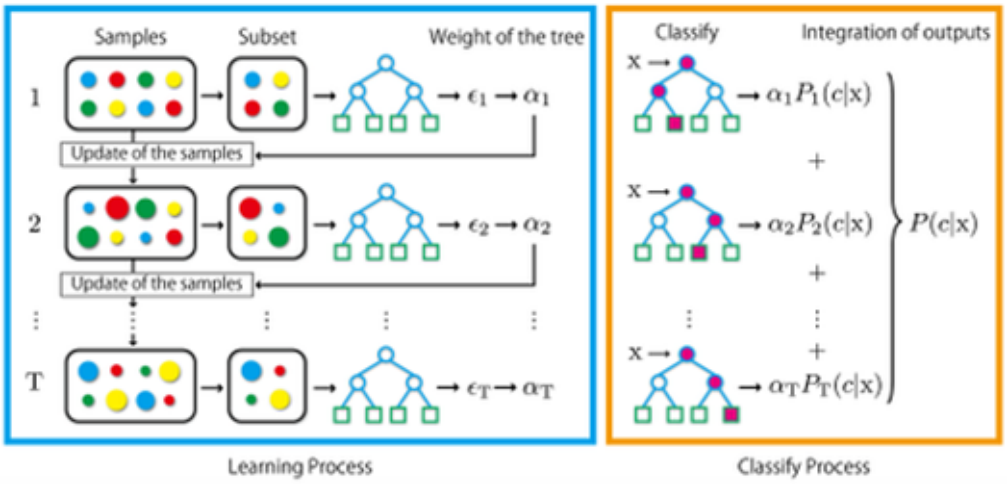
2. La combinación construida clasifica por medio del voto de la mayoría, es decir, para cada $x \in S$:

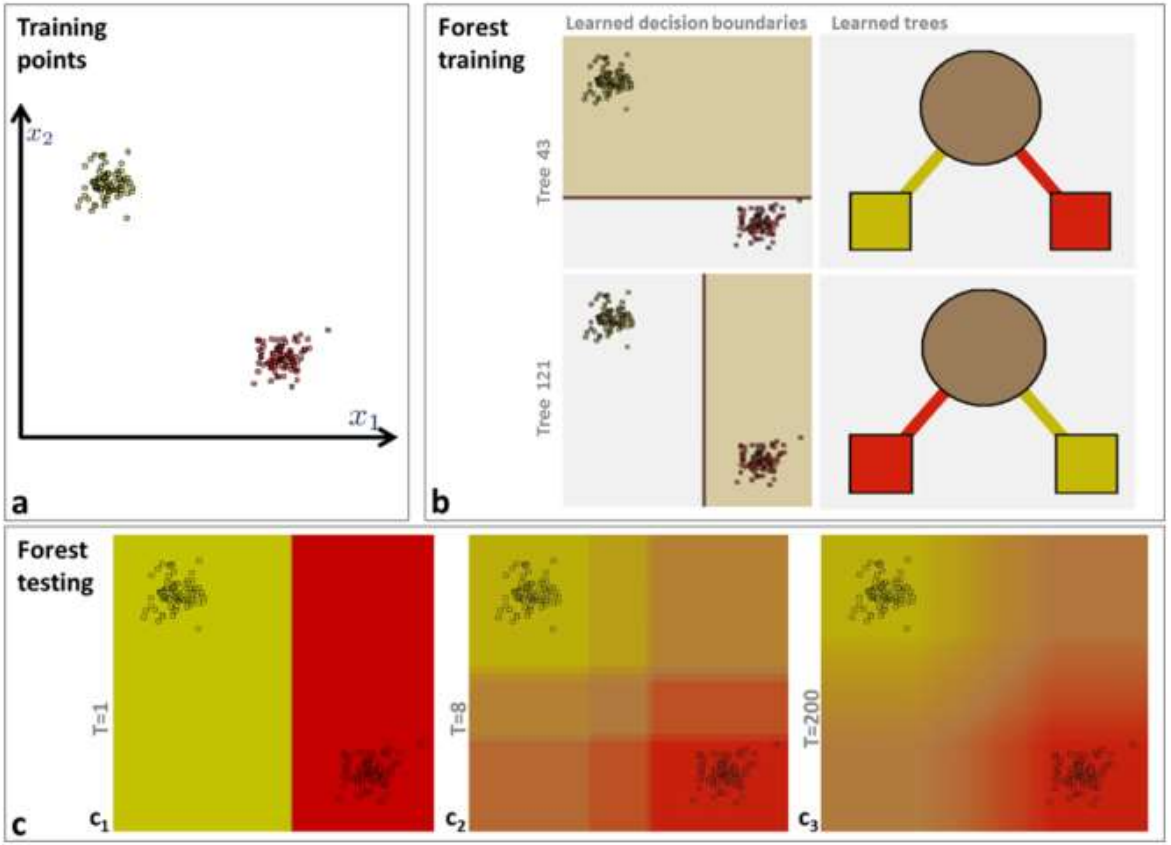
$$h(x) = signo(\sum_{i=1}^{L} h_i(x))$$

o devuelve la media de los valores dados por los aproximadores (también se puede dar una media ponderada):

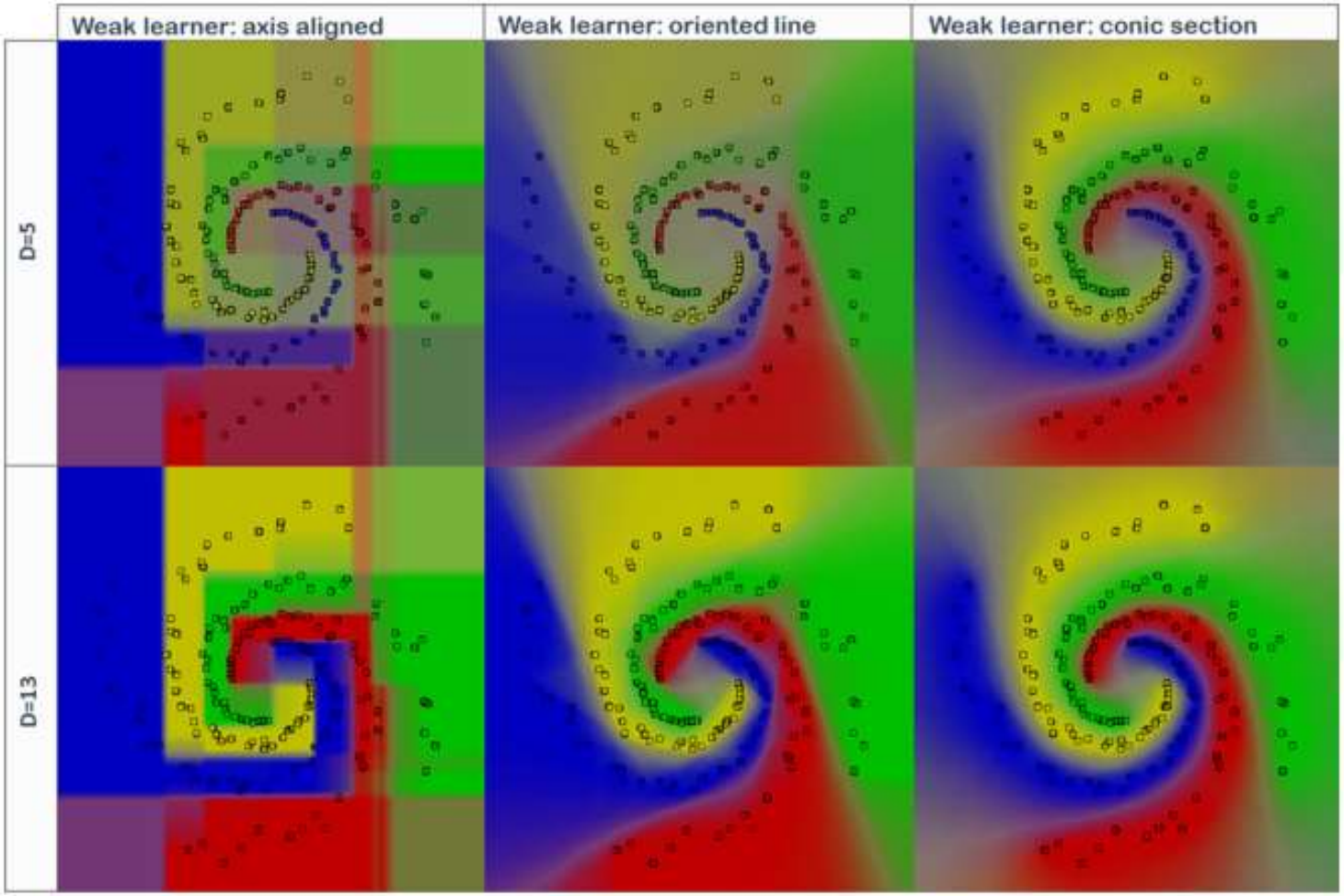
$$h(x) = \frac{1}{L} \sum_{i=1}^{L} h_i(x)$$

Random Subspaces





Random Subspaces



Boosting

Boosting supone una aproximación ligeramente diferente que intenta responder desde un punto de vista teórico dos problemas fundamentales:

1. El problema del sesgo-varianza (cómo conseguir reducir los dos a la vez): cuanto más expresiva es $\mathcal{H}$, menor es el sesgo, pero mayor suele ser la varianza: complejidad creciente en $\mathcal{H}$.
2. La complejidad computacional de $\mathcal{A}$ para conseguir un aprendizaje efectivo (se puede probar que, en general, es un problema **NP**-duro): aprendizaje por *weak learners* (intuitivamente, hipótesis que son ligeramente mejores que el azar, pero que son implementables muy eficientemente).

Boosting

- La pregunta era: ¿puede potenciarse (boost) un *weak learner* que sea computacionalmente eficiente en un *strong learner* de manera computacionalmente eficiente? (Kearns & Valiant, 1988)
- Resuelto en 1990 por Robert Schapire (estudiante graduado del MIT), aunque una solución no muy buena, y mejorado en 1995 con ayuda de Yoav Freund en el método AdaBoost.
- Supuso un salto cualitativo en el ML, haciendo que los resultados mejorasen tanto que solo con el DL se han obtenido después resultados comparables.
- Vamos a explicar brevemente porqué es tan importante…

PAC-learner: Strong Learner

Supongamos que S sigue una distribución de probabilidad (que es la que queremos aprender en forma de función).

Decimos que $\mathcal{A}$ genera un *strong learner* (verifica **PAC**) si:

para cada $\varepsilon, \delta > 0$, $\mathcal{A}$ produce h de $\mathcal{H}$ cuyo error es menor que ε, con probabilidad mayor que $1-\delta$.

Estas probabilidades están tomadas en la aleatoriedad de $\mathcal{A}$ y de S, pero para que $\mathcal{A}$ funcione bien debe ser para cualquier $\mathcal{P}$.

PAC-learner: Weak Learner

Supongamos que S sigue una distribución de probabilidad (que es la que queremos aprender en forma de función).

Decimos que $\mathcal{A}$ genera un ε-*weak learner* si :

para cada ε, δ> 0, $\mathcal{A}$ produce h de $\mathcal{H}$ cuyo error es menor que $\frac{1}{2} - \varepsilon$, con probabilidad mayor que 1-δ.

Estas probabilidades están tomadas en la aleatoriedad de $\mathcal{A}$ y en S, pero para que $\mathcal{A}$ funcione bien debe ser para cualquier S.

El problema computacional del aprendizaje

- Desde el punto de vista estadístico, no hay diferencia entre ambos aprendizajes (los dos vienen caracterizados por una dimensión VC finita). Es decir, $\mathcal{A}$ es PAC-learner si y solo si genera un weak-learner.
- Pero desde el punto de vista computacional puede haber diferencias… quizás encontrar un weak-learner no sea **NP**-duro.
- Si es así, responder al problema del boosting es esencial para poder convertir una búsqueda NP-dura en otra que no lo es…
- Se pueden dar ejemplos sencillos de weak-learners que se pueden potenciar eficientemente a strong-learners:

$$h_{\theta_1,\theta_2,b}(x) = \begin{cases} +b & \text{if } x < \theta_1 \text{ or } x > \theta_2 \\ -b & \text{if } \theta_1 \leq x \leq \theta_2 \end{cases}$$

An example hypothesis (for $b = 1$) is illustrated as follows:

AdaBoost

A diferencia del **Bagging**, el **boosting** no crea versiones de S, sino que se trabaja siempre con el conjunto completo de entrada, y se manipulan los pesos de los datos para generar modelos distintos.
En cada iteración se incrementa el peso de los objetos mal clasificados por el predictor de esa iteración, por lo que en la construcción del próximo predictor estos objetos serán más importantes y será más probable clasificarlos bien.

AdaBoost

input:
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
weak learner WL
number of rounds T
initialize $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.
for $t = 1, \ldots, T$:
invoke weak learner $h_t = \mathrm{WL}(\mathbf{D}^{(t)}, S)$
compute $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$
update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
output the hypothesis $h_s(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right)$.

AdaBoost

Se supone que WL genera un weak-learner a partir de S y una distribución de probabilidad en S. Devuelve la media ponderada (o la clasificación ponderada) de los diversos weak-learners aprendidos… que se puede probar que es un strong-learner.

AdaBoost

input:

training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$

weak learner WL

number of rounds T

initialize $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.

for $t = 1, \ldots, T$:

invoke weak learner $h_t = \mathrm{WL}(\mathbf{D}^{(t)}, S)$

compute $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$

let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$

update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$

output the hypothesis $h_s(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right)$.

AdaBoost

THEOREM 10.2 *Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$. Then, the training error of the output hypothesis of AdaBoost is at most*

$$L_S(h_s) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{[h_s(\mathbf{x}_i) \neq y_i]} \leq \exp(-2\gamma^2 T) .$$

- Cada iteración de AdaBoost conlleva O(m) operaciones y una llamada a WL.
- Si WL es eficiente, AdaBoost también.

Aprendizaje Ensemble vs Aprendizaje Múltiple