

Ensemble Methods

Aprendizaje Automático INF393 II-2018



Ricardo Ñanculef
UTFSM Campus San Joaquín.

Agenda

- Motivación
- Bagging & Random Forests
- Boosting & Gradient Boosting
- Mezcla de Expertos

Motivación

Ensembles

- Métodos que combinan varios modelos (típicamente sencillos e inestables) para formar un predictor más robusto y con mayor poder de generalización.
- Es una amplia familia de métodos que se distinguen en cómo obtienen/eligen cada modelo individual y cómo éstos se combinan.
 - En clasificación típicamente mayoría de votos.
 - En regresión típicamente un promedio o suma con pesos.

Majority Voting

- Supongamos que contamos con un clasificador que garantiza la clase correcta con probabilidad p .
- Si usamos K (impar) “instancias independientes” de este clasificador (es decir, los K clasificadores se equivocan independientemente), la probabilidad de que la mayoría se equivoque viene dada por:

$$P = 1 - \sum_{j=0}^{(k-1)/2} \binom{k}{j} (1-p)^j p^{n-j}$$

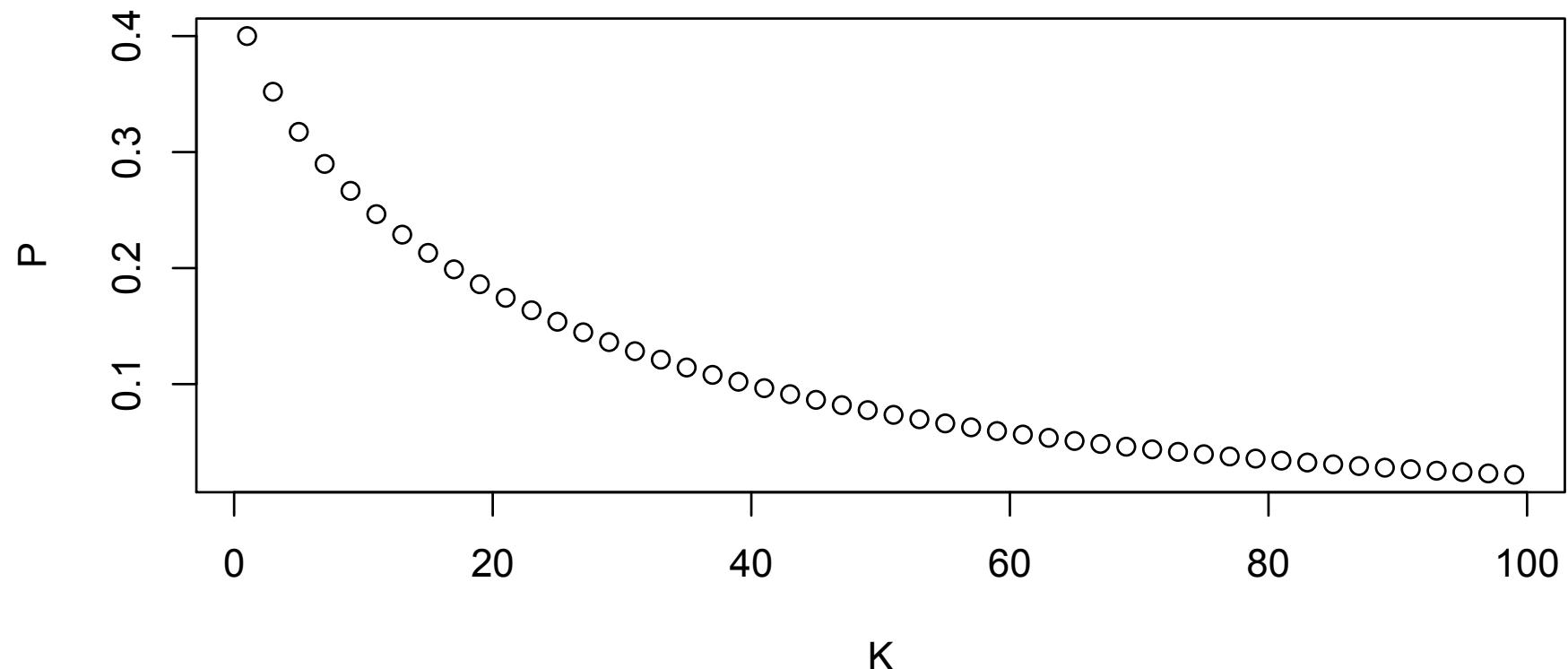
Majority Voting

- Si $p=0.5$, $P=0.5$, independiente de K .
- Si $p=0.6$, tenemos que

Ens. Size (K)	Prob. Error (P)
11	0,247
21	0,174
51	0,073
101	0,021
501	0,000003

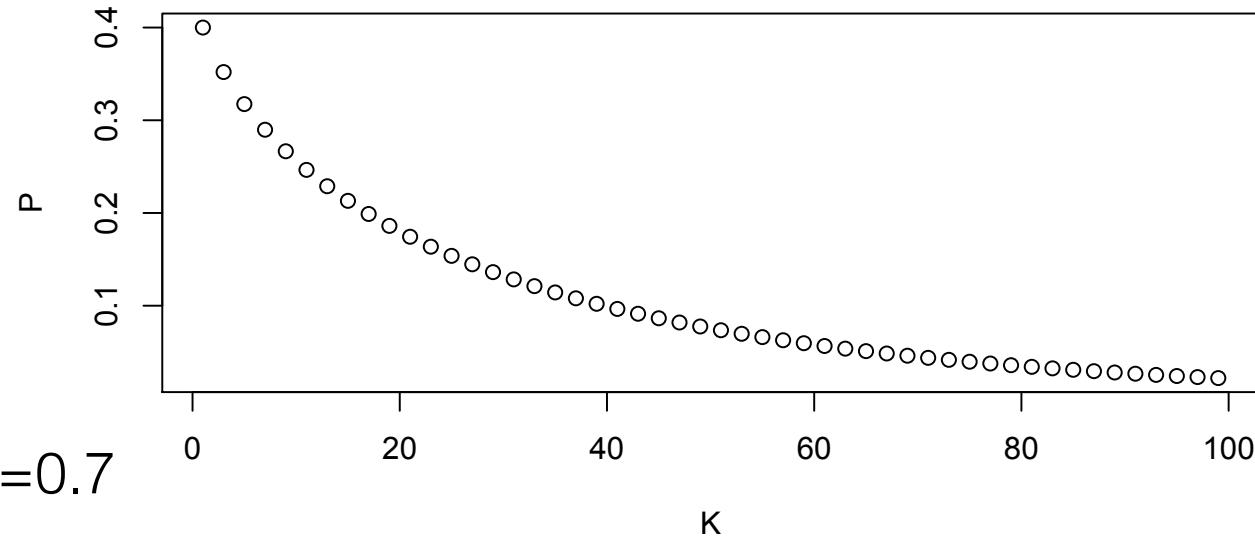
Majority Voting

- Si $p=0.5$, $P=0.5$, independiente de K .
- Si $p=0.6$, tenemos que

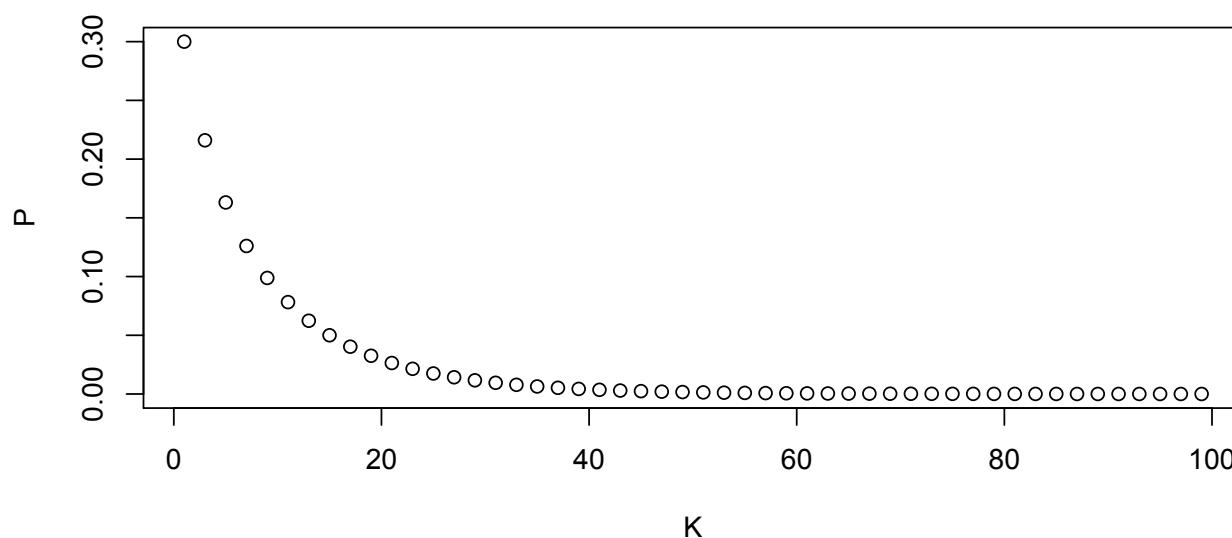


Majority Voting

- $p=0.6$



- $p=0.7$



Ensembles Continuos

- Sea $P(x,y)$ la distribución de probabilidad que genera las observaciones (x,y) que intentamos modelar y asumamos que la respuesta \mathbf{y} se genera a partir de \mathbf{x} de la siguiente forma

$$y = f(x) + \epsilon, \quad \mathbb{E}(\epsilon) = 0, \quad \mathbb{V}(\epsilon) = \sigma^2$$

- Dado un conjunto de datos de entrenamiento S , nuestro learner implementa una función $f_S(x)$ que no necesariamente coincide con $f(x)$.

Ensembles Continuos

- Queremos descomponer el error de predicción sobre un nuevo par (x,y) (digamos de test)

$$\mathbb{E} [(y - f_S(x))^2]$$

- Si asumimos que (x,y) se genera IID respecto del conjunto de ejemplos, tenemos que:

$$\begin{aligned}\mathbb{E} [(y - f_S(x))^2] &= \mathbb{E} [y^2 - 2f_S(x)y + f_S(x)^2] \\ &= \mathbb{E} [f_S(x)^2] - 2\mathbb{E} [f_S(x)y] + \mathbb{E} [y^2] \\ &= \mathbb{E} [f_S(x)^2] - 2\mathbb{E} [f_S(x)] \mathbb{E} [y] + \mathbb{E} [y^2]\end{aligned}$$

Descomposición Sesgo-Varianza

- Re-acomodando

$$\begin{aligned}\mathbb{E} [(y - f_S(x))^2] &= \mathbb{E} [f_S(x)^2] - 2\mathbb{E} [f_S(x)]\mathbb{E} [y] + \mathbb{E} [y^2] \\&= \mathbb{V} [f_S(x)] + \mathbb{E} [f_S(x)]^2 - 2\mathbb{E} [f_S(x)]f(x) + \mathbb{V} [y] + \mathbb{E} [y]^2 \\&= \mathbb{V} [f_S(x)] + \mathbb{E} [f_S(x)]^2 - 2\mathbb{E} [f_S(x)]f(x) + \mathbb{V} [y] + f(x)^2 \\&= \mathbb{V} [f_S(x)] + (\mathbb{E} [f_S(x)] - f(x))^2 + \mathbb{V} [y] \\&= \mathbb{V} [f_S(x)] + (\mathbb{E} [f_S(x)] - f(x))^2 + \sigma^2\end{aligned}$$

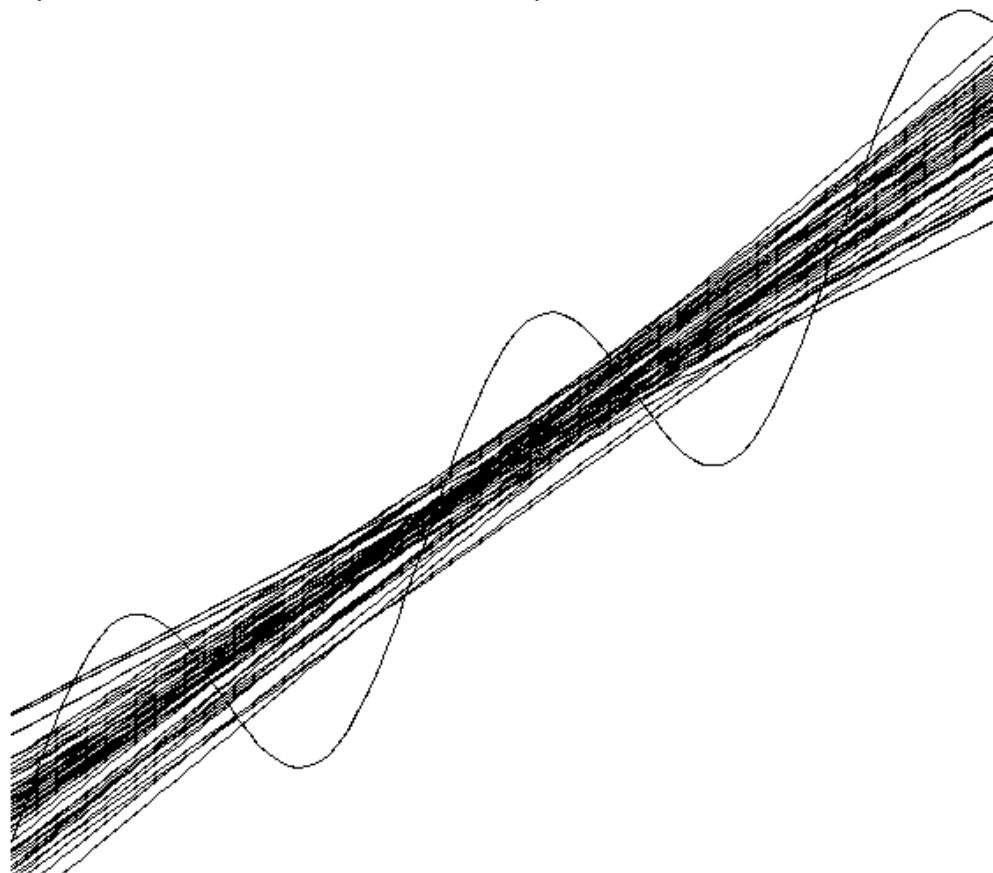
- Podemos reducir el error de predicción reduciendo el sesgo o la varianza del predictor.

Descomposición Sesgo-Varianza

- **Bias:** Diferencia media entre el valor verdadero y el valor que predice el modelo considerando diferentes realizaciones del conjunto de entrenamiento.
- **Varianza:** cuánto cambian las predicciones del modelo (red neuronal) respecto de la media cuando cambia el conjunto de ejemplos.

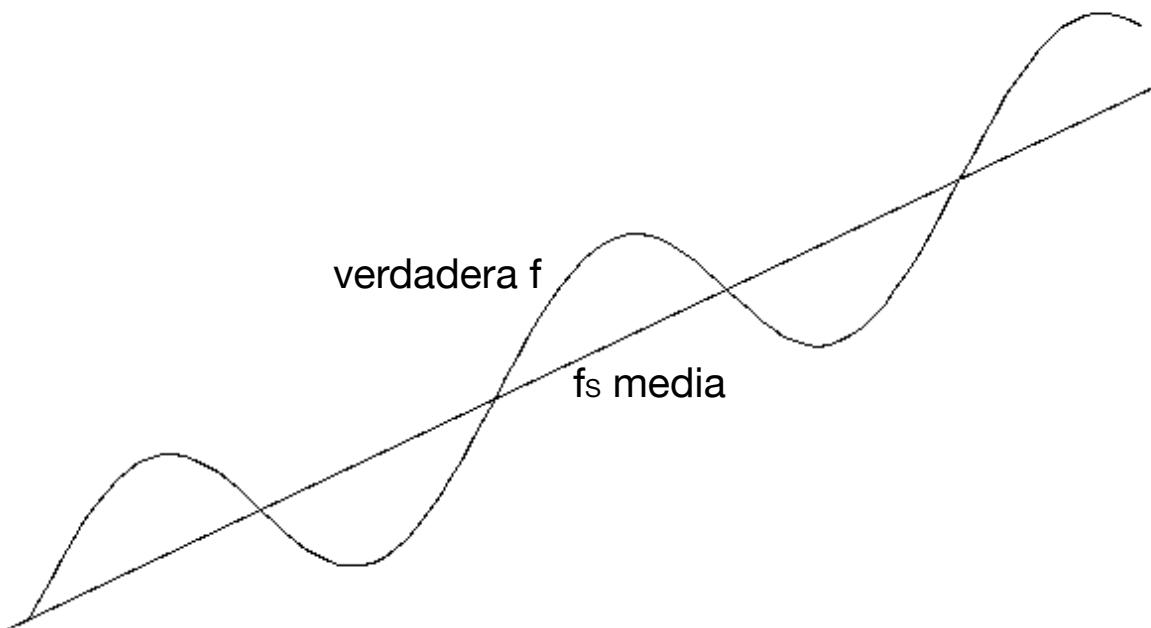
Descomposición Sesgo-Varianza

- Regresión lineal sobre 50 conjuntos de entrenamiento diferentes (misma distribución).



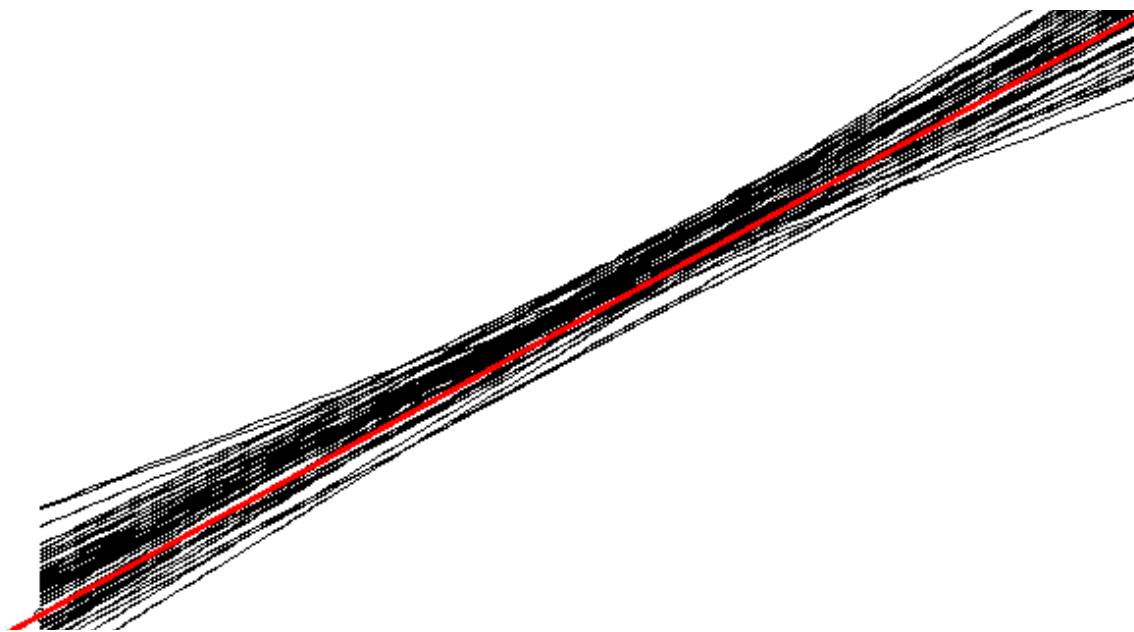
Descomposición Sesgo-Varianza

- Sesgo



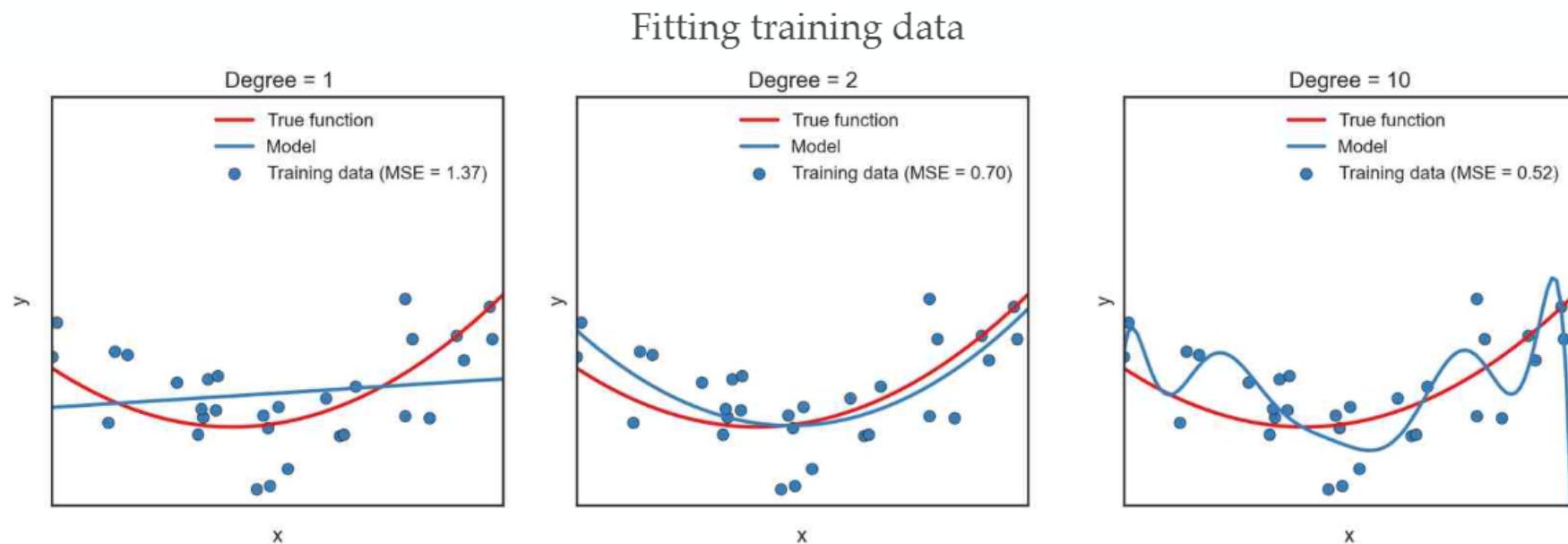
Descomposición Sesgo-Varianza

- Varianza



Descomposición Sesgo-Varianza

- Con frecuencia, un modelo con poco sesgo (bias) exhibe alta varianza y un modelo con con poca varianza tiene sesgo alto.



Descomposición Sesgo-Varianza

$$\begin{aligned}\mathbb{E} [(y - f_S(x))^2] &= \mathbb{E} [f_S(x)^2] - 2\mathbb{E} [f_S(x)] \mathbb{E} [y] + \mathbb{E} [y^2] \\&= \mathbb{V} [f_S(x)] + \mathbb{E} [f_S(x)]^2 - 2\mathbb{E} [f_S(x)] f(x) + \mathbb{V} [y] + \mathbb{E} [y]^2 \\&= \mathbb{V} [f_S(x)] + \mathbb{E} [f_S(x)]^2 - 2\mathbb{E} [f_S(x)] f(x) + \mathbb{V} [y] + f(x)^2 \\&= \mathbb{V} [f_S(x)] + (\mathbb{E} [f_S(x)] - f(x))^2 + \mathbb{V} [y] \\&= \mathbb{V} [f_S(x)] + (\mathbb{E} [f_S(x)] - f(x))^2 + \sigma^2\end{aligned}$$

- Podemos reducir el error de predicción reduciendo el sesgo o la varianza del predictor.
- Con frecuencia, un modelo con poco sesgo (bias) exhibe alta varianza y un modelo con poca varianza tiene sesgo alto.

Ensembles Continuos

- Supongamos que construimos nuestro modelo como promedio de K modelos estadísticamente independientes:

$$f(x) = \frac{1}{K} \sum_i f^{(i)}(x)$$

- Si cada modelo del “ensemble” tiene sesgo B (o su sesgo puede acotarse por B),

$$\text{Bias}(f) = \frac{1}{K} \sum_i \text{Bias}\left(f^{(i)}(x)\right) = B$$

- El ensemble mantiene el sesgo.

Ensembles Continuos

- Si cada modelo del “ensemble” tiene varianza C (o su varianza puede acotarse por C),

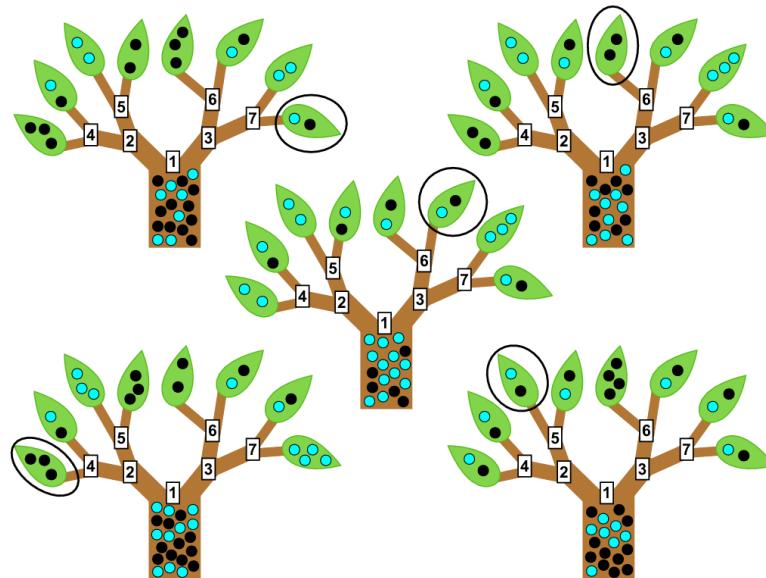
$$\mathbb{V}(f) = \frac{1}{K^2} \sum_i \mathbb{V}\left(f^{(i)}(x)\right) = \frac{C}{K}$$

- El ensemble reduce la varianza!!
- Mientras más modelos combinamos mejor!

Ensembles Continuos

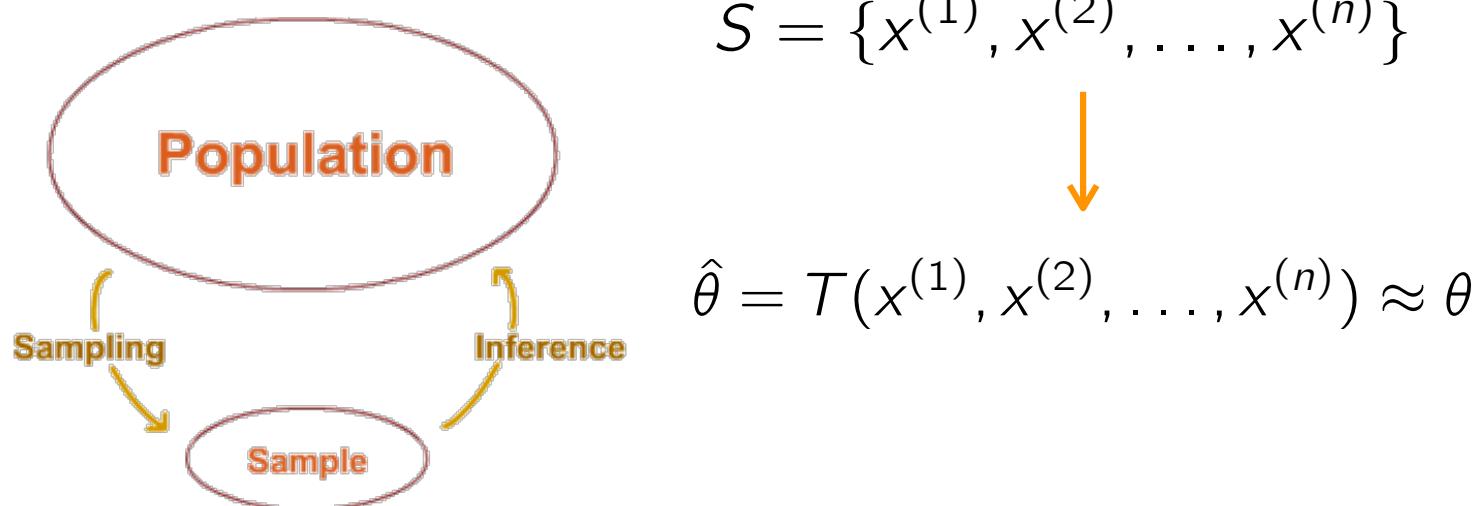
- Ok, lo anterior ocurre si los modelos que se combinan son estadísticamente independientes. En la práctica esto es difícil de obtener porque los modelos se ajustan con el mismo conjunto de ejemplos.
- Muchos métodos para “ensamblar” hipótesis buscan generar modelos “aproximadamente independientes”.

Bagging & Random Forests



Bootstrap

- Método computacional para estimar la distribución de probabilidad de un estimador.
- Un estimador T es en términos generales, una función de una muestra que se extrae de una población y que intenta estimar alguna característica de ésta.



Bootstrap

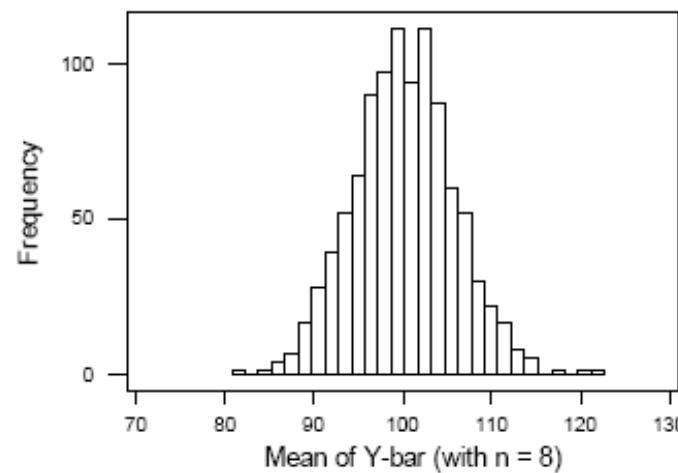
- Por ejemplo, el estimador podría ser la media muestral (estima la media poblacional) y nuestro objetivo podría ser conocer su valor esperado y su varianza.

$$T = \sum_i x^{(i)}/n$$

- Si suponemos que la población es normal con media μ y varianza σ^2 , y asumimos que la muestra es IID, sabemos que

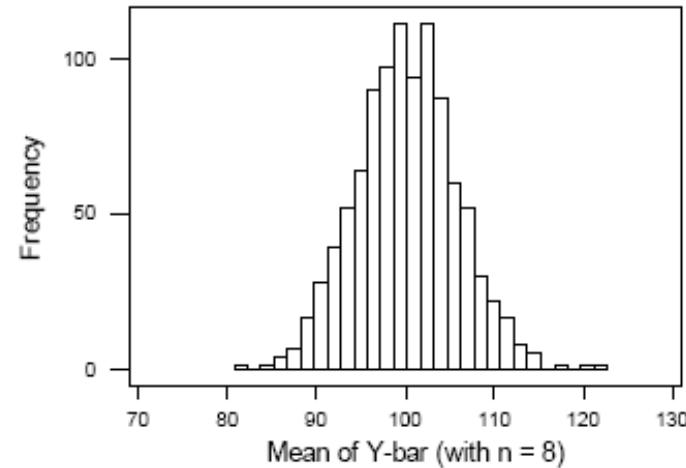
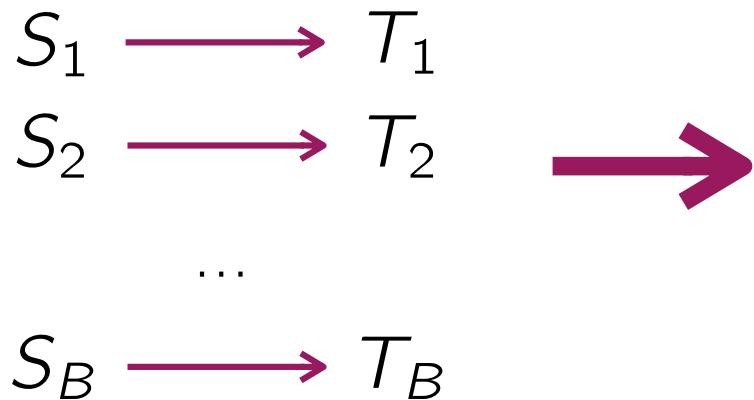
$$\mathbb{E}[T] = \mu$$

$$\text{Var}[T] = \sigma^2/n$$



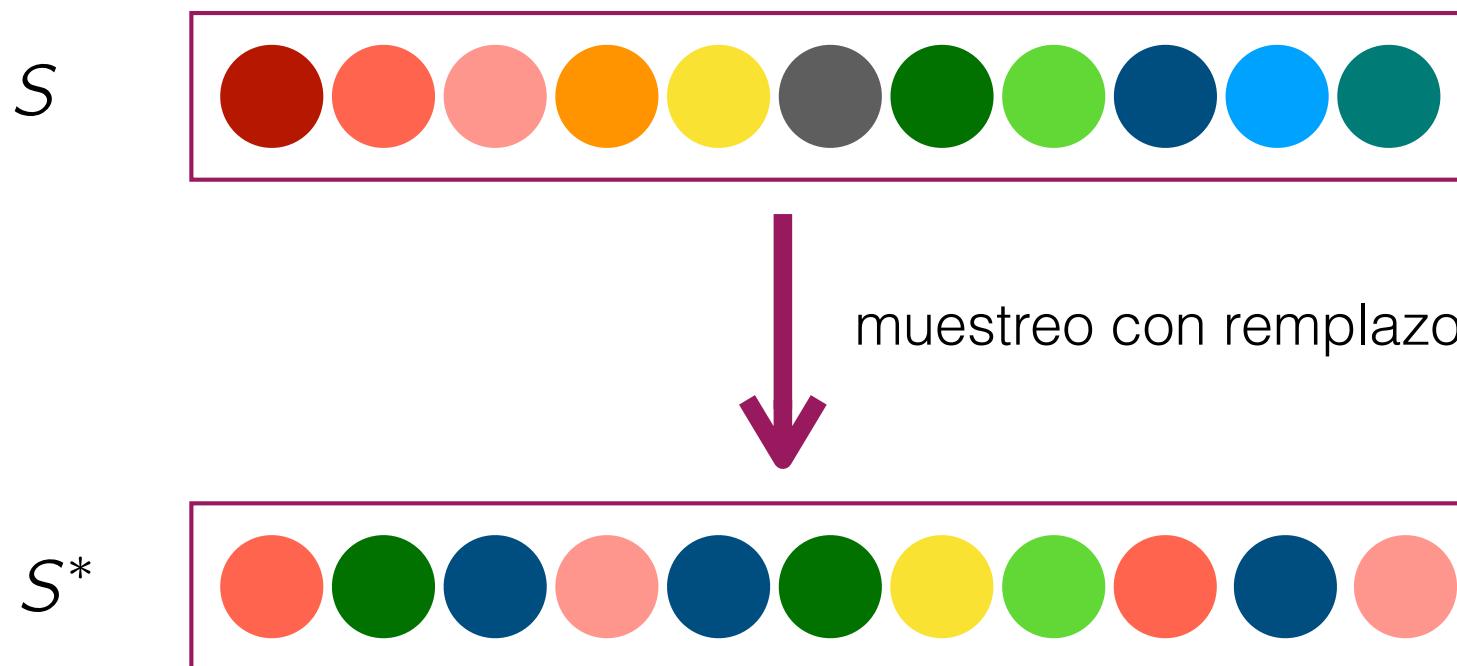
Bootstrap

- Deducir esta distribución sin asumir normalidad o para estimadores mucho más complicados que la media es difícil.
- Idea: Si tuviésemos acceso a la población (distribución real de los datos), podríamos obtener varias muestras IID y estudiar como cambia el valor del estimador.



Bootstrap

- Idea: En vez de muestrear de la población (cosa que no podemos hacer en general), muestrear desde la muestra.

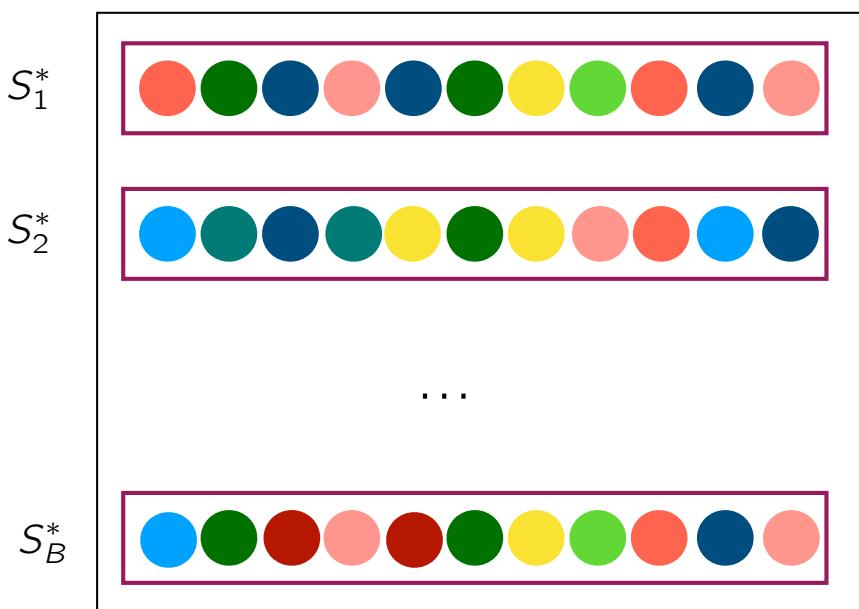


$$|S^*| = |S| = n$$

Bootstrap

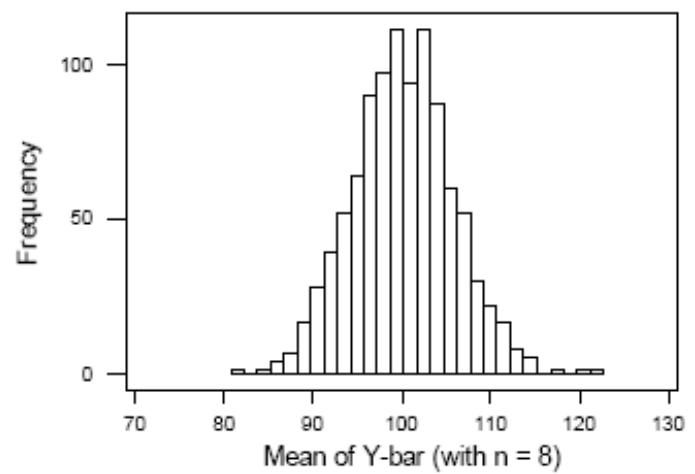


muestra original



re-muestreos
(muestras bootstrap)

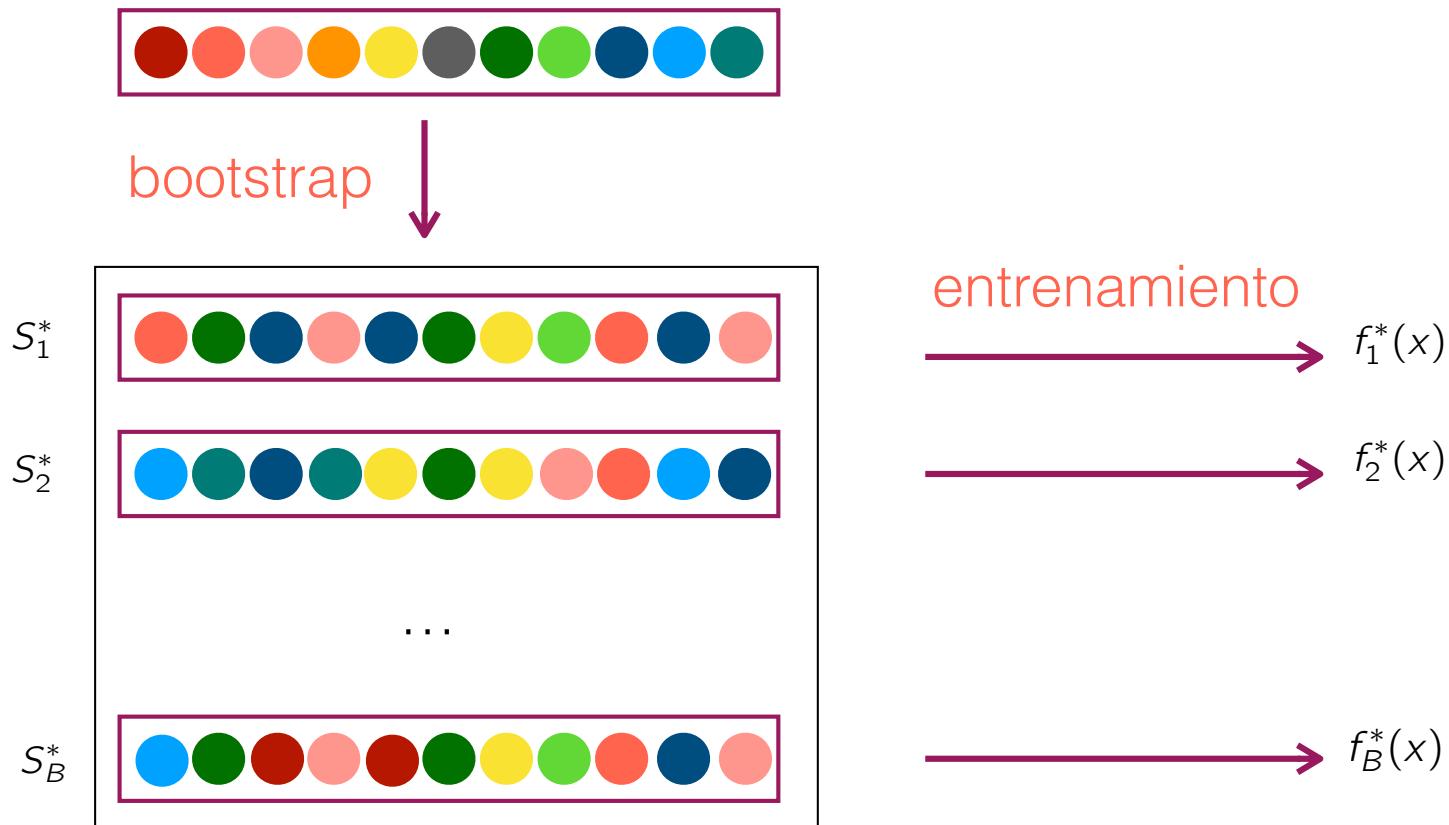
estimación



Bootstrap

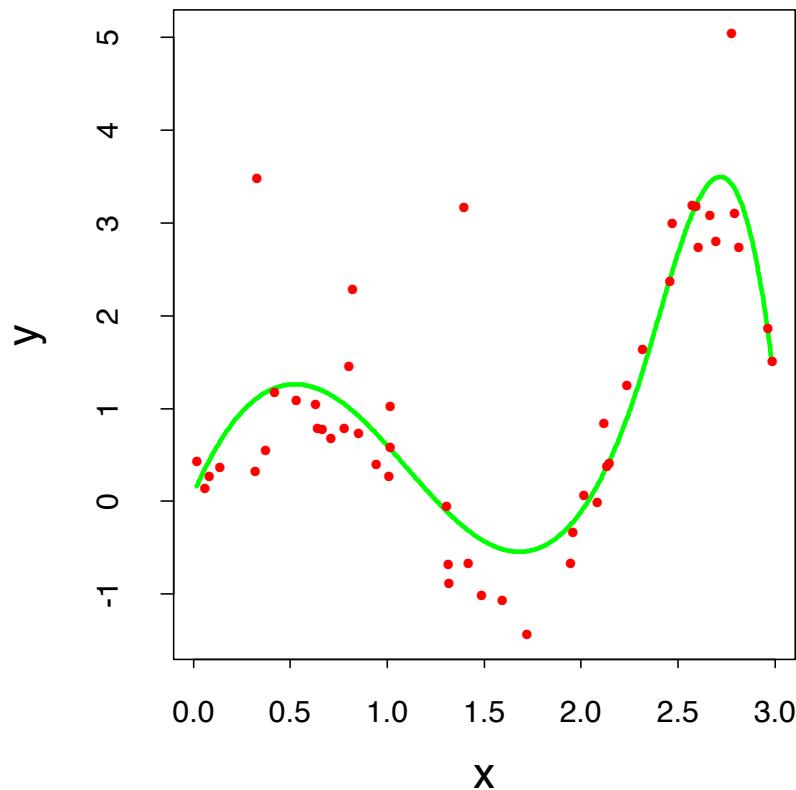
- Un modelo entrenado es, estadísticamente hablando un estimador.
Podemos usar el bootstrap para estimar su sesgo o su varianza.

ejemplos



Ejemplo: Bootstrap para obtener un PI

- Consideraremos un modelo lineal (entrenado con características polinomiales) sobre un conjunto de ejemplos.

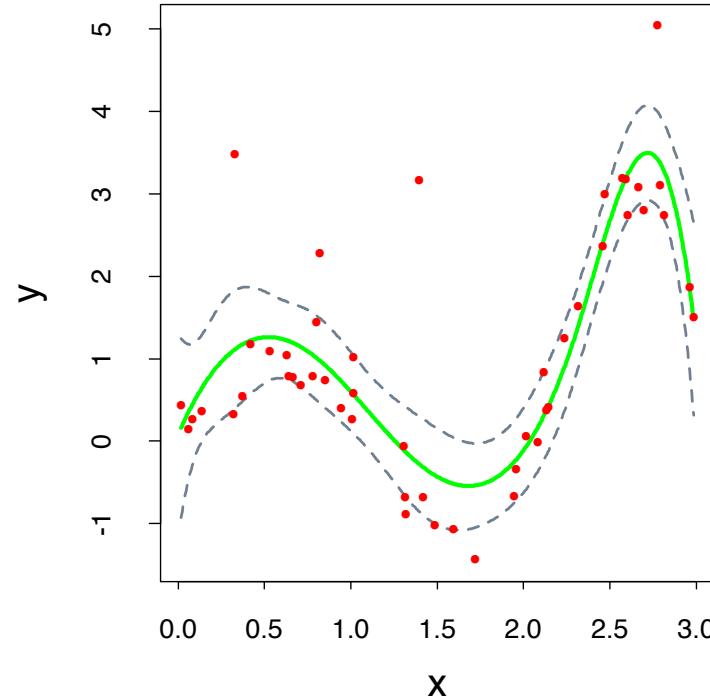
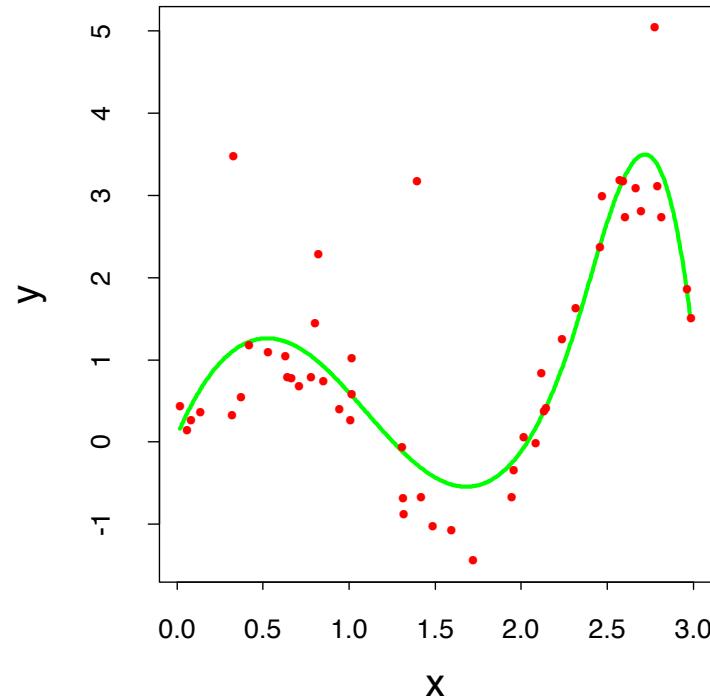


Ejemplo: Bootstrap para obtener un PI

- Suponiendo que

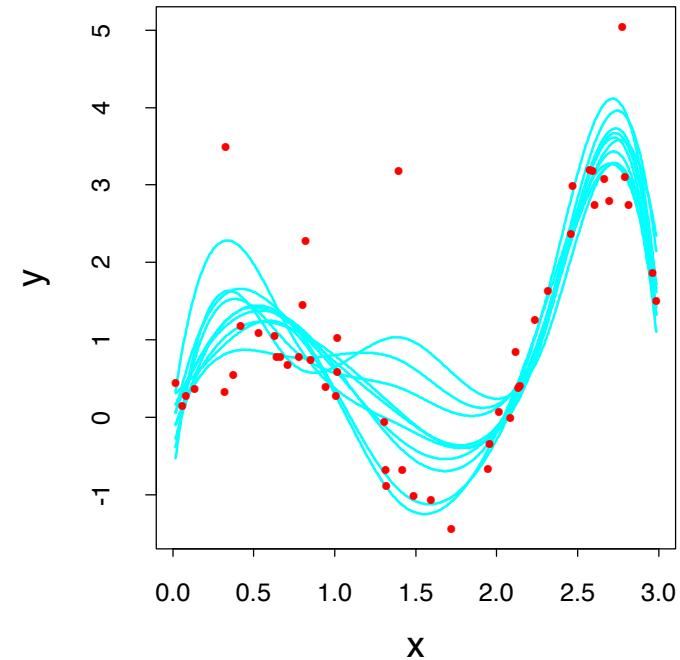
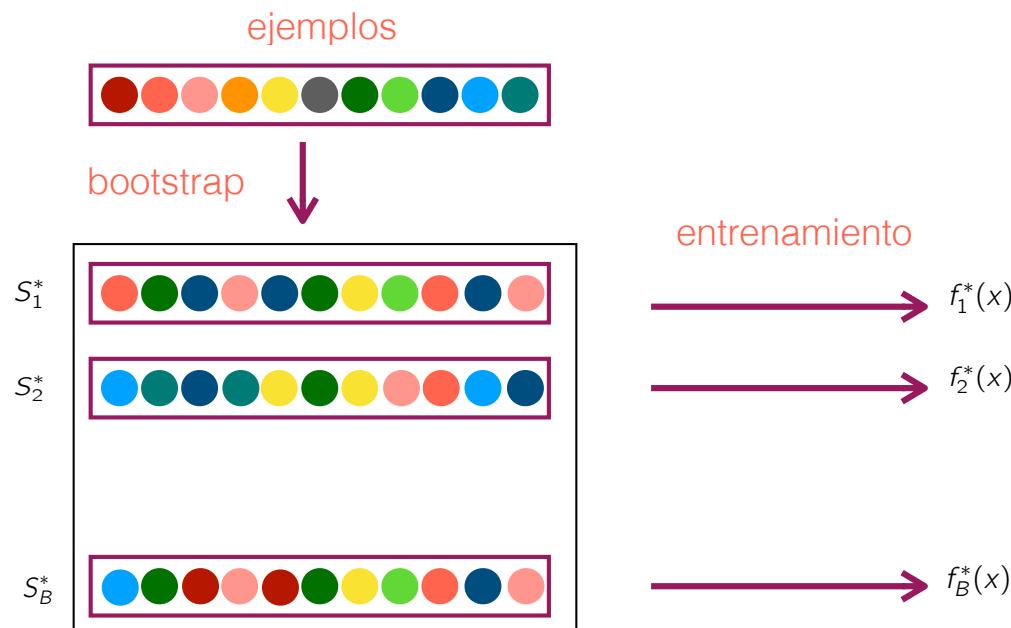
$$y = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

es “fácil” encontrar un intervalo de confianza para el valor de y que corresponde a un determinado x^* .

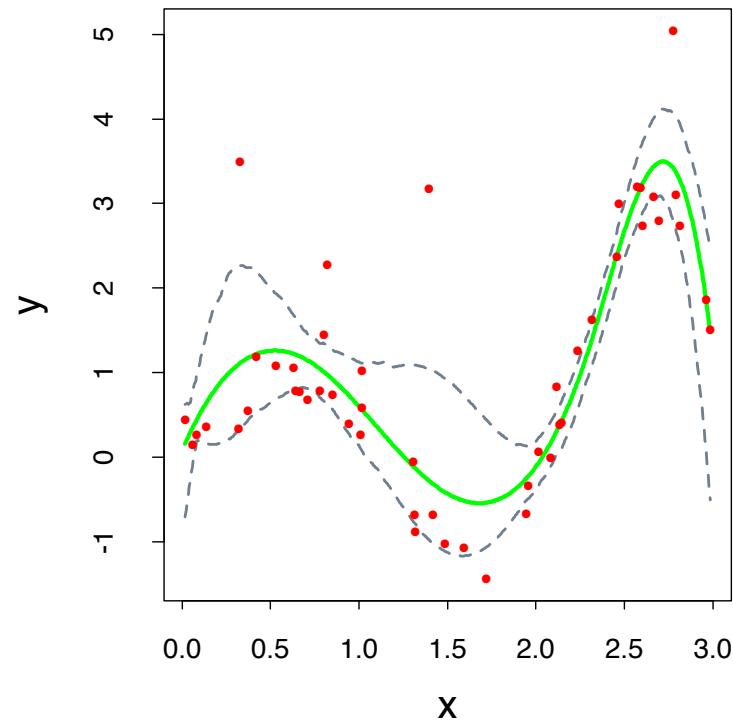
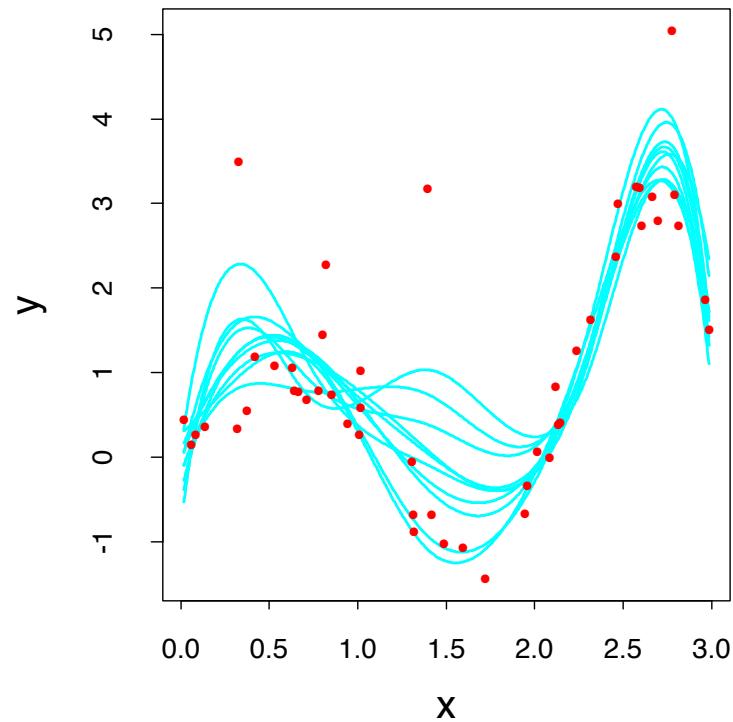


Ejemplo: Bootstrap para obtener un PI

- El bootstrap nos puede servir para encontrar un I.C. para la predicción sin asumir normalidad o en el caso de modelos mucho más complejos que el modelo lineal, para el cual no es “fácil” derivar la distribución de la predicción. En el ejemplo anterior:



Ejemplo: Bootstrap para obtener un PI



Bagging

- Idea: En vez de usar el bootstrap para estimar el sesgo o varianza de un modelo, podemos usarlo para reducir su varianza!
- Recordemos que: si podemos entrenar K modelos de modo que sean independientes y cada uno tiene varianza C, la varianza del promedio de los modelos viene dada por

$$\mathbb{V}(f) = \frac{1}{K^2} \sum_i \mathbb{V}\left(f^{(i)}(x)\right) = \frac{C}{K}$$

- Podemos usar las muestras bootstrap para intentar obtener modelos “aproximadamente independientes”.

Bagging

Ejemplos



1

Bootstrap



S_1^*



S_2^*



...

S_B^*



2

Entrenamiento

$$\longrightarrow f_1^*(x)$$

$$\longrightarrow f_2^*(x)$$

$$\longrightarrow f_B^*(x)$$

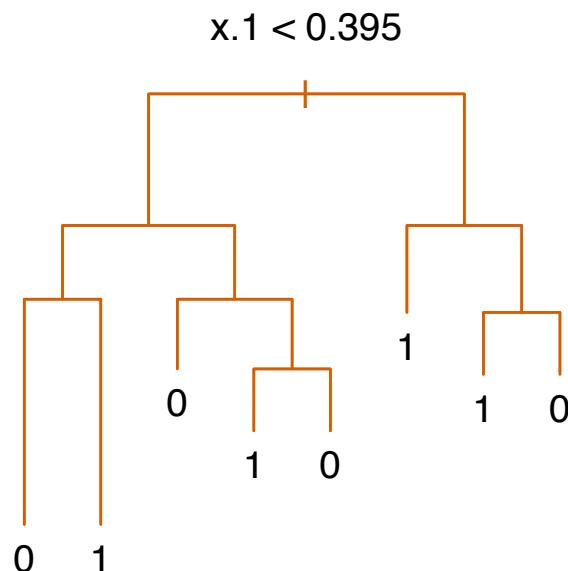
3

Bagged Predictor:

$$f^*(x) = \frac{1}{B} \sum_j f_j^*(x)$$

Ejemplo: Bagged Trees

Original Tree



Data. Simulada, x gausiana estándar $d=5$ con correlación uniforme de 0.95. Clase y binaria generada en base al primer atributo con la siguiente regla:

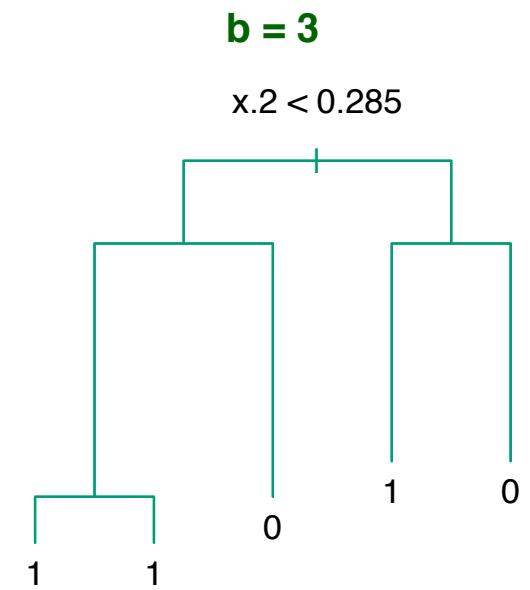
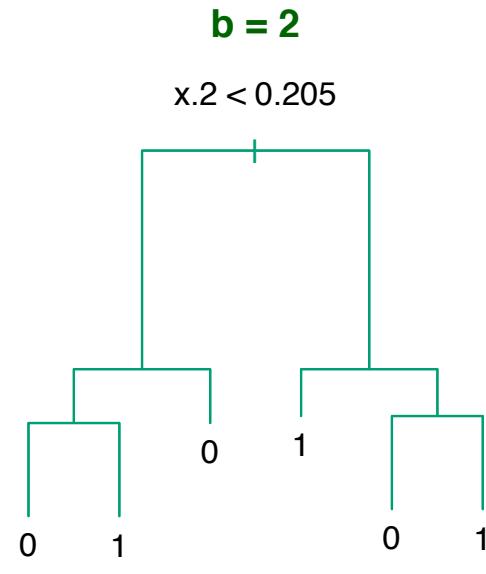
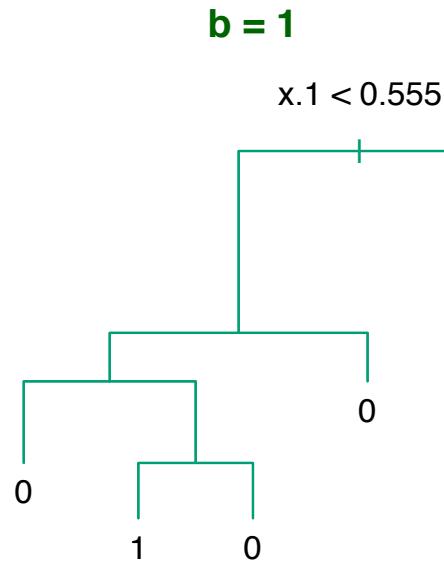
$$P(y = 1|x_1 < 0.5) = 0.2$$

Tree. CART sin pruning construido a partir de $N=30$ ejemplos.

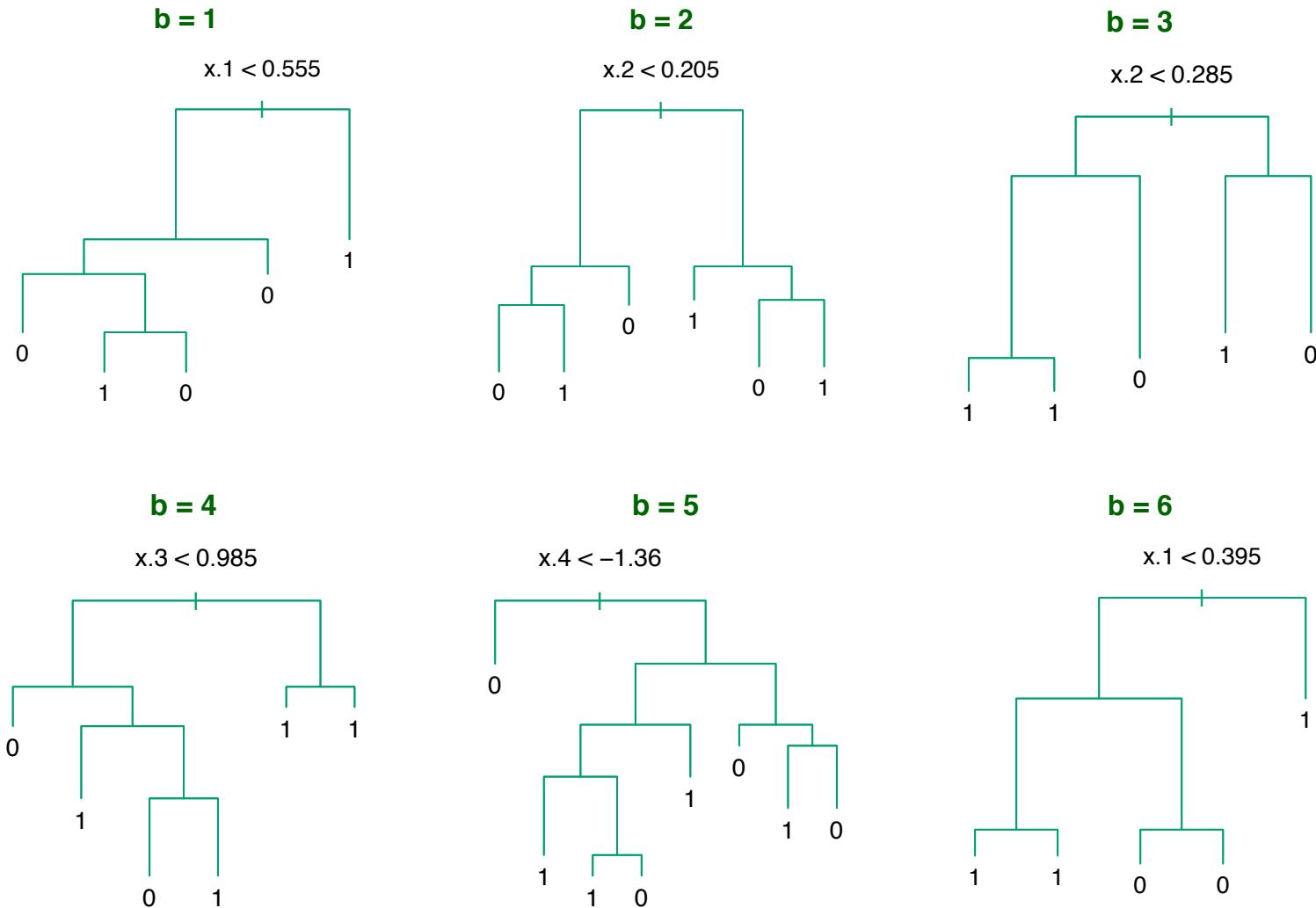
Bagging. $B=200$ remuestreos, CART sin pruning sobre cada remuestreo.

Ejemplo: Bagged Trees

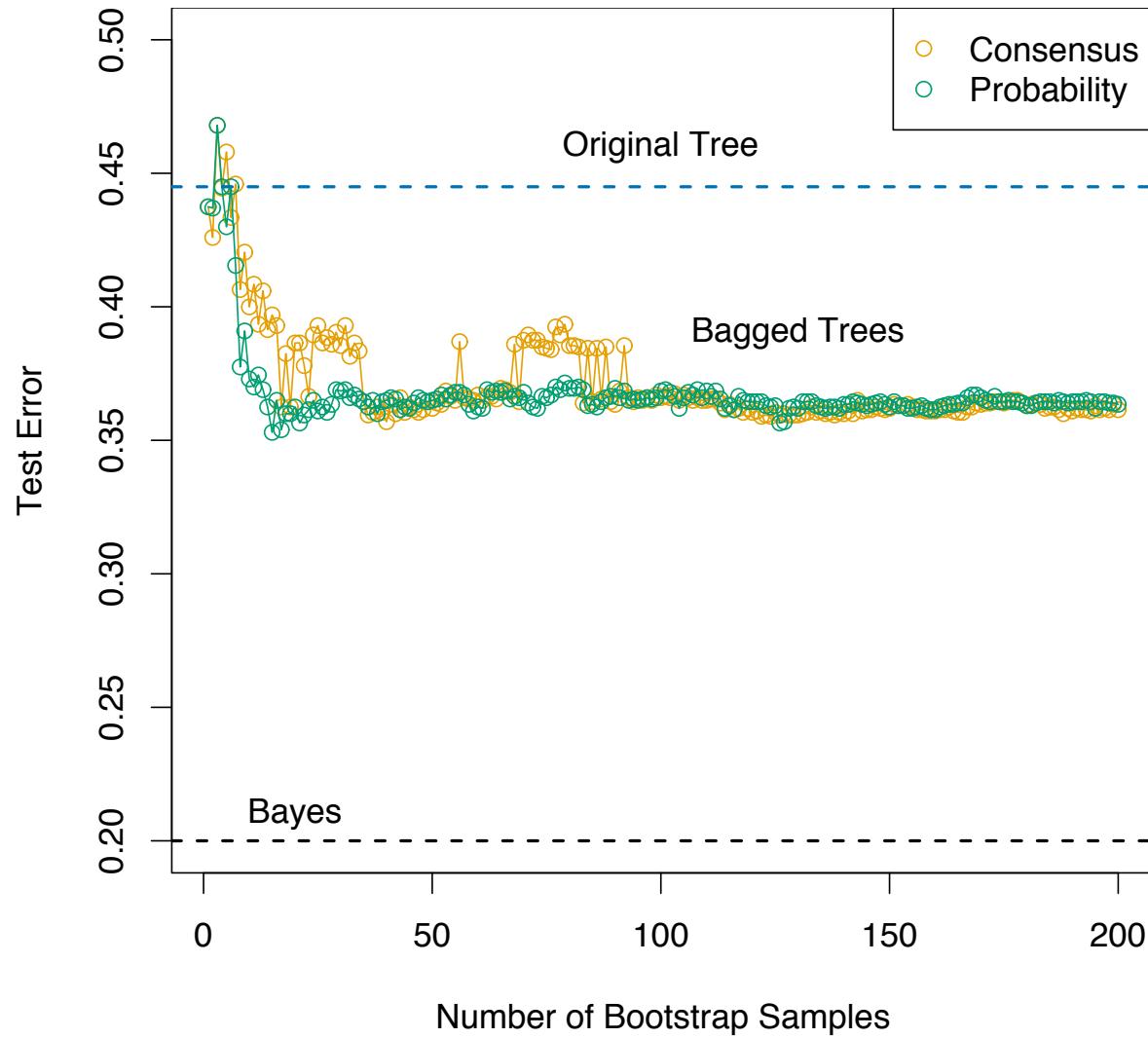
- Algunos árboles obtenidos vía bootstrap



Ejemplo: Bagged Trees

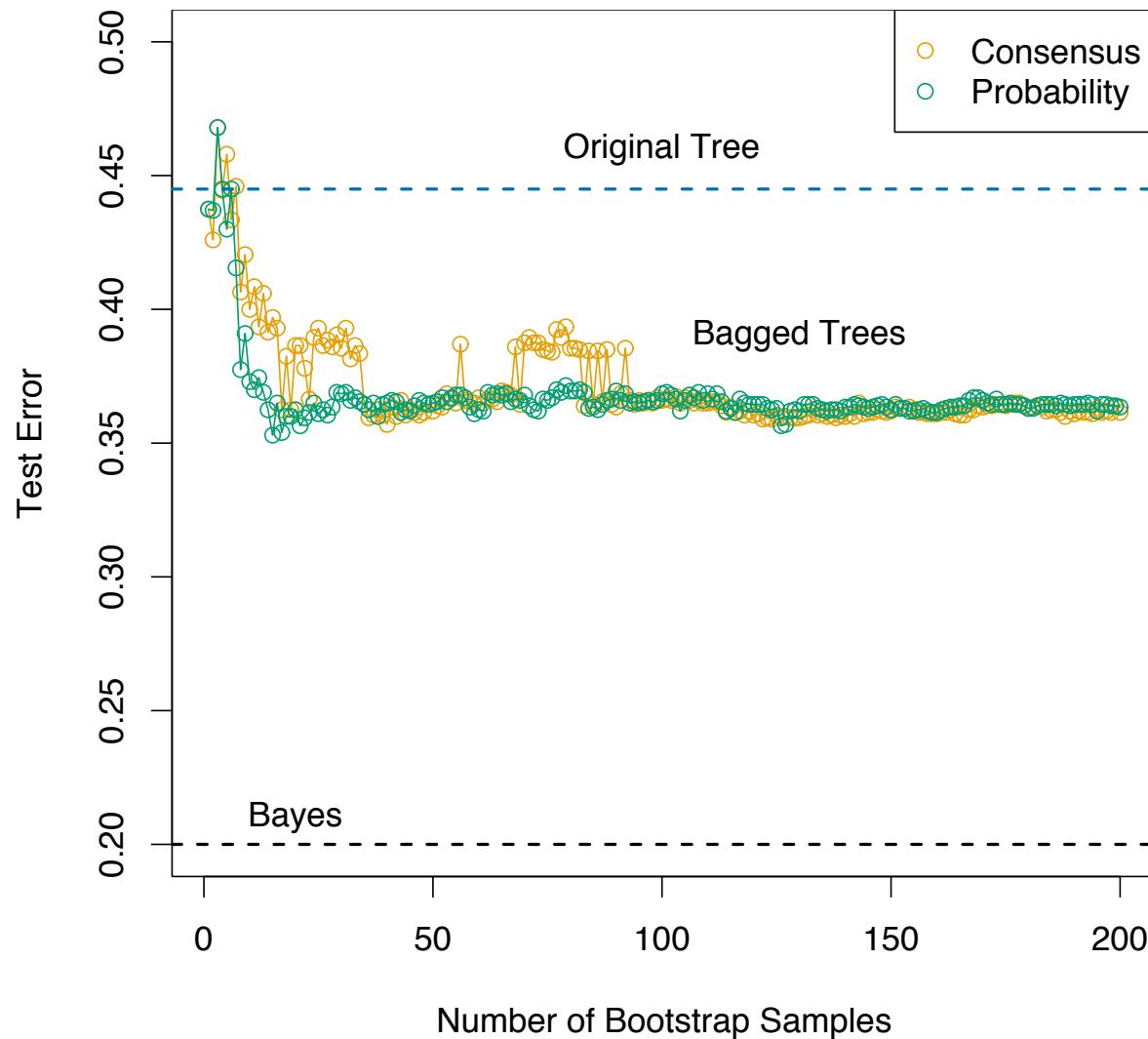


Ejemplo: Bagged Trees



Curva naranja:
Bagged predictor vía
majority voting.

Ejemplo: Bagged Trees



Curva verde: Bagged predictor vía media de las distribuciones en cada hoja)

$$f^*(x) = \frac{1}{B} \sum_j f_j^*(x)$$

Cuándo “Conviene” Bagging?

- Dado que la motivación esencial para Bagging es la posibilidad de reducir la varianza (en la descomposición sesgo-varianza) manteniendo el sesgo:
 - Bagging será más efectivo para un learner con poco sesgo.
 - Además, los efectos de Bagging (respecto a un learner individual) serán más relevantes cuando el learner presente alta varianza.
- Un candidato que reúne estas características es un árbol de decisión (clasificación o regresión) no regularizado o poco regularizado (muchos niveles, sin pruning).

Random Forests

- La varianza del promedio de K variables aleatorias (idénticamente distribuidas) cuando el supuesto de independencia no se cumple, viene dada por

$$\begin{aligned}\text{var}\left(\frac{1}{K} \sum_j x_j\right) &= \frac{1}{K^2} \sum_j \text{var}(x_j) + \frac{2}{K^2} \sum_{i < j} \text{covar}(x_i, x_j) \\ &= \frac{\sigma^2}{K} + \frac{2(K-1)K}{K^2} \rho \sigma^2 = \frac{\sigma^2}{K} + \rho \sigma^2 - \rho \frac{\sigma^2}{K} \\ &= \rho \sigma^2 + (1 - \rho) \frac{\sigma^2}{K}\end{aligned}$$

con σ^2 la varianza de cada v.a. y ρ la correlación entre un par cualquiera.

Random Forests

$$\text{var} \left(\frac{1}{K} \sum_j x_j \right) = \rho\sigma^2 + (1 - \rho)\frac{\sigma^2}{K}$$

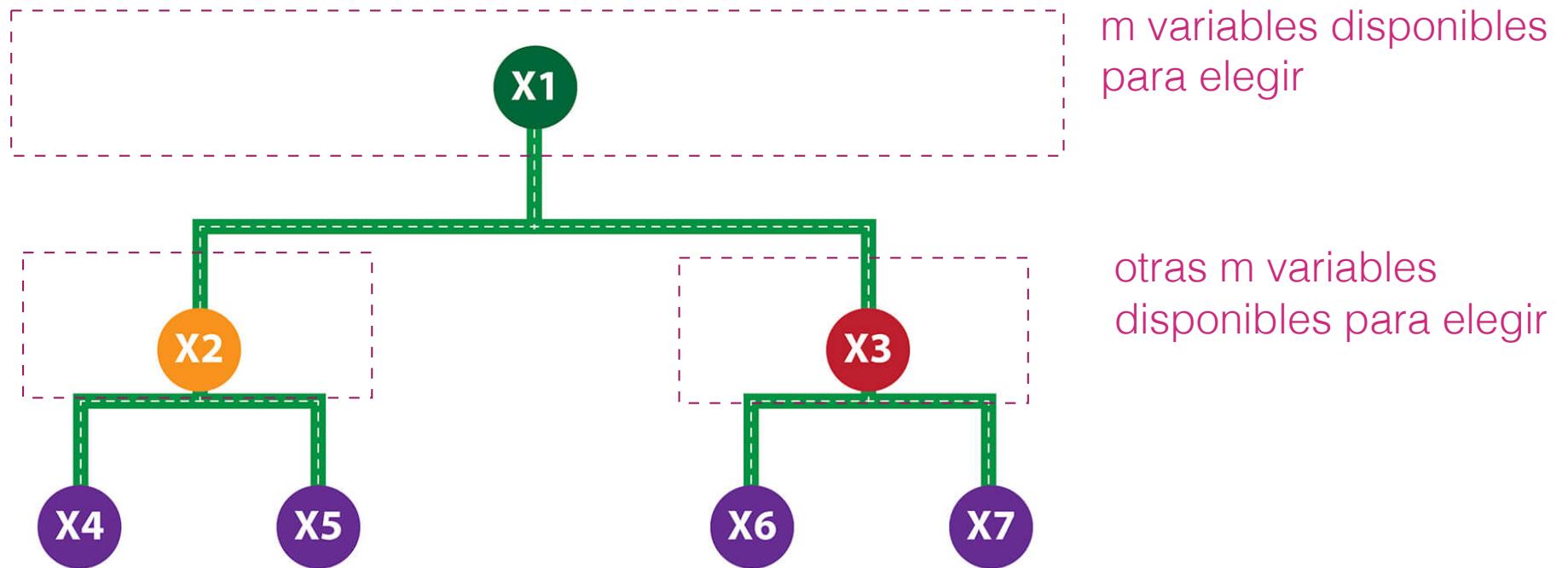
- Si el bien el segundo término de esta expresión va a cero a medida que K se hace grande, existe una componente debida a la correlación que permanece igual.
- La idea de random forests (RF) es mejorar Bagging, reduciendo esa componente de la varianza del ensemble.
- Para implementar esta idea RF combinará el re-muestreo de ejemplos con el muestreo de atributos.

Random Forests

Algorithm 1: Construction of a Random Forest.

```
1 for  $b = 1, \dots, B$  do
2   Generar una muestra Bootstrap  $S_b^*$  de la muestra original  $S$ .
3   Inicializar un árbol  $T_b(x)$  como un nodo hoja  $s$  correspondiente región
   trivial  $\mathbb{R}^d$ .
4   while Existan hojas  $s$  con soporte  $n(s) \geq n_{min}$  en  $T_b$  do
5     for Cada hoja  $s$  del árbol do
6       if  $s$  tiene soporte  $n(s) \geq n_{min}$  then
7         Seleccionar  $m$  variables  $V$  de las  $d$  disponibles.
8         Elegir el mejor split de  $s$  usando solo las variables en  $V$ .
9         Dividir  $s$  en dos hijas.
10  Devolver el bosque  $\{T_b(x)\}_{b=1}^B$ 
```

Random Forests



Random Forests

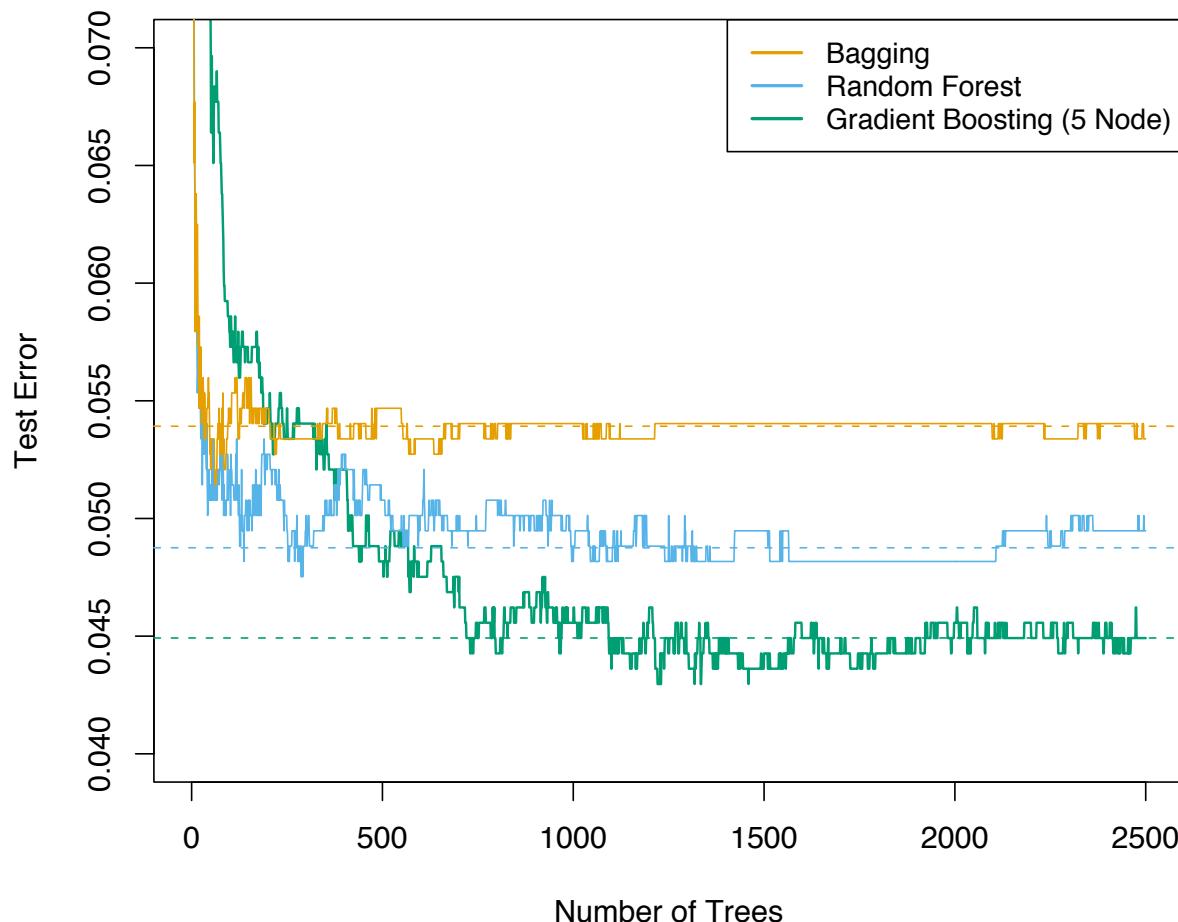
- En regresión el bosque ensambla usando típicamente el promedio simple de las hipótesis correspondientes a cada árbol

$$T_{rf}(x) = \frac{1}{B} \sum_b T_b(x)$$

- En clasificación, cada árbol elige una clase y el ensemble implementa majority voting.
- El crecimiento del árbol tiende a ser promiscuo: si se regulariza demasiado se pierde la sensibilidad del learner al re-muestreo.
- Los autores aconsejan usar $m = \lfloor \sqrt{d} \rfloor$, $n_{\min} = 1$ en clasificación, y $m = \lfloor d/3 \rfloor$, $n_{\min} = 5$ en regresión.
- En la práctica se usa un gran número de árboles.

Random Forests

- Comparación Bagging/RF en un problema real de clasificación de correo spam (ver página 604, texto guía por detalles).



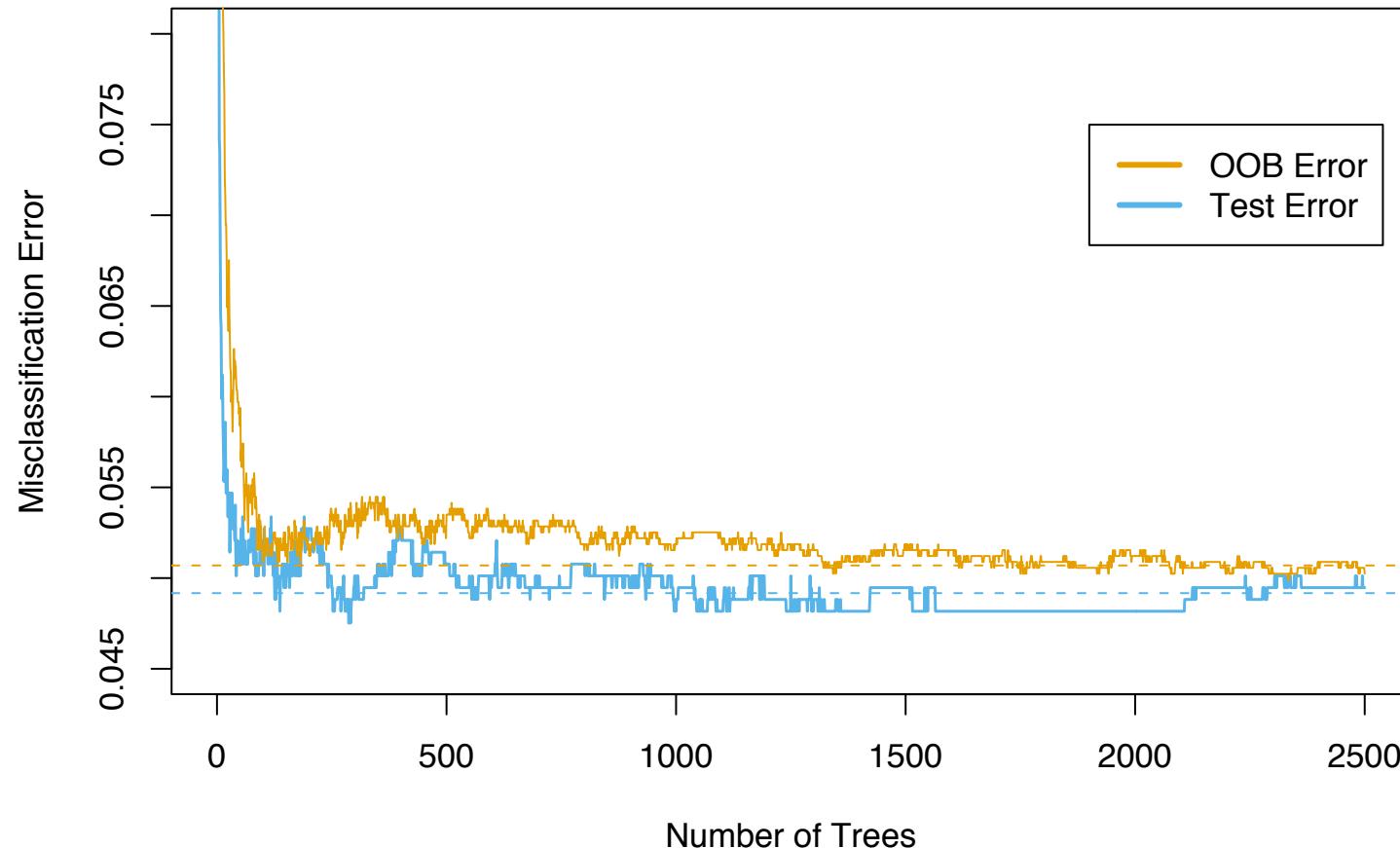
Out-of-Bag (OOB) Estimates

- Una de las cosas más atractivas de Bagging y RF es la posibilidad de construir estimadores del error de predicción “gratis”.
- La idea es que, en cada re-muestreo bootstrap, hay un subconjunto de ejemplos de entrenamiento que no son seleccionados

$$\left(1 - \frac{1}{n}\right)^n \underset{n \rightarrow \infty}{\rightarrow} e^{-1} \approx 0.36$$

- Estos ejemplos se pueden usar como “conjunto de validación”.
- **OOB estimate:** Más específicamente, el error de predicción de un bosque se puede estimar usando todo el conjunto de entrenamiento, pero calculando la predicción para un ejemplo sólo a partir de los árboles que no vieron ese ejemplo.

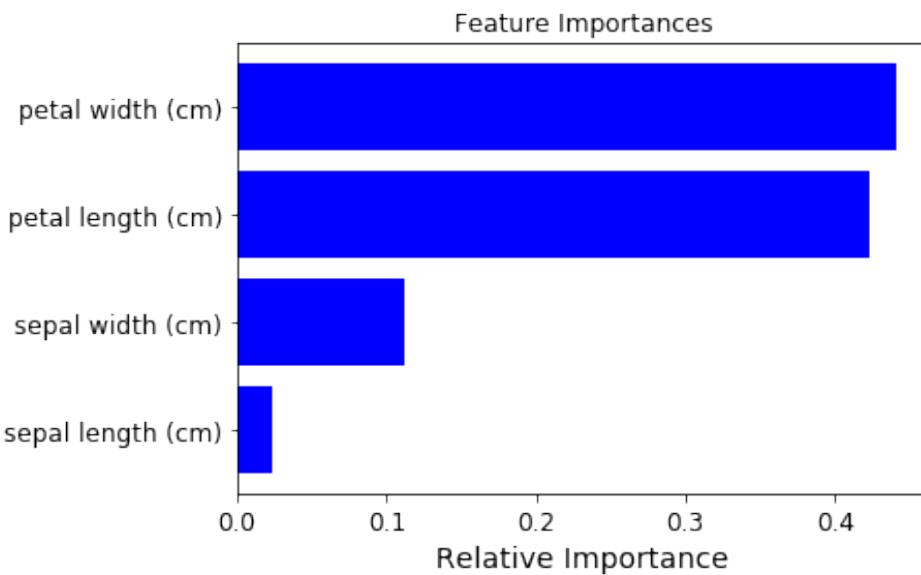
Out-of-Bag (OOB) Estimates



- El estimador obtenido suele diferir poco de K-fold cross-validation.

Gráficos de Importancia

- Otro de los “productos” que hacen a RF muy popular en aplicaciones es que el bosque permite obtener fácilmente una estimación sobre la “importancia” de cada atributo en el modelo.



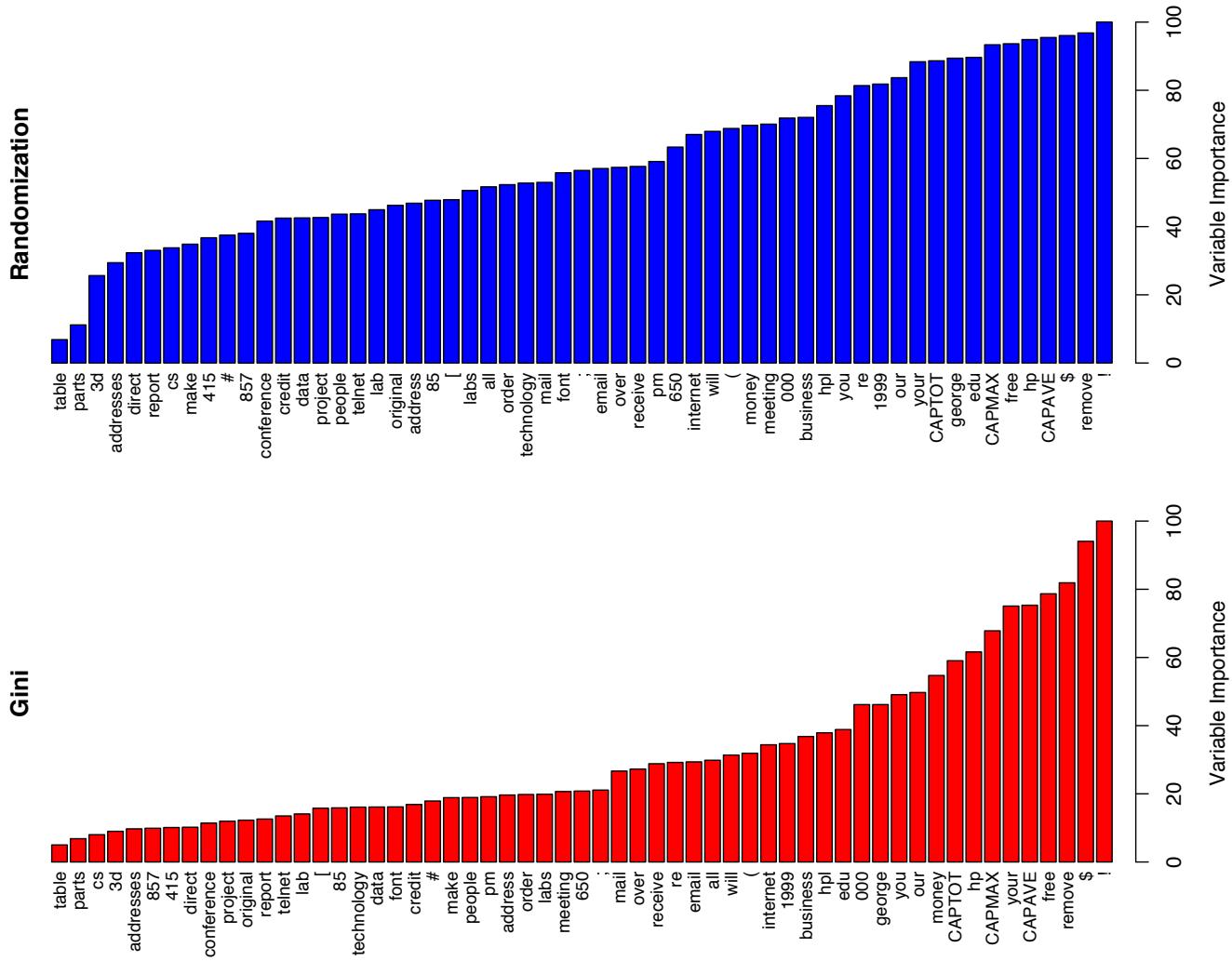
Gráficos de Importancia

- En cada nodo interno de un árbol, el “puntaje” de la variable **usada para implementar el split** se puede calcular midiendo cuánto se redujo el error (o la función objetivo que se haya adoptado) con respecto a la madre.
- La importancia de una variable se puede calcular “acumulando” esos puntajes sobre todos los árboles del bosque, y sobre todos los nodos internos a medida que se construyen.
- Esta idea por supuesto se puede aplicar a otros ensembles de árboles, pero como RF aleatoriza la selección de los atributos, el puntaje final de cada variable es “menos propenso a overfitting” (una variable tiene más posibilidades de entrar en un nodo porque el criterio ese vuelve menos greedy).

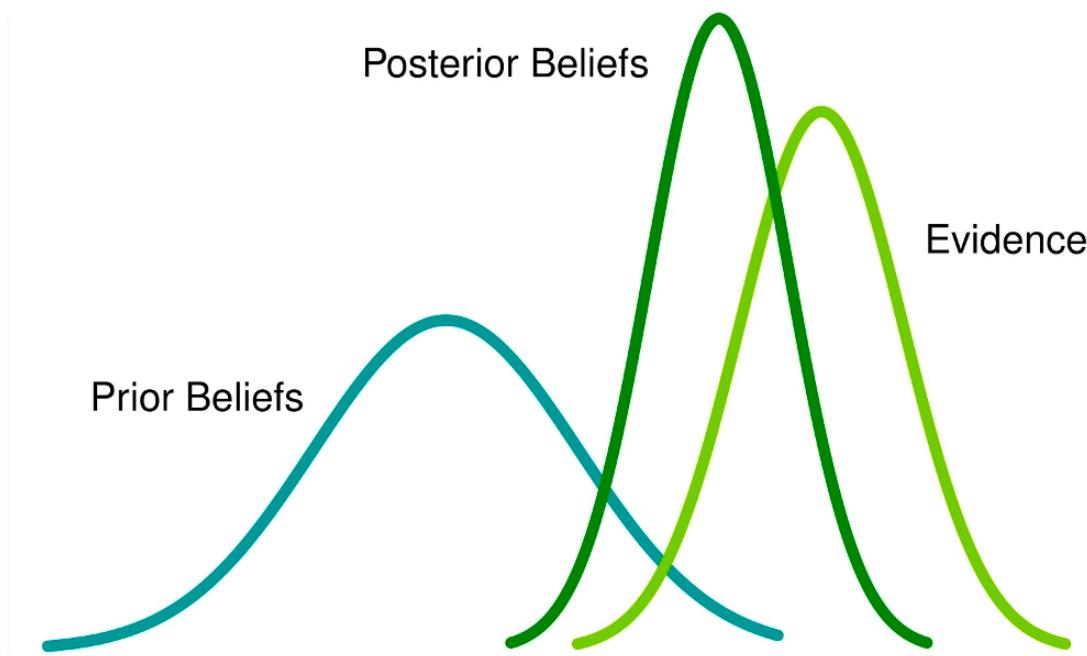
Gráficos de Importancia

- En RF, la idea anterior se suele combinar con la idea de los OOB.
- Al momento de calcular el error sobre un conjunto de ejemplos OOB para un determinado árbol, ejecutar este procedimiento sobre cada nodo interno:
 - Permutar aleatoriamente los valores de la variable que corresponda al split de ese nodo.
 - Medir la importancia de la variable como la diferencia entre el error sobre los ejemplos OOB antes y después de permutar.
- La importancia de una variable se obtiene acumulando los valores calculados sobre los distintos nodos de internos de todos los árboles del bosque, usando los ejemplos OOB correspondientes a cada árbol.

Gráficos de Importancia



Perspectiva Bayesiana

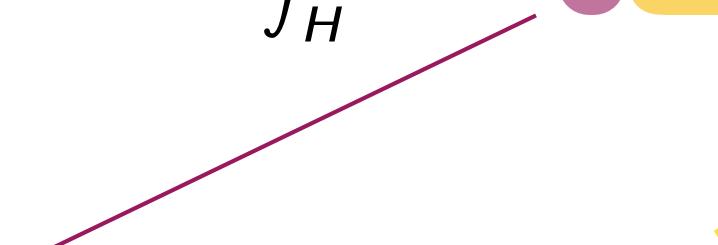


Perspectiva Bayesiana

- Desde un punto de vista bayesiano, el método “óptimo” para ensamblar consiste en calcular:

$$P(y|x) = \int_H P(y|x, f) P(f) df$$

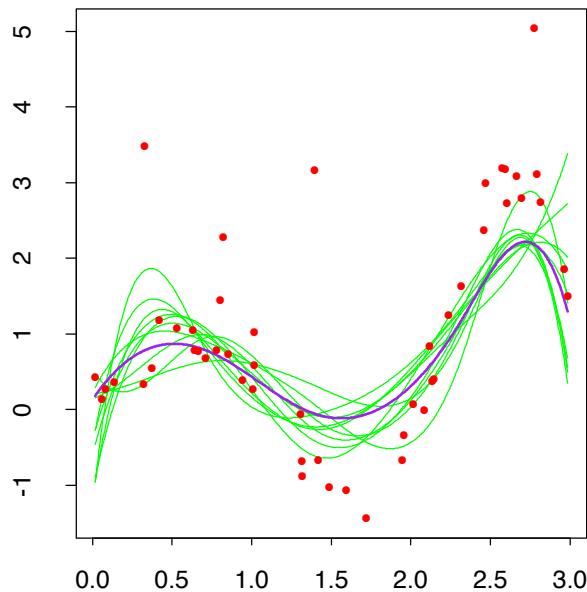
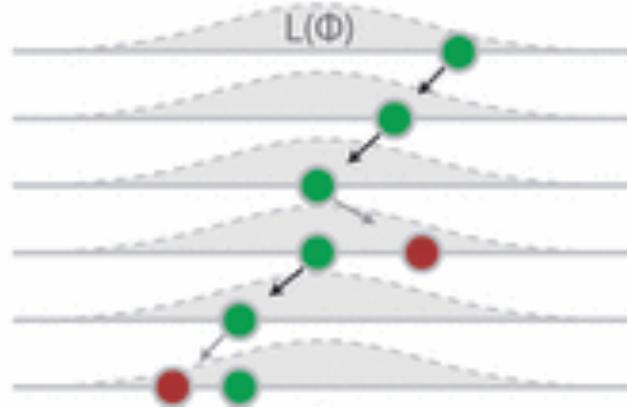
modelos posibles “probabilidad de que ese sea el modelo correcto”



- Naturalmente, este cálculo es poco factible computacionalmente (implica ensamblar un número potencialmente infinito de modelos).

Perspectiva Bayesiana

- Hemos dicho que el bootstrap es un modo computacional para estimar la distribución de un estimador.
- Bagging puede verse entonces como un método para estimar la aproximar la integral del “ensemble ideal” que sale de la perspectiva Bayesiana, vía muestreo.



Boosting



Un Poco de Historia

- Método propuesto por Yoav Freund (Premio Gödel 2003) & Robert Shapire entre 1990 y 1996 para mejorar el desempeño de un clasificador. De acuerdo a Leo Breiman (autor de Bagging, RF y CART): “best off-the-shelf classifier in the world”.



Un Poco de Historia

- En realidad Boosting es una familia de métodos, de los cuales su principal exponente es AdaBoost.
- La primera versión de Boosting (1990) funcionaba entrenando 3 “instancias” de un clasificador:
 - Clasificador c1: se entrena normalmente.
 - Clasificador c2: se entrena re-muestreando el dataset, de modo que la probabilidad de elegir un ejemplo mal clasificado por c1 es $1/2$ y la probabilidad de elegir otro ejemplo es $1/2$.
 - Clasificador c3: se entrena re-muestreando sólo de los ejemplos para los cuales c1 y c2 no están de acuerdo.

Un Poco de Historia

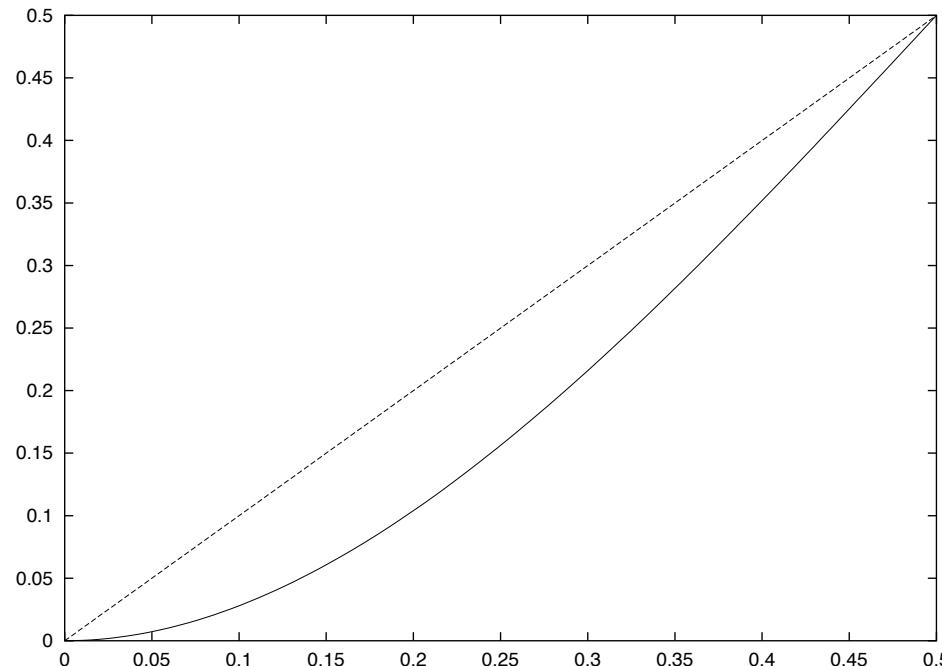
Teorema (1990)

Supongamos que los errores de c_1, c_2, c_3 se pueden acortar por p. Entonces, la probabilidad P de clasificar incorrectamente un ejemplo de entrenamiento combinando c_1, c_2, c_3 mediante *majority voting*, se puede acotar como sigue

$$P \leq 3p^2 - 2p^3$$

- Remark: esto vale sobre el conjunto de entrenamiento.

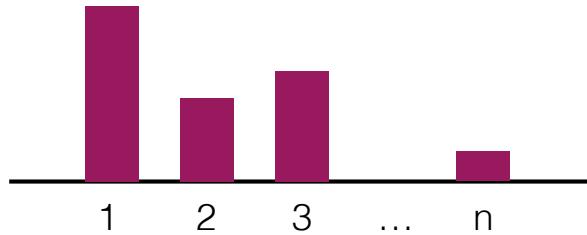
Un Poco de Historia



- **Idea interesante:** un clasificador se puede entrenar explícitamente para corregir los errores de otro. Esto es muy diferente de Bagging en que la “complementariedad” entre learners es sólo implícita.

Adaboost (Adaptive Boosting)

- La idea es mantener una distribución de probabilidad D sobre los datos de entrenamiento para re-muestrearlos adaptivamente.



- Se entrena un conjunto de clasificadores en secuencia, modificando la distribución, para aumentar la probabilidad de seleccionar los ejemplos que se estén clasificado mal.
- Dependiendo de su desempeño, cada clasificador obtiene un peso, que usará para “votar” por una clase determinada, cuando sea necesario clasificar un dato.

Adaboost (Discreto)

Algorithm 1: Adaboost (2 clases).

Input : Ejemplos $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, Tamaño del Ensemble N .

Output: Ensemble $F(x)$.

1 Inicializar la distribución D sobre S como $D_1(j) = 1/n, \forall j$.

2 **for** $i = 1, \dots, N$ **do**

3 Generar un conjunto de ejemplos S_i^* remuestreando S con pesos $D_i(j)$.

4 Entrenar un clasificador $f_i : \mathbb{X} \rightarrow \{-1, 1\}$ con S_i^* .

5 Determinar la probabilidad de error sobre S_i^* ,

$$\epsilon_i = \sum_j D_i(j) I(y^{(j)} \neq f_i(x^{(j)}))$$

6 Determinar el peso de f_i como $\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right)$.

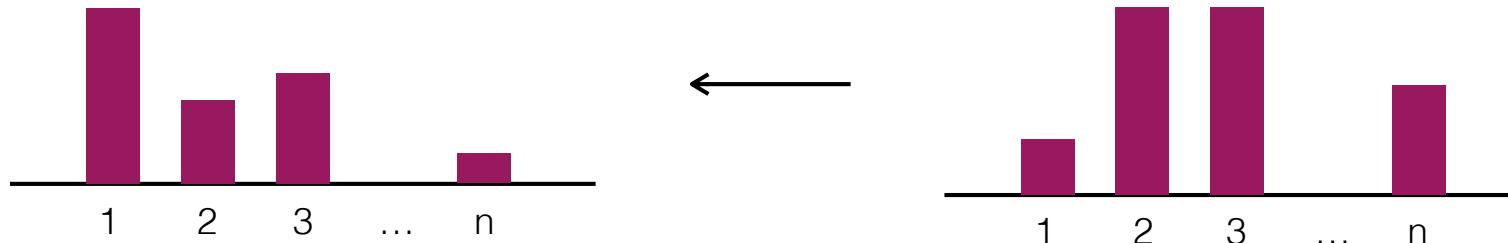
7 Actualizar la distribución D como

$$D_{i+1}(j) = \frac{D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)}))}{Z_i} \quad (1)$$

con $Z_i = \sum_j \exp(\alpha_i y^{(j)} f_i(x^{(j)}))$.

8 Devolver $F(x) = \text{sign} \left(\sum_i \alpha_i f_i(x) \right)$.

Adaboost - Distribución



$$D_{i+1}(j) = \frac{D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)}))}{Z_i}$$

- El componente esencial en el update de la distribución que mantiene Adaboost es un término conocido
- Si $f_i(x)$ representa un hiperplano normalizado, este producto es exactamente el **margen de clasificación** correspondiente al j-ésimo punto de entrenamiento.

Adaboost - Distribución

- En la definición que hemos dado de Adaboost cada hipótesis individual es discreta (por eso Discrete Adaboost)

$$f_i : \mathbb{X} \rightarrow \{-1, 1\}$$

- Como además la etiquetas han sido codificadas como ± 1

$$y^{(j)} f_i(x^{(j)}) = \begin{cases} +1 & \text{si } y^{(j)} = f_i(x^{(j)}) \\ -1 & \text{si } y^{(j)} \neq f_i(x^{(j)}) \end{cases}$$

y obtenemos que:

$$D_{i+1}(j) = \begin{cases} \propto D_i(j) \exp(-\alpha_i) & \text{si } y^{(j)} = f_i(x^{(j)}) \\ \propto D_i(j) \exp(+\alpha_i) & \text{si } y^{(j)} \neq f_i(x^{(j)}) \end{cases}$$

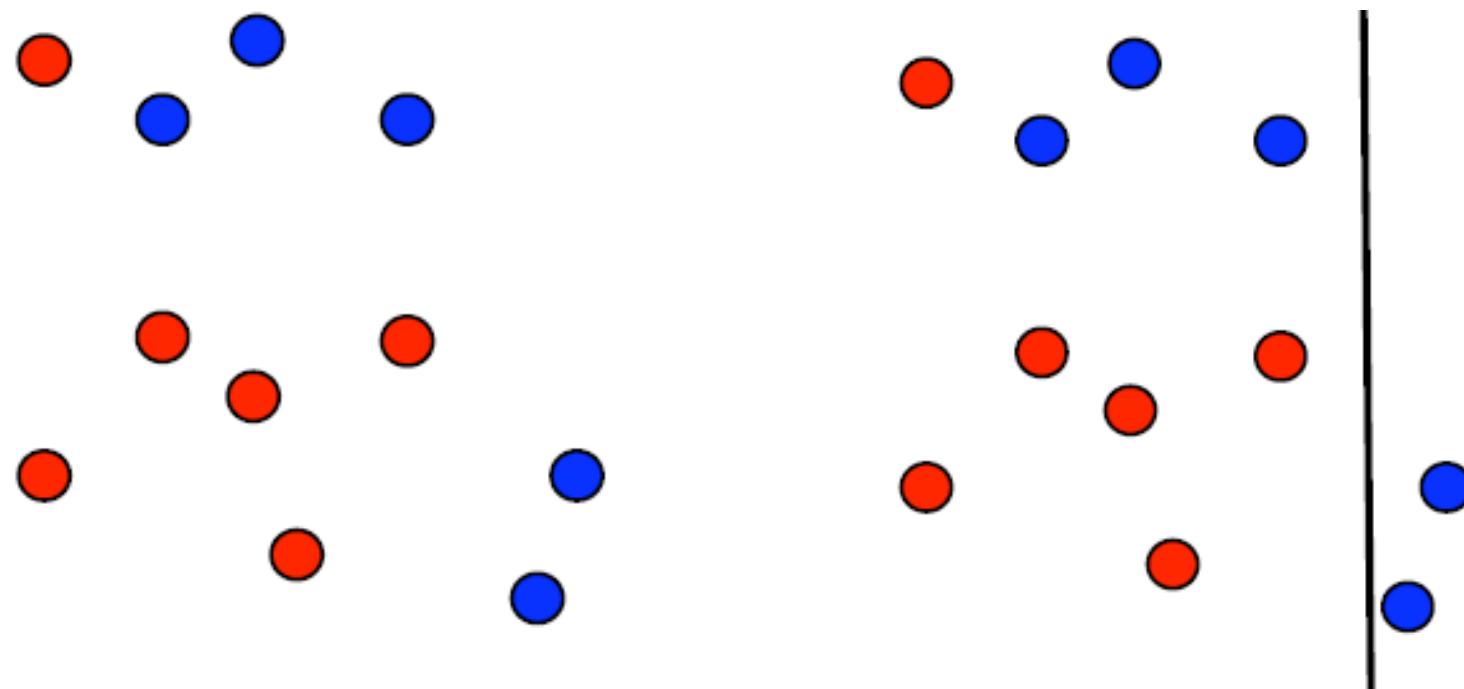
Adaboost - Distribución

$$D_{i+1}(j) = \begin{cases} \propto D_i(j) \exp(-\alpha_i) & \text{si } y^{(j)} = f_i(x^{(j)}) \\ \propto D_i(j) \exp(+\alpha_i) & \text{si } y^{(j)} \neq f_i(x^{(j)}) \end{cases}$$

- Por lo tanto, en cada iteración, la masa de probabilidad concentrada en un punto j se modifica dependiendo del desempeño general del clasificador y de su desempeño específico en ese punto.
- Asumiendo $\alpha_t > 0$, los ejemplos mal clasificados aumentan su probabilidad de ser seleccionados en la ronda siguiente y los ejemplos bien clasificados la disminuyen.

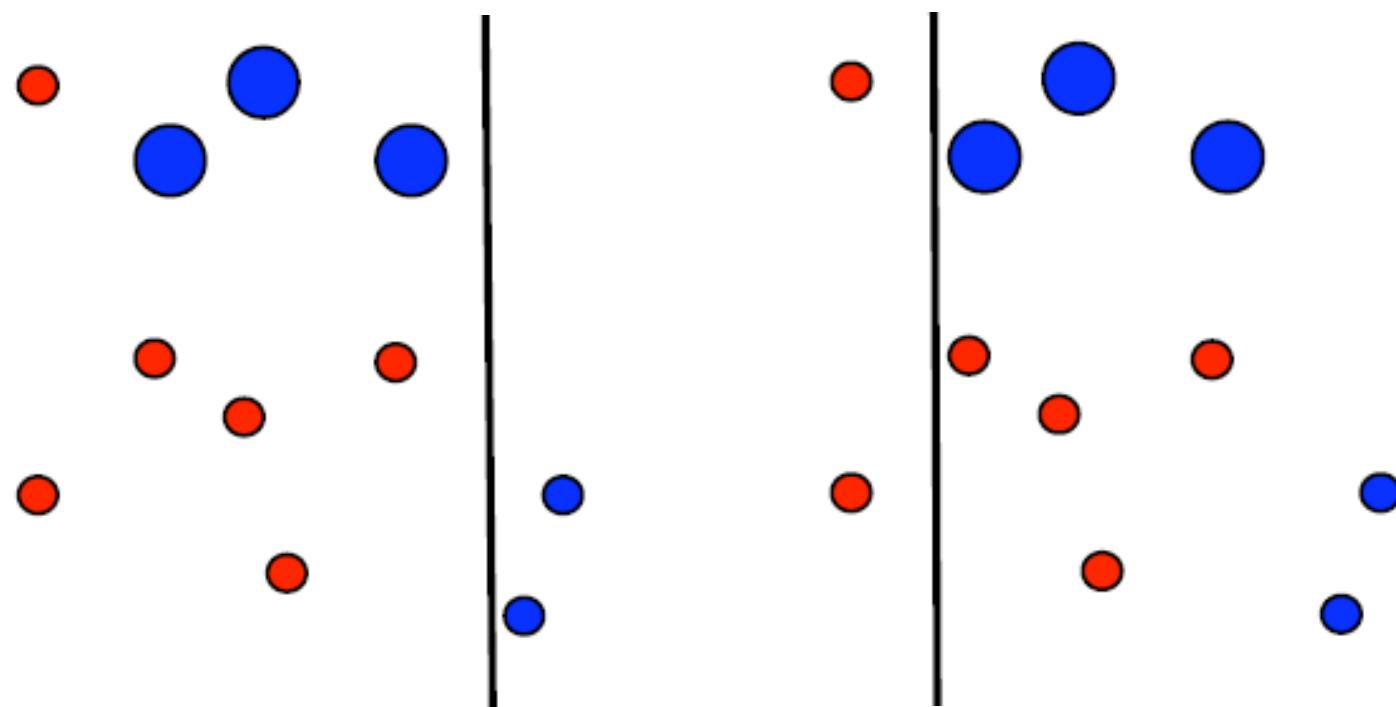
Ilustración

$t=1$



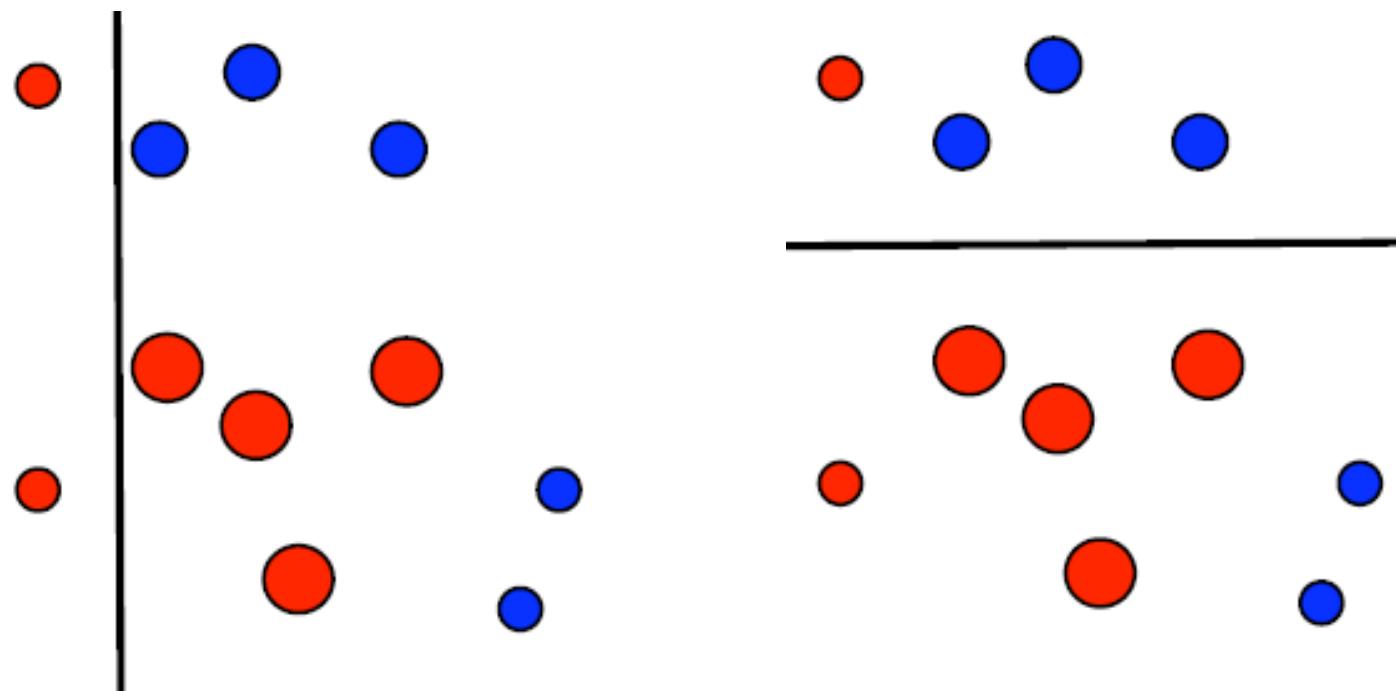
Ilustración

$t=2$

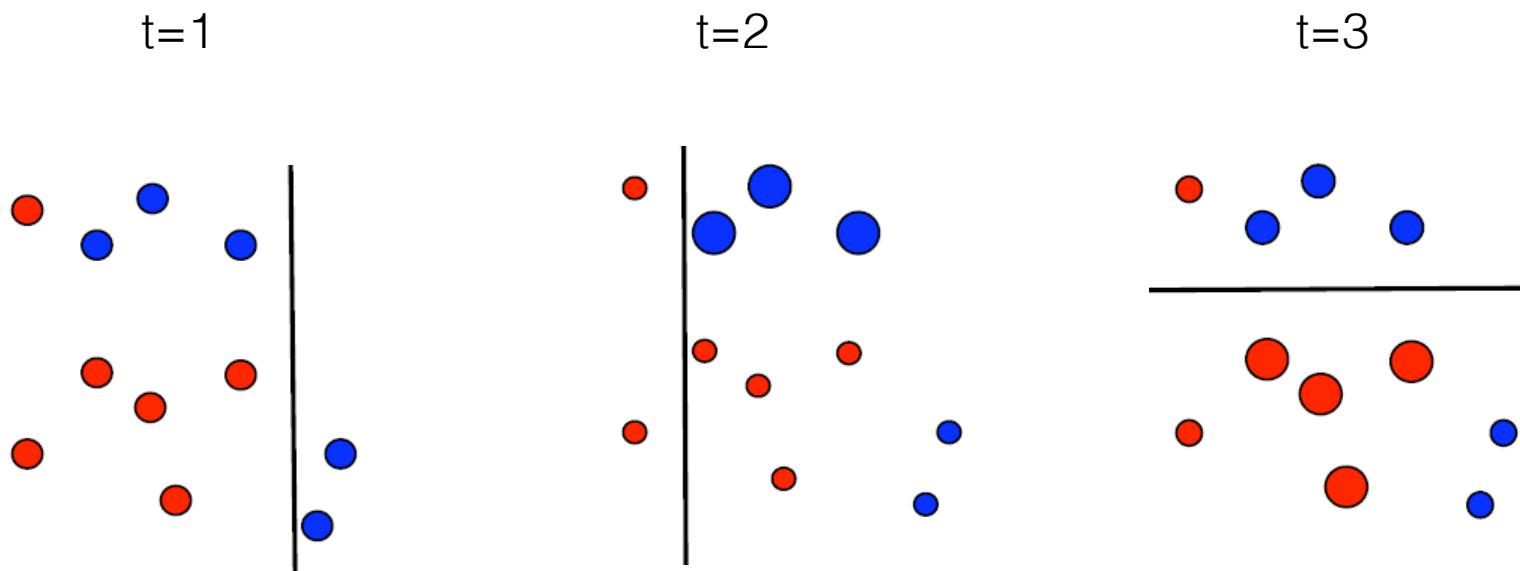


Ilustración

$t=3$

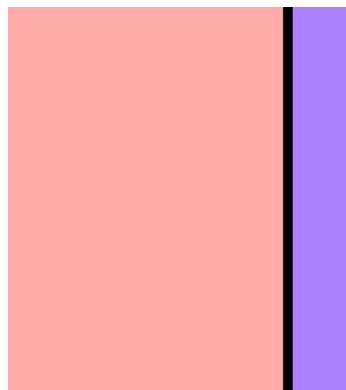


Ilustración

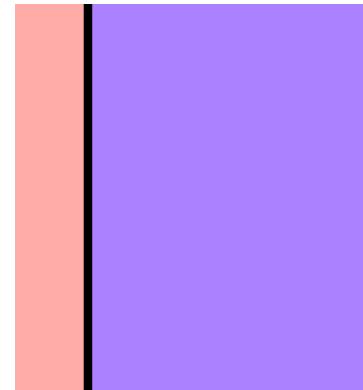


Ilustración

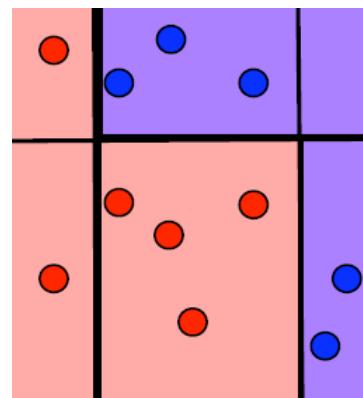
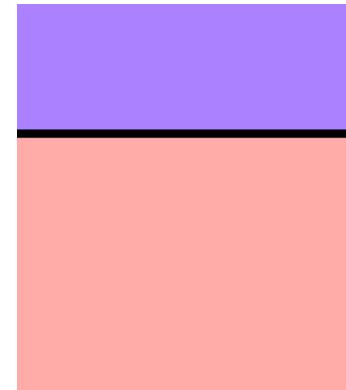
t=1



t=2



t=3



Adaboost - Errores

- El error de un clasificador se calcula “pesando” los errores en cada ejemplo de acuerdo a la distribución de esa iteración

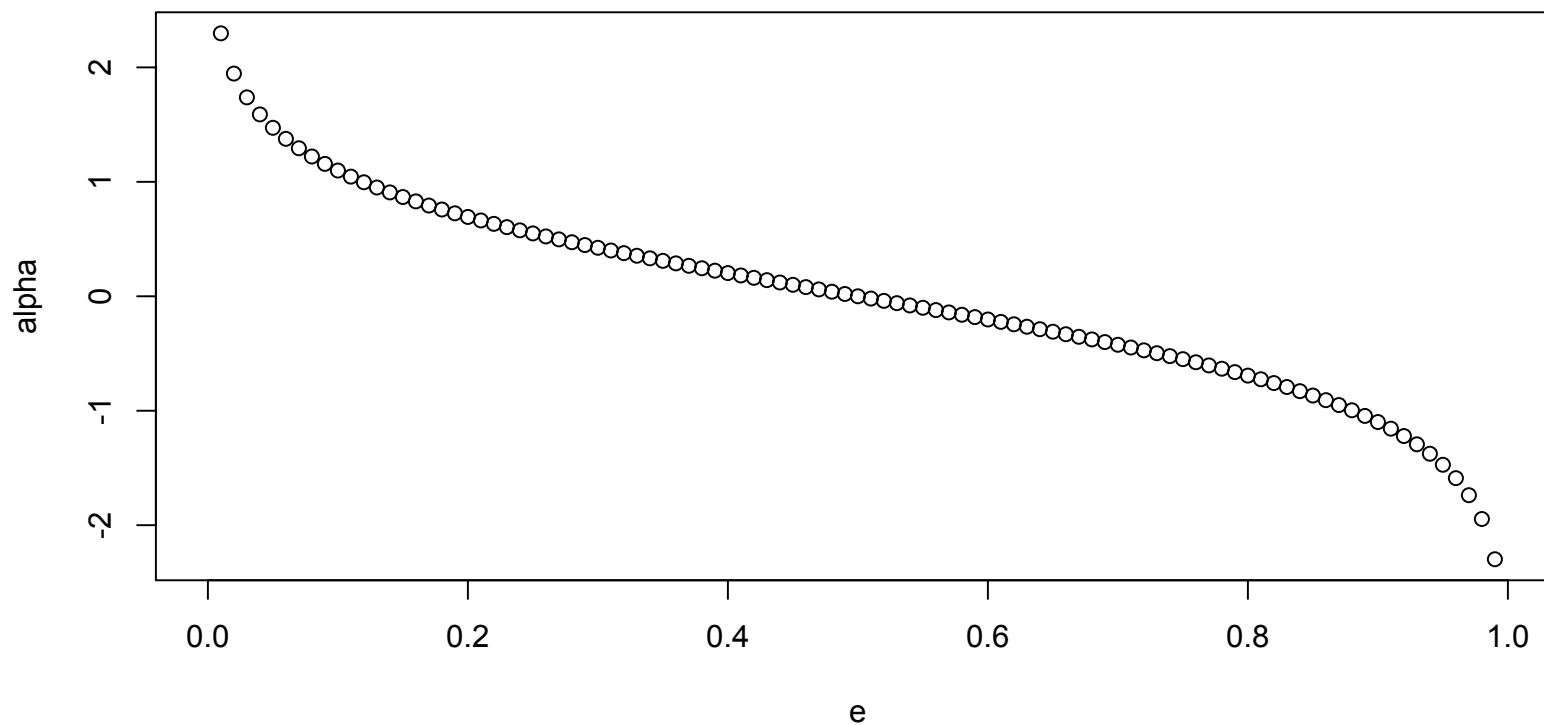
$$\begin{aligned}\epsilon_i &= \sum_j D_i(j) I(y^{(j)} \neq f_i(x^{(j)})) \\ &= P_{D_i}(y^{(j)} \neq f_i(x^{(j)}))\end{aligned}$$

- El “peso” de cada clasificador se calcula usando una función estrictamente decreciente en ϵ_i

$$\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right)$$

Adaboost - Peso de una Hipótesis

- A mayor error, menor peso.
- Si es más probable equivocarse que acertar, el peso es negativo (se invierten las decisiones del clasificador)

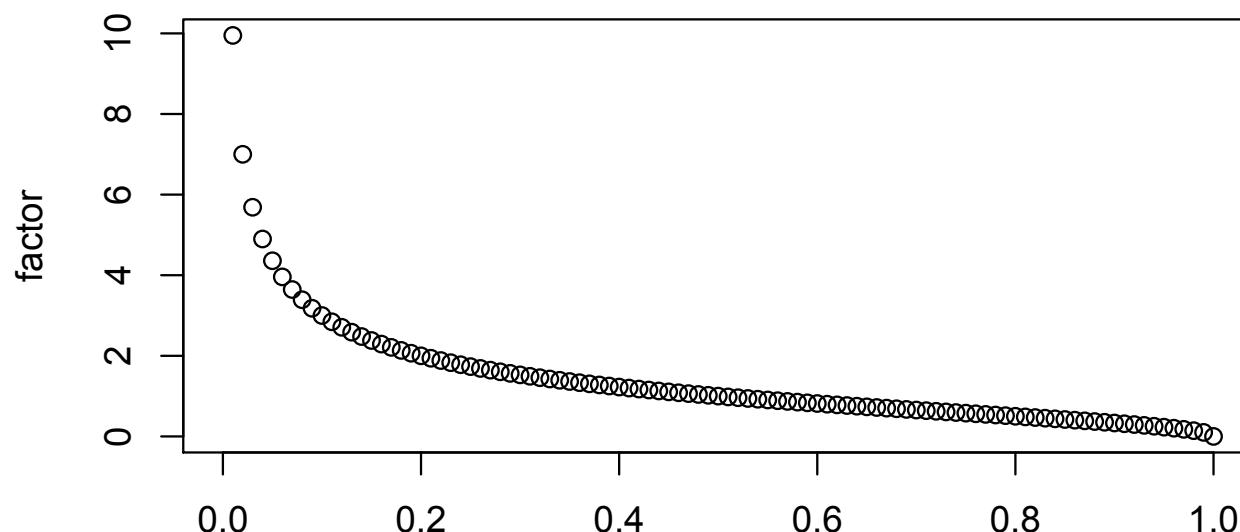


Adaboost - Distribución

- En el caso de ejemplo mal clasificado, el factor que ajusta el “peso” de muestreo del ejemplo para la próxima ronda

$$D_{i+1}(j) = D_i(j) \exp(+\alpha_i)$$

es monótonamente creciente en α_i y por lo tanto, monótonamente decreciente en ϵ_i .

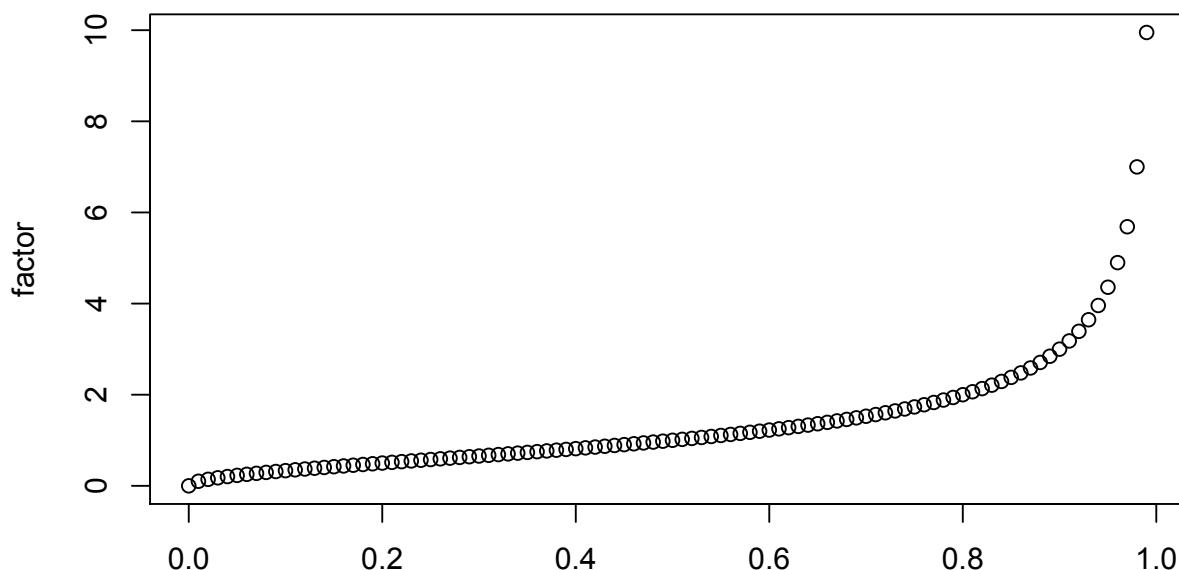


Adaboost - Distribución

- En el caso de ejemplo bien clasificado, el factor que ajusta el “peso” de muestreo del ejemplo para la próxima ronda

$$D_{i+1}(j) = D_i(j) \exp(-\alpha_i)$$

es monótonamente decreciente en α_i y por lo tanto, monótonamente creciente en ϵ_i .



Adaboost - Ensemble

- Notemos que como cada clasificador es discreto, cada clasificador esencialmente vota por una clase y la suma que hemos usado para ensamblar se puede descomponer como

$$F(x) = \text{sign} \left(\sum_i \alpha_i f_i(x) \right)$$

$$F(x) = \text{sign} \left(\sum_{i:f_i(x)=+1} \alpha_i - \sum_{i:f_i(x)=-1} \alpha_i \right)$$

$$F(x) = \begin{cases} -1 & \sum_{i:f_i(x)=-1} \alpha_i > \sum_{i:f_i(x)=+1} \alpha_i \\ +1 & \sum_{i:f_i(x)=-1} \alpha_i \leq \sum_{i:f_i(x)=+1} \alpha_i \end{cases}$$

- La decisión es -1 si y sólo si, la mayoría de los votos (con pesos) soporta esa decisión.

Garantías

Teorema (Freund et al., 1998)

La probabilidad P de que el ensemble construido por AdaBoost clasifique incorrectamente un ejemplo de entrenamiento se puede acotar como sigue

$$P \leq 2^N \prod_{i=1}^N \sqrt{\epsilon_i(1 - \epsilon_i)}$$

- El error del ensamblado decrece exponencialmente rápido.
- Esto vale sobre el conjunto de entrenamiento.

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5), 1651-1686.

Garantías

Teorema (versión alternativa)

Sea $\epsilon_i = \frac{1}{2} - \gamma_i$, $\gamma_i > 0$. Entonces, la probabilidad P de que el ensemble construido por AdaBoost clasifique incorrectamente un ejemplo de entrenamiento se puede acotar como sigue

$$P \leq \prod_{i=1}^N \sqrt{1 - 4\gamma_i^2}$$

- La “ventaja” de cada hipótesis individual con respecto a un clasificador “aleatorio” se amplifica exponencialmente rápido.
- Esto vale sobre el conjunto de entrenamiento.

Demostración Cota Clásica

- Notemos que

$$\begin{aligned} D_{t+1}(j) &= \frac{D_t(j) \exp(-y^{(j)} \alpha_t f_t(x^{(j)}))}{Z_t} \\ &\quad \dots \quad \dots \\ &= \frac{D_1(j) \exp(-y^{(j)} \sum_{i=1}^t \alpha_i f_i(x^{(j)}))}{\prod_{i=1}^t Z_i} \end{aligned}$$

$$D_{t+1}(j) = \frac{\frac{1}{n} \exp(-y^{(j)} F_t(x^{(j)}))}{\prod_{i=1}^t Z_i}$$

Demostración Cota Clásica

- Notemos ahora que el ensemble comete un error de clasificación si y solo si $y^{(j)} F_t(x^{(j)}) < 0$. Pero

$$y^{(j)} F_t(x^{(j)}) < 0 \Rightarrow \exp(-y^{(j)} F_t(x^{(j)})) > 1$$

- De este modo,

$$\begin{aligned} \frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) &\leq \sum_{j:y^{(j)} \neq F_t(x^{(j)})} \frac{1}{n} \exp(-y^{(j)} F_t(x^{(j)})) \\ &\leq \sum_j D_{t+1}(j) \prod_{i=1}^t Z_i = \prod_{i=1}^t Z_i \end{aligned}$$

Demostración Cota Clásica

- Para la versión de Adaboost que estamos usando, la constante normalizadora es fácil de calcular,

$$\begin{aligned} Z_i &= \sum_j D_{i+1}(j) = \sum_j D_i(j) \exp \left(-\alpha_i y^{(j)} f_i(x^{(j)}) \right) \\ &= \sum_{j:y^{(j)}=f_i(x^{(j)})} D_i(j) \exp (-\alpha_t) + \sum_{j:y^{(j)} \neq f_i(x^{(j)})} D_i(j) \exp (\alpha_t) \end{aligned}$$

$$Z_i = \exp (-\alpha_i) (1 - \epsilon_i) + \exp (\alpha_i) \epsilon_i$$

- Pero

$$\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right)$$

Demostración Cota Clásica

- Sustituyendo,

$$\begin{aligned}Z_i &= \sqrt{\frac{\epsilon_i}{1-\epsilon_i}}(1 - \epsilon_i) + \sqrt{\frac{1-\epsilon_i}{\epsilon_i}}\epsilon_i \\&= \sqrt{\frac{\epsilon_i(1-\epsilon_i)^2}{1-\epsilon_i}} + \sqrt{\frac{(1-\epsilon_i)\epsilon_i^2}{\epsilon_i}} \\&= 2\sqrt{\epsilon_i(1 - \epsilon_i)}\end{aligned}$$

- Por lo tanto,

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq 2^T \prod_{i=1}^t \sqrt{\epsilon_i(1 - \epsilon_i)}$$

Optimalidad de los Pesos

- Este análisis revela, de paso, que la elección de los pesos de cada hipótesis es óptima para minimizar la cota del error de clasificación del ensamblado. En efecto, si consideramos

$$Z_i = \exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i$$

y optimizamos en α_i

$$\begin{aligned}\frac{\partial Z_i}{\partial \alpha_i} &= -\exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i = 0 \\ \Rightarrow -\alpha_i + \ln(1 - \epsilon_i) &= \alpha_i + \ln \epsilon_i \\ \Rightarrow 2\alpha_i &= \ln(1 - \epsilon_i) - \ln \epsilon_i = 0 \\ \Rightarrow \alpha_i &= \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right)\end{aligned}$$

Optimalidad de los Pesos

- La segunda derivada de Z_i es:

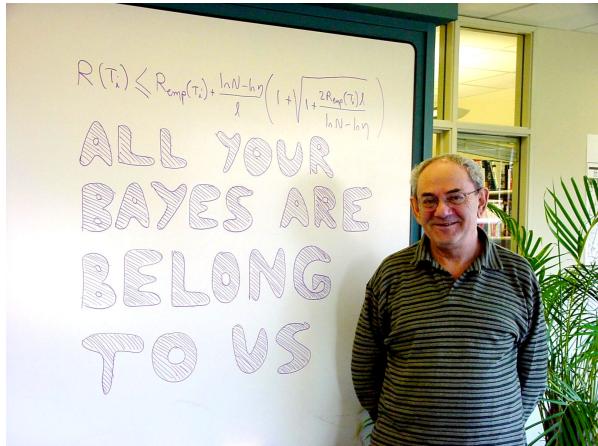
$$\frac{\partial^2 Z_i}{\partial \alpha_i^2} = \exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i$$

- Evaluando en el punto crítico:

$$\left. \frac{\partial^2 Z_i}{\partial \alpha_i^2} \right|_{\alpha_i = \ln 1 - \epsilon_i / \epsilon_i} = 2\sqrt{\epsilon_i(1 - \epsilon_i)} > 0$$

confirmamos que se trata de un mínimo.

Adaboost & Overfitting



- Recordemos que la diferencia entre el error de predicción de un learner y el error de entrenamiento depende esencialmente de la “complejidad” del espacio de hipótesis que implemente. Una forma clásica de medir esta complejidad es la **dimensión VC**.

$$R(f) \leq \hat{R}_{\text{emp}}(f) + \sqrt{\frac{c \log(\frac{2n}{c} + 1) - \log(\frac{\eta}{4})}{n}}$$

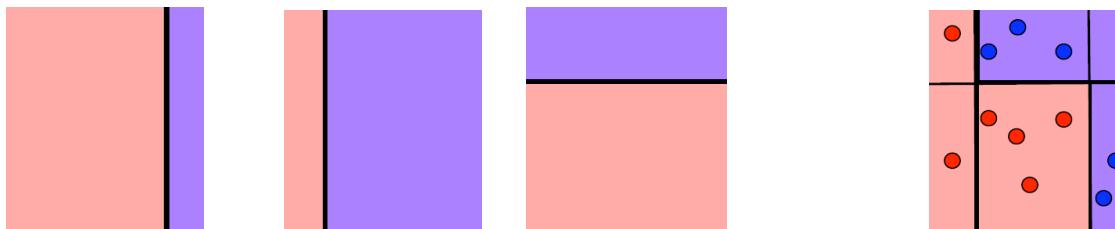
Error “de pruebas”
(riesgo)

Error de
entrenamiento

Complejidad

Adaboost & Overfitting

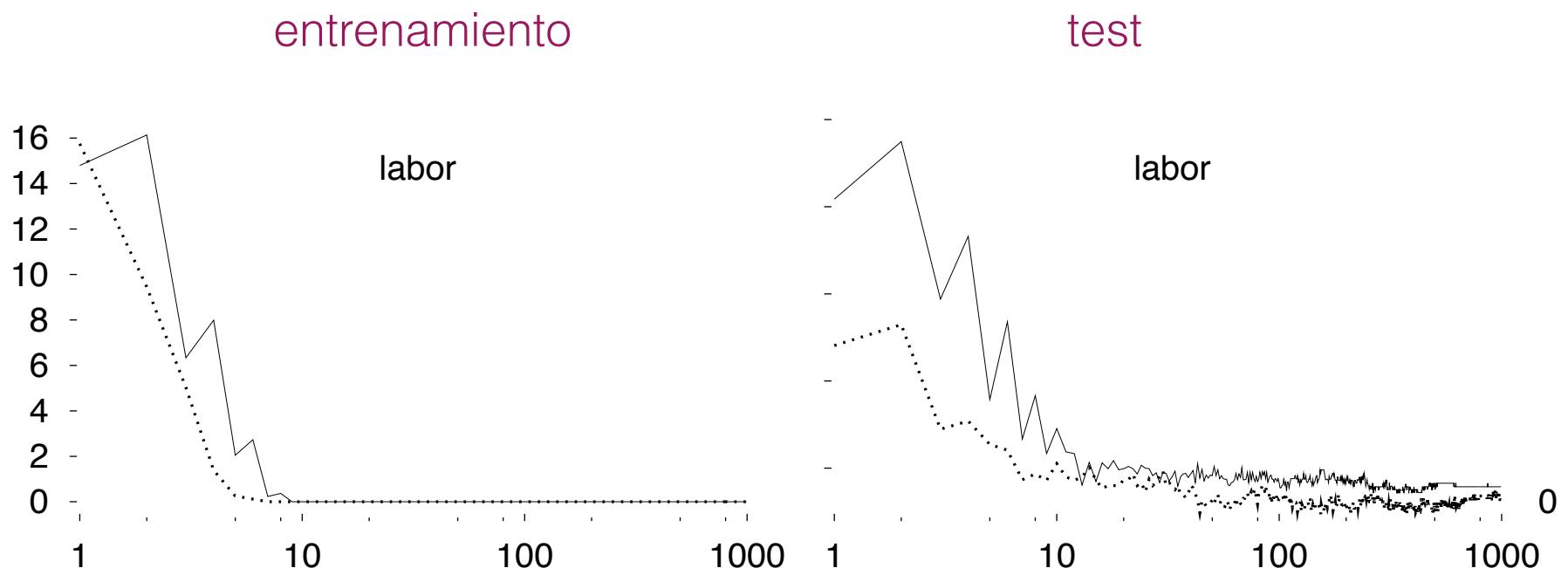
- Lamentablemente, se puede mostrar que (en general) la dimensión VC aumenta con el número de hipótesis combinadas por Adaboost, en el peor caso linealmente.



- Se minimiza el error de entrenamiento y se aumenta la complejidad, peligrosa combinación!

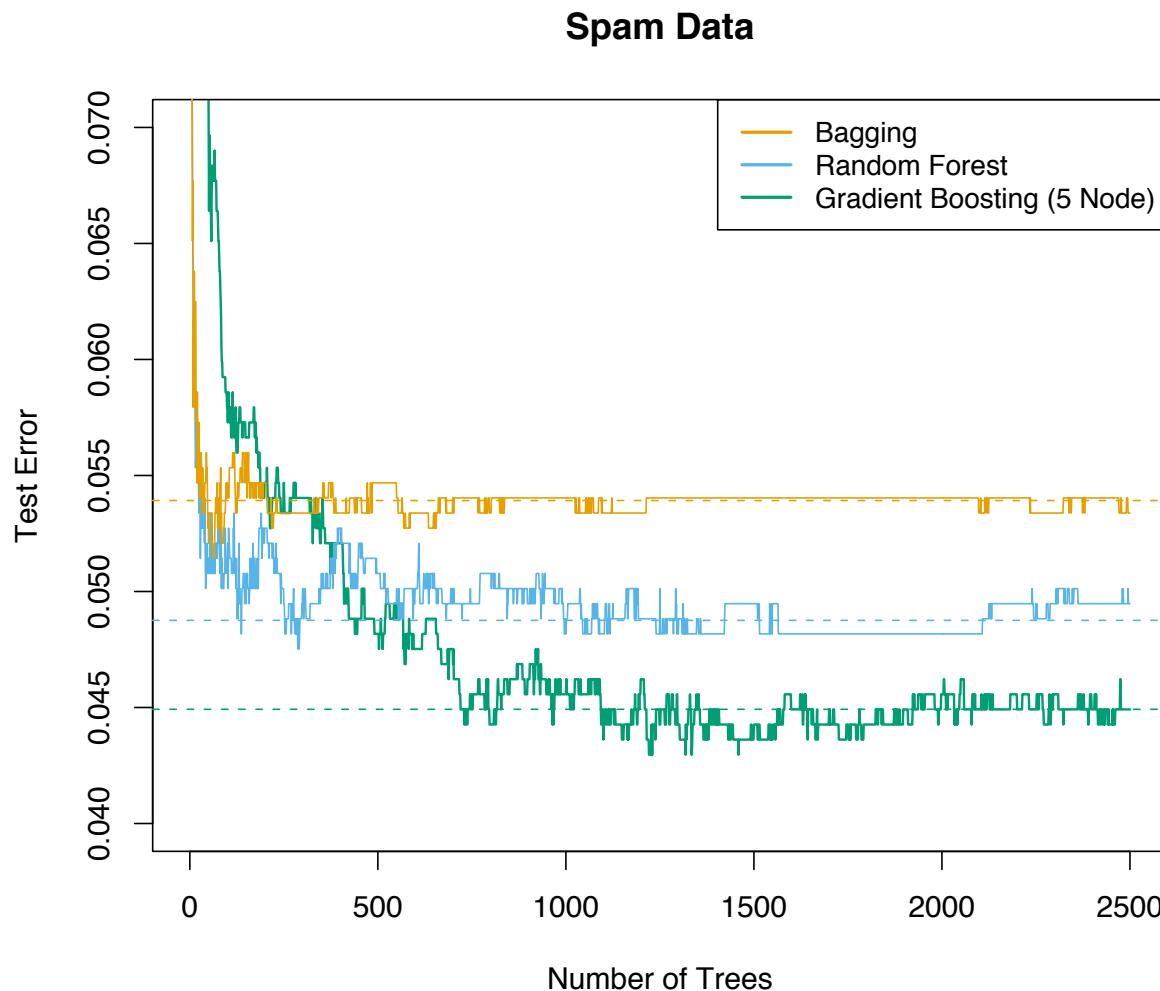
Adaboost & Overfitting

- En general, la evidencia experimental muestra que con mucha frecuencia esto no ocurre.



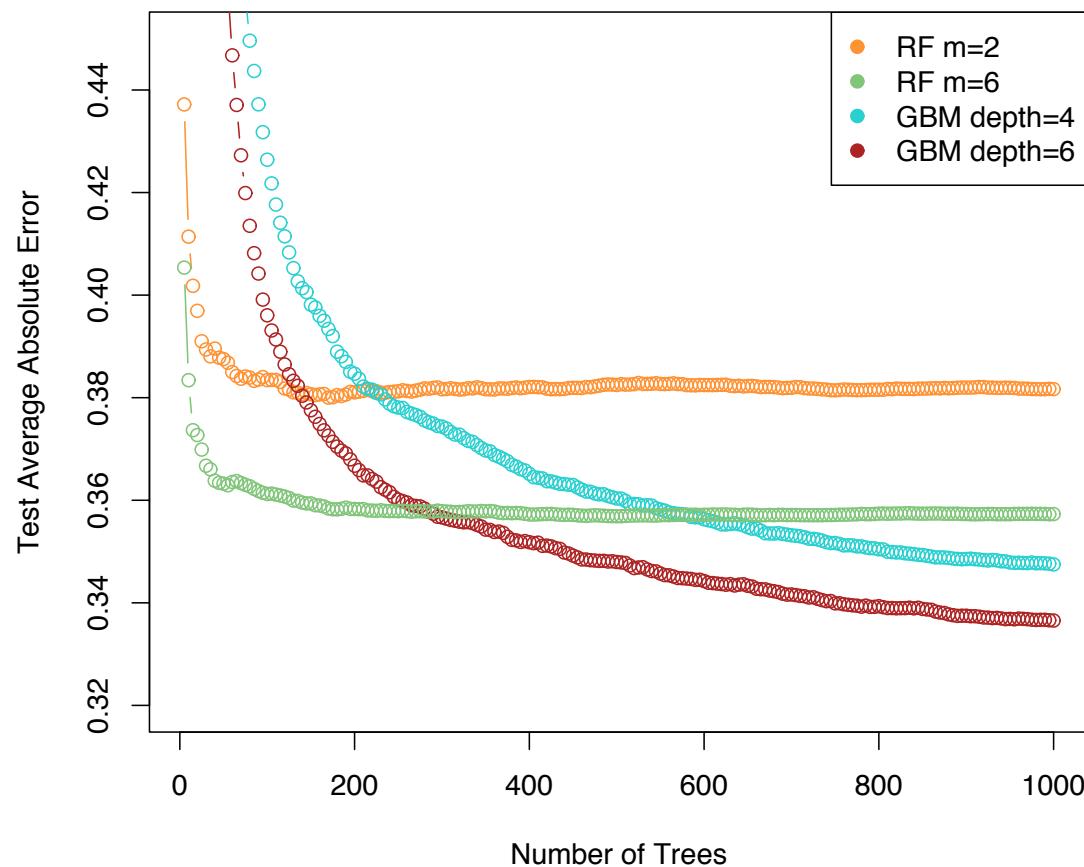
Adaboost & Overfitting

- En general, la evidencia experimental muestra que con mucha frecuencia esto no ocurre.



Adaboost & Overfitting

- En general, la evidencia experimental muestra que con mucha frecuencia esto no ocurre.



Adaboost & Margen

Teorema (Freund et al., 1998)

Sean f_1, f_2, \dots, f_t las hipótesis generadas por Adaboost. Para el ensamblado

$$F_t(x) = \frac{\sum_i \alpha_i f_i(x)}{\sum_i \alpha_i}$$

tenemos que $\forall \theta$

$$\begin{aligned} P_S(y F_t(x) \leq \theta) &= \frac{1}{n} \sum_j I(y^{(j)} F_t(x^{(j)}) \leq \theta) \\ &\leq 2^t \prod_{i=1}^t \sqrt{\epsilon_i^{1-\theta} (1 - \epsilon_i)^{1+\theta}} \end{aligned}$$

Adaboost & Margen

- Adaboost “mejora” en cada iteración la probabilidad de encontrar márgenes grandes entre los ejemplos de entrenamiento. Esto aumenta la probabilidad de encontrar márgenes blandos amplios y de consecuencia la capacidad de generalización de la máquina.

Teorema (Freund et al., 1998)

Si denotamos por d la dimensión VC del espacio del cual se extraen las hipótesis individuales, D es la distribución real de los datos, y $n>d$,

$$\begin{aligned} P_D(yF_t(x) \leq \theta) &\leq P_S(yF_t(x) \leq \theta) \\ &+ \mathcal{O}\left(\frac{1}{\sqrt{n}} \left(\frac{d \log^2(n/d)}{\theta^2} + \log(1/\delta) \right)\right) \end{aligned}$$

Hipótesis Continuas

- Muchos clasificadores (e.g. SVMs) son capaces no sólo de entregar una clasificación frente a un patrón de entrada, sino un **valor continuo** que sea puede interpretar como la “confianza” con la que el clasificador toma una determinada decisión. ¿Es posible usar hipótesis continuas en vez de discretas en Adaboost?

$$f_i : \mathbb{X} \rightarrow \mathbb{R}$$

- Consideremos entonces una versión un poco más general de Adaboost.

Optimalidad de los Pesos

- Este análisis revela, de paso, que la elección de los pesos de cada hipótesis es óptima para minimizar la cota del error de clasificación del ensamblado. En efecto, si consideramos

$$Z_i = \exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i$$

y optimizamos en α_i

$$\begin{aligned}\frac{\partial Z_i}{\partial \alpha_i} &= -\exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i = 0 \\ \Rightarrow -\alpha_i + \ln(1 - \epsilon_i) &= \alpha_i + \ln \epsilon_i \\ \Rightarrow 2\alpha_i &= \ln(1 - \epsilon_i) - \ln \epsilon_i = 0 \\ \Rightarrow \alpha_i &= \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right)\end{aligned}$$

Optimalidad de los Pesos

- La segunda derivada de Z_i es:

$$\frac{\partial^2 Z_i}{\partial \alpha_i^2} = \exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i$$

- Evaluando en el punto crítico:

$$\left. \frac{\partial^2 Z_i}{\partial \alpha_i^2} \right|_{\alpha_i = \ln 1 - \epsilon_i / \epsilon_i} = 2\sqrt{\epsilon_i(1 - \epsilon_i)} > 0$$

confirmamos que se trata de un mínimo.

Hipótesis Continuas

Algorithm 1: Adaboost Generalizado (2 clases).

Input : Ejemplos $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, $y^{(i)} \in \{\pm 1\}$, $N \in \mathbb{N}$.

Output: Ensemble $F(x)$.

- 1 Inicializar la distribución D sobre S como $D_1(j) = 1/n, \forall j$.
 - 2 **for** $i = 1, \dots, N$ **do**
 - 3 Generar un conjunto de ejemplos S_i^* remuestreando S con pesos $D_i(j)$.
 - 4 Entrenar un **clasificador** $f_i : \mathbb{X} \rightarrow [-1, 1]$ con S_i^* .
 - 5 Determinar el peso α_i de f_i .
 - 6 Actualizar la distribución D como
$$D_{i+1}(j) = \frac{D_i(j) \exp(\alpha_i y^{(j)} f_i(x^{(j)}))}{Z_i} \quad (1)$$
con $Z_i = \sum_j \exp(\alpha_i y^{(j)} f_i(x^{(j)}))$.
 - 7 Devolver $F(x) = \text{sign} (\sum_i \alpha_i f_i(x))$.
-

Hipótesis Continuas

- La cota para el error de clasificación en función de Z_i sigue siendo perfectamente válida:

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t Z_i$$

- Para minimizar la cota sobre el error debemos minimizar Z_i

$$Z_i = \sum_j D_{i+1}(j) = \sum_j D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)}))$$

Hipótesis Continuas

- La cota para el error de clasificación en función de Z_i sigue siendo perfectamente válida:

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t Z_i$$

con

$$Z_i = \sum_j D_{i+1}(j) = \sum_j D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)}))$$

- Para la versión discreta,

$$Z_i = \exp(-\alpha_i)(1 - \epsilon_i) + \exp(\alpha_i)\epsilon_i$$

- Esto no es cierto si las hipótesis son continuas. La diferencia fundamental con la versión discreta, es entonces que los pesos de cada hipótesis ya no son óptimos.

Hipótesis Continuas

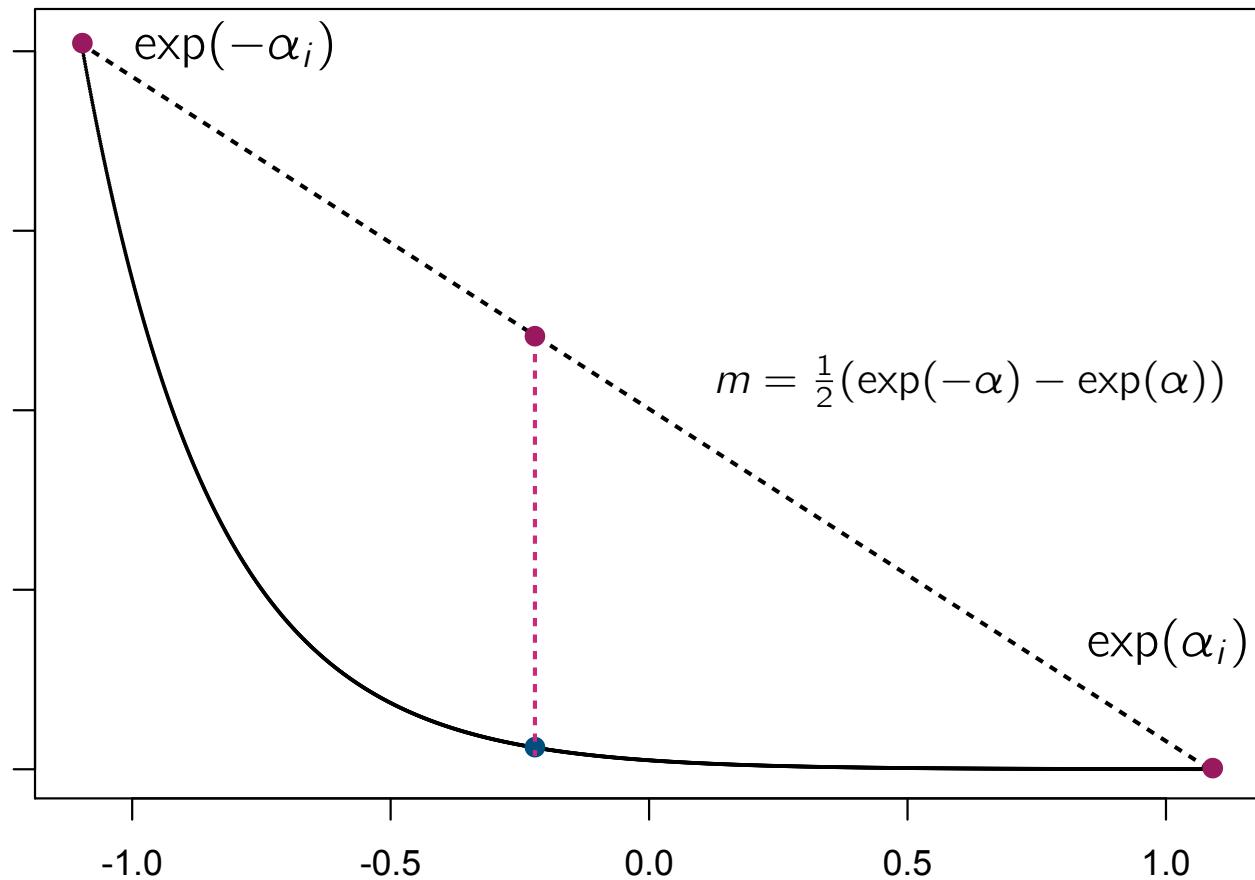
$$Z_i = \sum_j D_{i+1}(j) = \sum_j D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)}))$$

- Una posibilidad es elegir α_i minimizando numéricamente esta expresión. Otra opción (más común) consiste en encontrar una cota superior que se pueda minimizar analíticamente.
- En el caso discreto $y^{(j)} f_i(x^{(j)}) \in \{\pm 1\}$, de modo que

$$\exp(-\alpha_i y^{(j)} f_i(x^{(j)})) = \begin{cases} \exp(-\alpha_i) & \text{si } y^{(j)} = f_i(x^{(j)}) \\ \exp(+\alpha_i) & \text{si } y^{(j)} \neq f_i(x^{(j)}) \end{cases}$$

- En el caso continuo, estos son extremos una función convexa.

Hipótesis Continuas



Hipótesis Continuas

- Obtenemos que

$$\begin{aligned} Z_i &= \sum_j D_i(j) \exp(-\alpha_i y^{(j)} f_i(x^{(j)})) \\ &\leq \sum_j D_i(j) \left(\frac{1 + y^{(j)} f_i(x^{(j)})}{2} \exp(-\alpha_i) + \frac{1 - y^{(j)} f_i(x^{(j)})}{2} \exp(\alpha_i) \right) \end{aligned}$$

(para la versión discreta esta es una identidad)

- Optimizando en α_i

$$\begin{aligned} \frac{\partial}{\partial \alpha_i} \sum_j D_i(j) \left(\frac{1 + y^{(j)} f_i(x^{(j)})}{2} \exp(-\alpha_i) + \frac{1 - y^{(j)} f_i(x^{(j)})}{2} \exp(\alpha_i) \right) \\ = -\exp(-\alpha_i) \frac{1 + \sum_j D_i(j) y^{(j)} f_i(x^{(j)})}{2} + \exp(\alpha_i) \frac{1 - \sum_j D_i(j) y^{(j)} f_i(x^{(j)})}{2} \end{aligned}$$

Hipótesis Continuas

- Reacomodando

$$\begin{aligned}\frac{\partial \text{cota}}{\partial \alpha_i} &= \sum_j D_i(j) \left(-\frac{1 + y^{(j)} f_i(x^{(j)})}{2} \exp(-\alpha_i) + \frac{1 - y^{(j)} f_i(x^{(j)})}{2} \exp(\alpha_i) \right) \\ &= -\exp(-\alpha_i) \frac{1 + \sum_j D_i(j) y^{(j)} f_i(x^{(j)})}{2} + \exp(\alpha_i) \frac{1 - \sum_j D_i(j) y^{(j)} f_i(x^{(j)})}{2}\end{aligned}$$

- Por lo tanto (definiendo $r_i = \sum_j D_i(j) y^{(j)} f_i(x^{(j)})$) tenemos

$$\begin{aligned}\frac{\partial \text{cota}}{\partial \alpha_i} = 0 &\Rightarrow -\alpha_i + \ln \left(\frac{1 + r_i}{2} \right) = \alpha_i + \ln \left(\frac{1 - r_i}{2} \right) \\ &\Rightarrow 2\alpha_i = \ln \left(\frac{1 + r_i}{2} \right) - \ln \left(\frac{1 - r_i}{2} \right) \\ &\Rightarrow \alpha_i = \frac{1}{2} \ln \left(\frac{1 + r_i}{2} \right) - \ln \left(\frac{1 - r_i}{2} \right) \\ &\Rightarrow \alpha_i = \frac{1}{2} \ln \left(\frac{1 + r_i}{1 - r_i} \right)\end{aligned}$$

Hipótesis Continuas

- El término

$$r_i = \sum_j D_i(j) y^{(j)} f_i(x^{(j)})$$

- Se denomina el edge del clasificador i -ésimo y representa el valor esperado de los márgenes obtenidos sobre el conjunto de entrenamiento.
- De aquí y recordando que

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t Z_i$$

es fácil probar el siguiente teorema.

Garantías

Teorema (Schapire & Singer, 1999)

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t \sqrt{(1+r_i)(1-r_i)} = \prod_{i=1}^t \sqrt{(1-r_i^2)}$$

- Además, la cota sobre el error de generalización del caso discreto sigue siendo válida (con pequeñas diferencias técnicas)

$$\begin{aligned} P_D(yF_t(x) \leq \theta) &\leq P_S(yF_t(x) \leq \theta) \\ &\quad + \mathcal{O}\left(\frac{1}{\sqrt{n}} \left(\frac{d \log^2(n/d)}{\theta^2} + \log(1/\delta) \right)\right) \end{aligned}$$

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3), 297-336.

Hipótesis Continuas

- Comparando con la versión discreta tenemos

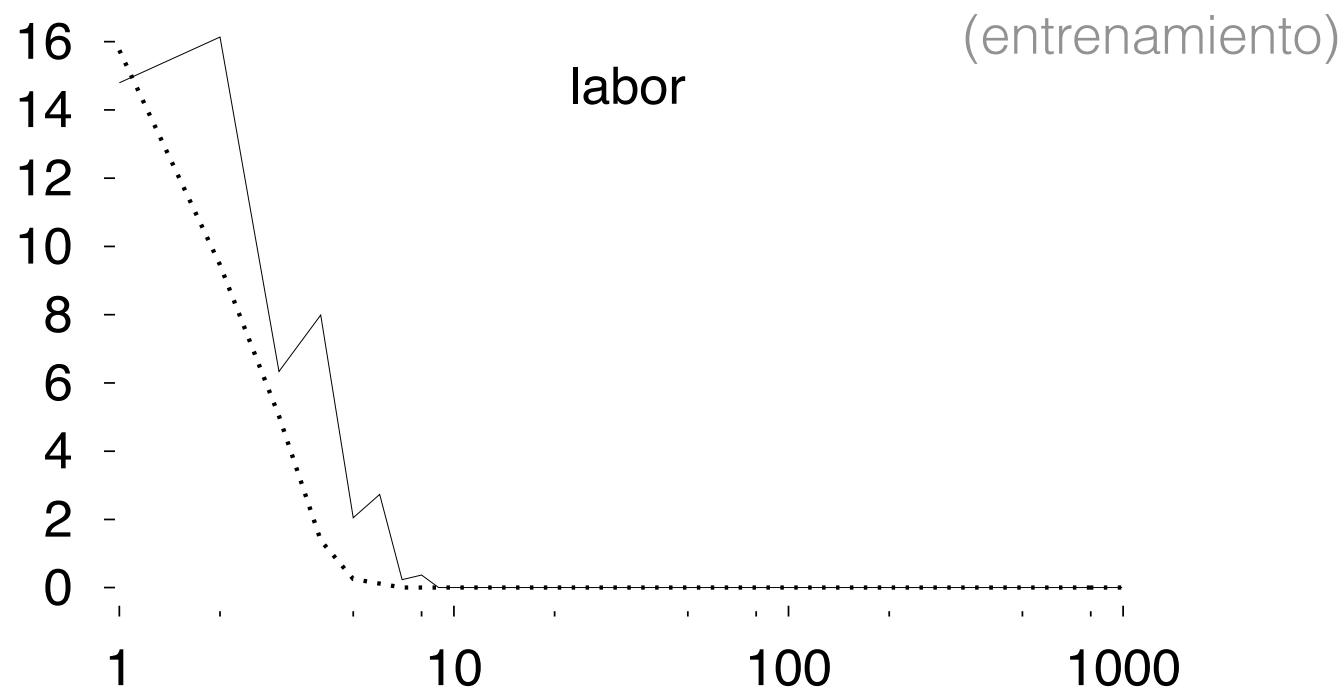
$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t \sqrt{(1 - 4\gamma_i^2)} \quad \gamma_i = 1/2 - \epsilon_i$$

$$\frac{1}{n} \sum_j I(y^{(j)} \neq F_t(x^{(j)})) \leq \prod_{i=1}^t \sqrt{(1 - r_i^2)}$$

- En general es “más fácil” mejorar el edge (margen promedio) que mejorar la probabilidad de clasificaciones correctas con respecto a la distribución de cada iteración.
- Esto sugiere que la versión continua de Adaboost puede ser más efectiva minimizando el error de entrenamiento, con cotas similares sobre el error de generalización.

Hipótesis Continuas

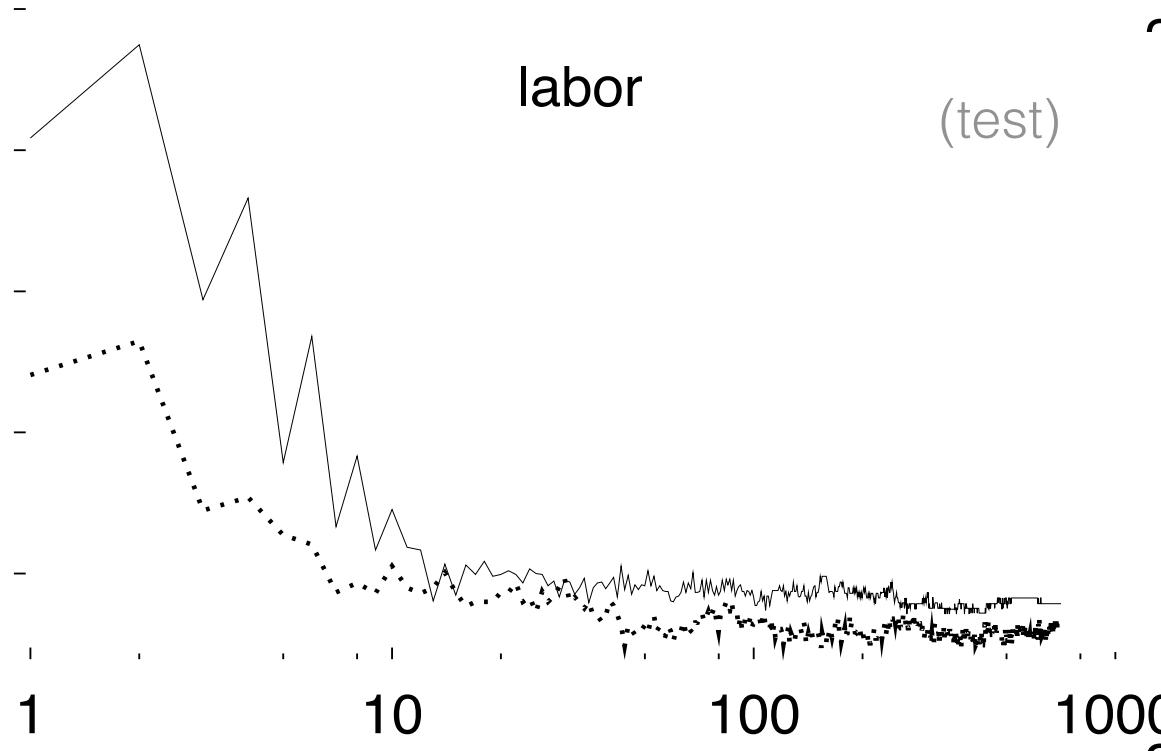
- Continuo (línea discreta) versus discreto (línea continua)



Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3), 297-336.

Hipótesis Continuas

- Continuo (línea discreta) versus discreto (línea continua)



Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3), 297-336.

AdaBoost Multi-categorías

- Es relativamente fácil extender Adaboost al caso de múltiples clases. Sin embargo varios trabajos han reportado que en este caso se obtienen rendimientos mucho más pobres que en el caso binario.

4 Entrenar un clasificador $f_i : \mathbb{X} \rightarrow \{1, 2, \dots, K\}$ con S_i^* .

5 Determinar la probabilidad de error sobre S_i^* ,

$$\epsilon_i = \sum_j D_i(j) I(y^{(j)} \neq f_i(x^{(j)})).$$

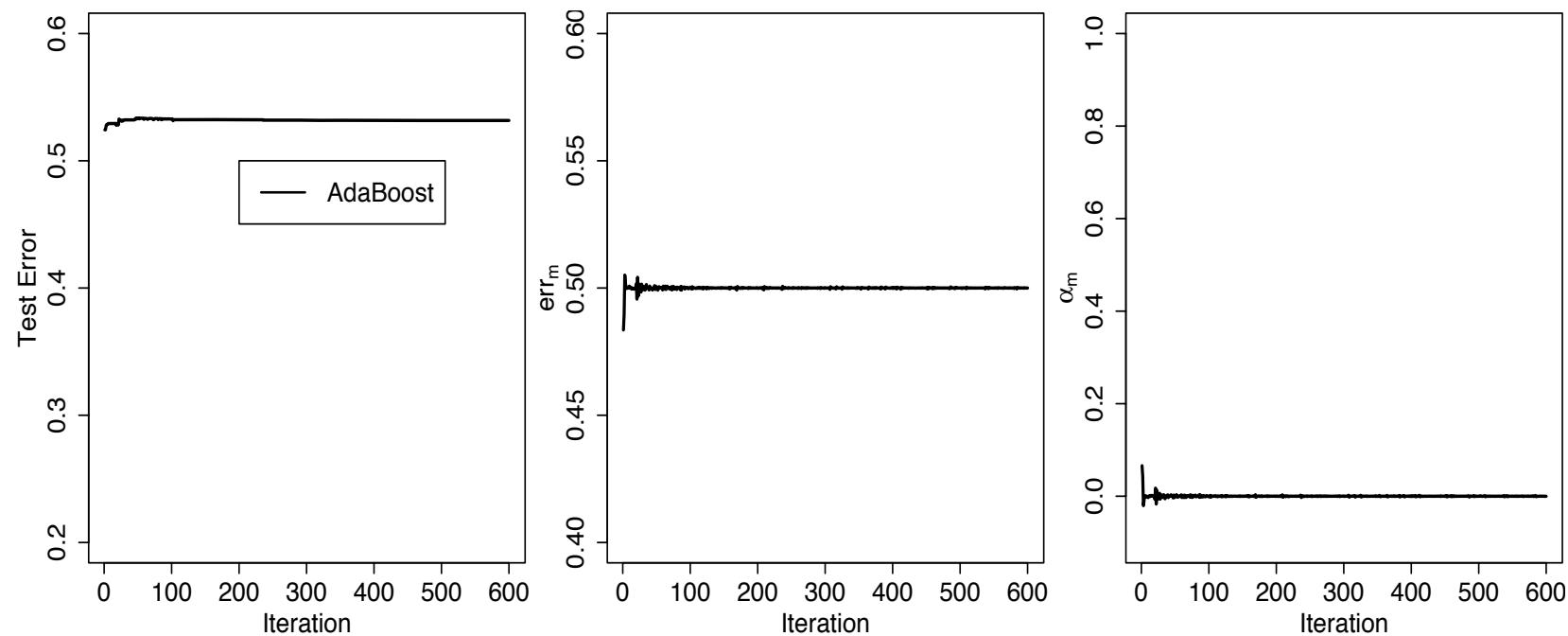
6 Determinar el peso de f_i como $\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right)$.

7 Actualizar la distribución D como

$$D_{i+1}(j) = \frac{D_i(j) \exp(-\alpha_i I(y^{(j)} \neq f_i(x^{(j)})))}{Z_i} \quad (1)$$

AdaBoost Multi-categorías

- De acuerdo a (Hastie et al. 2009), el problema se produce porque en este tipo de problemas, obtener un error un poco menor que 0.5 es mucho más difícil que en el caso binario. Como consecuencia, los pesos de las hipótesis son fácilmente negativos o prácticamente cero.



AdaBoost Multi-categorías

- Hastie et al. 2009 proponen la siguiente modificación sobre la forma en que calculan los pesos de cada hipótesis

4 Entrenar un clasificador $f_i : \mathbb{X} \rightarrow \{1, 2, \dots, K\}$ con S_i^* .

5 Determinar la probabilidad de error sobre S_i^* ,

$$\epsilon_i = \sum_j D_i(j) I(y^{(j)} \neq f_i(x^{(j)})).$$

6 Determinar el peso de f_i como $\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right) + \log(K-1)$.

7 Actualizar la distribución D como

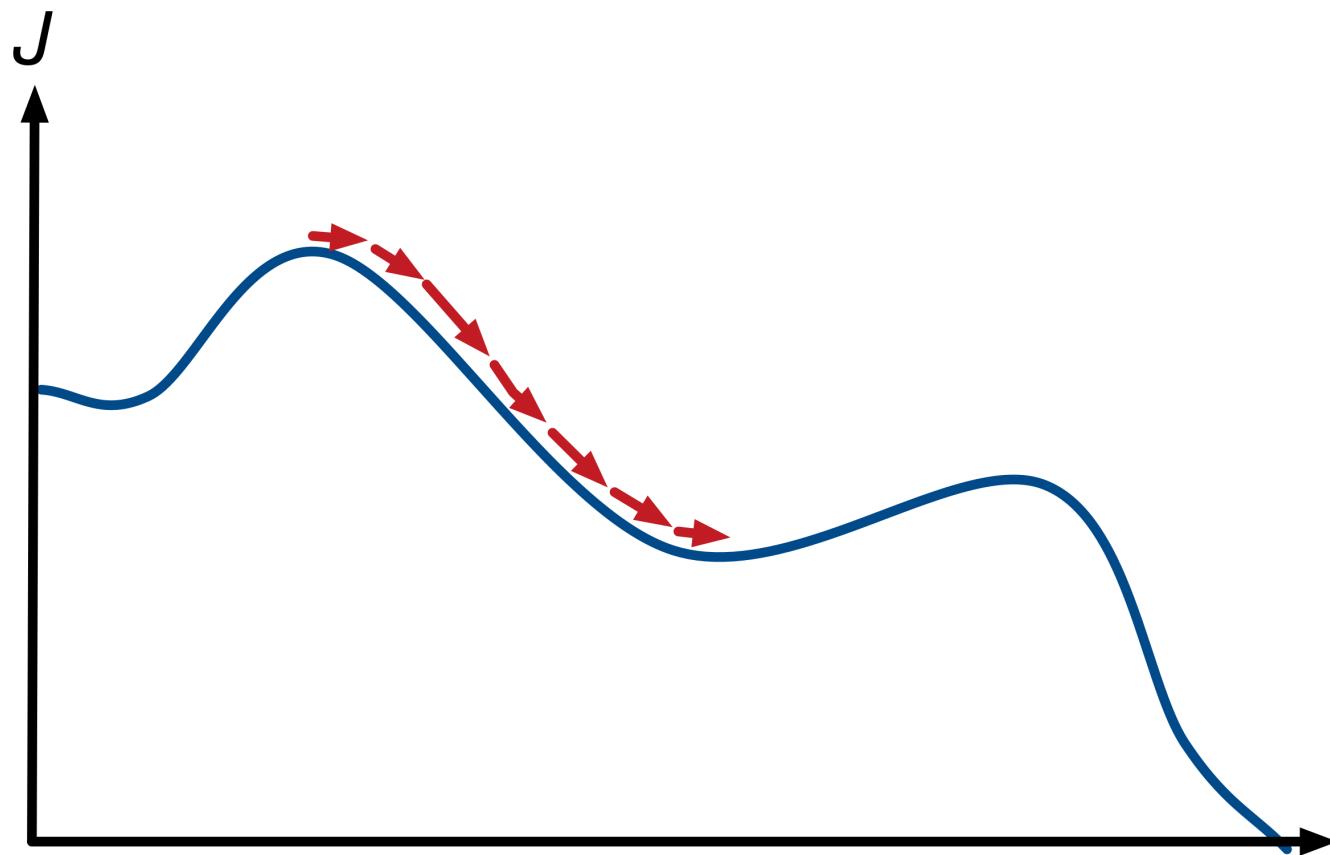
$$D_{i+1}(j) = \frac{D_i(j) \exp(-\alpha_i I(y^{(j)} \neq f_i(x^{(j)})))}{Z_i} \quad (1)$$

- Con esta modificación, el peso de una hipótesis es negativo sólo si ésta es peor que un clasificador aleatorio (error mayor a $1/K$).
- La justificación teórica es algo extensa y se basa en intentos anteriores que buscaban generalizar el concepto de margen al caso multi-categoría que permitieran el entrenamiento de SVMs.

Boosting versus Bagging (o RF)

- La principal motivación para Bagging es la posibilidad de reducir el error de predicción reduciendo la varianza. El sesgo no se reduce activamente. Todo esto hace que Bagging se use casi siempre con base learners no regularizados (o poco regularizados). El control de overfitting se produce al ensamblar.
- Boosting actúa más explícita y activamente sobre el sesgo. El error de entrenamiento puede reducirse “tanto como queramos” agregando predictores. La reducción de la varianza, si ocurre, es implícita, y viene de la complementariedad de los predictores. Esto hace que en la práctica, la posibilidad de overfitting sí sea una preocupación al usar Boosting y que con frecuencia se use con base learners regularizados (e.g. árboles de clasificación de 1 nivel).

Gradient Boosting

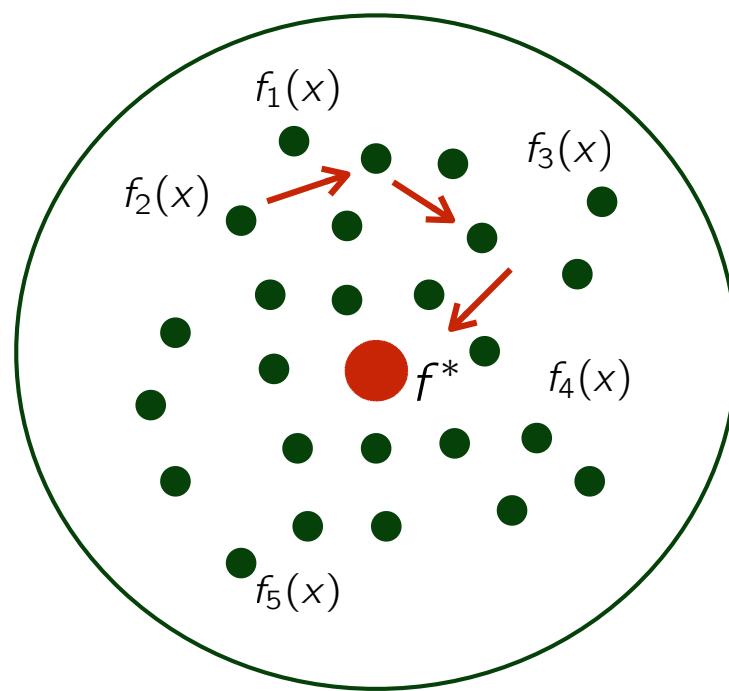


Boosting como Grad. Descendente

- Como sabemos, un problema de aprendizaje puede entenderse como un **problema de búsqueda** en un espacio de funciones que hemos denominado “espacio de hipótesis”. Esa búsqueda intenta maximizar un objetivo, por ejemplo:

$$J(f) = \frac{1}{n} \sum_{\ell=1}^n L(y^{(\ell)}, f(x^{(\ell)}))$$

- Para modelos anteriores, hemos abordado esta optimización parametrizando el espacio de funciones y ejecutando GD sobre el espacio de parámetros.



Boosting como Grad. Descendente

- Si el espacio de hipótesis estuviese equipado con un operador de diferenciación, podríamos abordar esta optimización usando gradiente descendente directamente en el espacio de funciones.

$$F_0 = 0$$

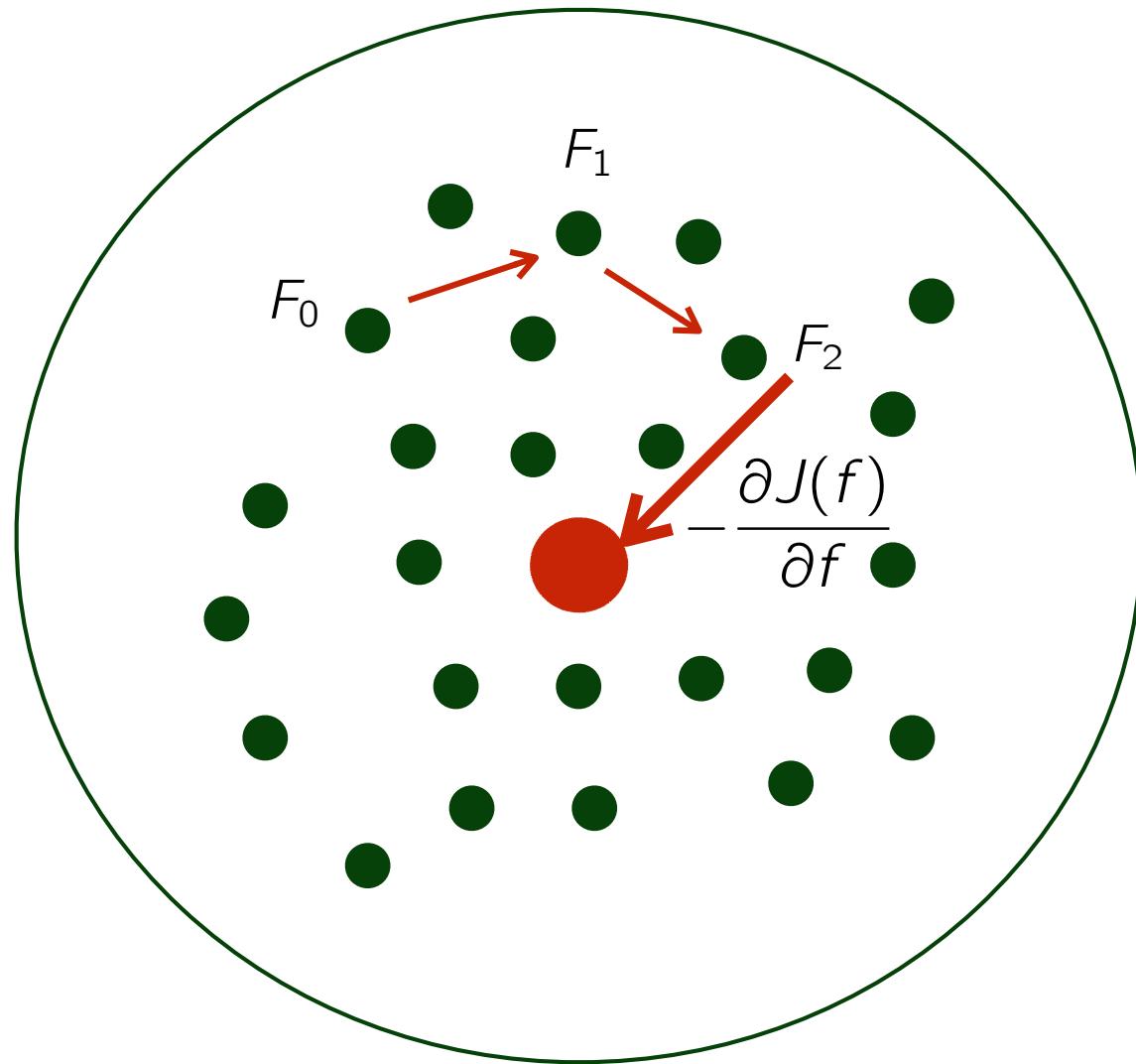
$$F_1 = F_0 - \alpha_1 \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_0} = -\alpha_1 \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_0}$$

$$F_2 = F_1 - \alpha_2 \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_1} = -\alpha_1 \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_0} - \alpha_2 \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_1}$$

.....

$$F_t = F_{t-1} - \alpha_t \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_{t-1}} = - \sum_i \alpha_i \left. \frac{\partial J(f)}{\partial f} \right|_{f=F_{i-1}}$$

Boosting como Grad. Descendente



Boosting como Grad. Descendente

- Afortunadamente, es posible equipar el espacio de funciones con un operador de diferenciación:

$$\frac{\partial J(f)}{\partial f}(x) = \left. \frac{\partial C(f + \alpha 1_x)}{\partial \alpha} \right|_{\alpha=0}$$

- Si la función objetivo tiene la forma

$$J(f) = \frac{1}{n} \sum_{\ell=1}^n L(y^{(\ell)} f(x^{(\ell)}))$$

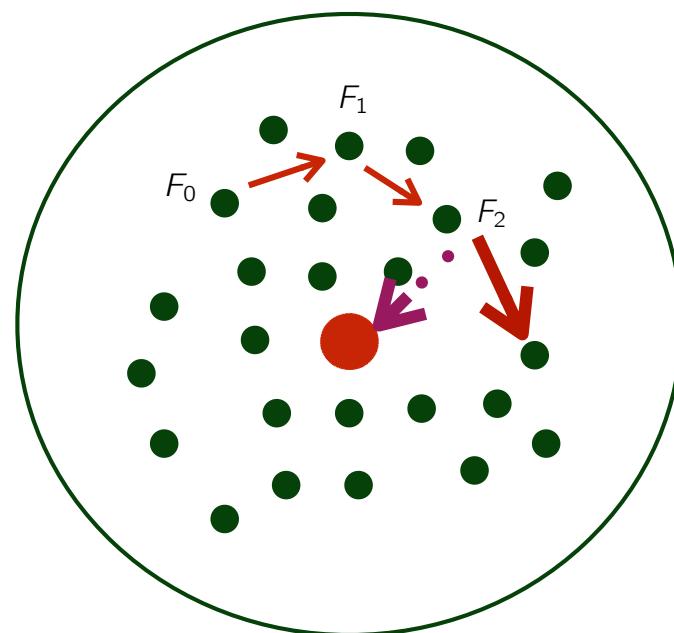
obtenemos que

$$\frac{\partial J(f)}{\partial f}(x) = \begin{cases} \frac{1}{n} y^{(\ell)} L'(y^{(\ell)} f(x^{(\ell)})) & x = x^{(\ell)} \\ 0 & \text{en otro caso} \end{cases}$$

Boosting como Grad. Descendente

$$\text{?} - \frac{\partial J(f)}{\partial f} \Big|_{f=F_{i-1}} \in \mathcal{H} \text{ ?}$$

- En la práctica, nuestro espacio de hipótesis es limitado y podría no contener esta función deseada.
- Una posible solución es entrenar el leaner base en la iteración i para encontrar una función que apunte aproximadamente en esa dirección.



- Para saber que tan buena es la aproximación, necesitamos una métrica. Una buena idea es equipar nuestro espacio con un producto punto que nos permita medir “el ángulo” entre ambas direcciones.

Boosting como Grad. Descendente

- Una elección posible es

$$\langle f, g \rangle = \frac{1}{n} \sum_{\ell} f(x^{(\ell)})g(x^{(\ell)})$$

- Para nuestra definición de gradiente, obtenemos que, en cada iteración, el learner base f_i debiese ser entrenado para maximizar

$$\left\langle f_i, -\frac{\partial J(f)}{\partial f} \Big|_{f=F_{i-1}} \right\rangle = -\frac{1}{n^2} \sum_{\ell} L'(y^{(\ell)} F_{i-1}(x^{(\ell)})) y^{(\ell)} f(x^{(\ell)})$$

o para minimizar

$$J_i(f) = \frac{1}{n^2} \sum_{\ell} L'(y^{(\ell)} F_{i-1}(x^{(\ell)})) y^{(\ell)} f(x^{(\ell)})$$

AnyBoost

- Obtenemos así un Boosting “genérico”, que funciona para cualquier función de pérdida (loss) que deseemos emplear.

Algorithm 1: AnyBoost

- 1 Inicializar $F_0 = 0$.
- 2 **for** $i = 1, \dots, N$ **do**
- 3 Entrenar f_i para minimizar

$$J_i(f) = \frac{1}{n^2} \sum_{\ell} L'(y^{(\ell)} F_{i-1}(x^{(\ell)})) y^{(\ell)} f(x^{(\ell)}) \quad (1)$$

Determinar un valor para la “tasa de aprendizaje” α_i . Por ejemplo, definir $F_\alpha = F_{i-1} + \alpha f_i$ y optimizar (quizás de modo aproximado)

$$J_i(\alpha) = \frac{1}{n} \sum_{\ell} L(y^{(\ell)} F_\alpha(x^{(\ell)})) \quad (2)$$

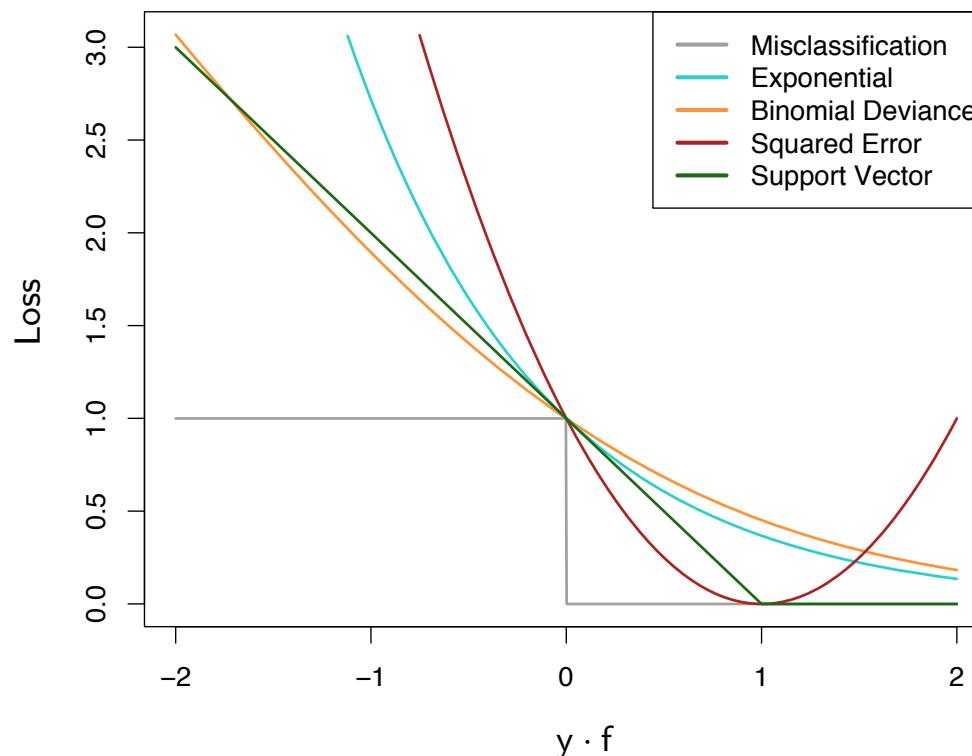
$$F_i(x) = \sum_i \alpha_i f_i(x)$$

Adaboost desde AnyBoost

- La versión clásica de Adaboost se obtiene como caso especial de este algoritmo genérico al usar la función de pérdida

$$L(y^{(\ell)} f(x^{(\ell)})) = \exp(-y^{(\ell)} f(x^{(\ell)}))$$

denominada **exponential loss**.



Adaboost desde AnyBoost

- Para este caso, la función objetivo de cada iteración queda

$$\begin{aligned} & \min_f \frac{1}{n^2} \sum_{\ell} -\exp(-y^{(\ell)} F_{i-1}(x^{(\ell)})) y^{(\ell)} f(x^{(\ell)}) \\ \Leftrightarrow & \max_f \frac{1}{n^2} \sum_{\ell} \exp(-y^{(\ell)} F_{i-1}(x^{(\ell)})) y^{(\ell)} f(x^{(\ell)}) \\ \Leftrightarrow & \max_f \sum_{\ell} D_i(\ell) y^{(\ell)} f(x^{(\ell)}) \end{aligned}$$

con

$$\begin{aligned} D_i(\ell) &\propto \exp(-y^{(\ell)} F_{i-1}(x^{(\ell)})) \\ D_{i+1}(\ell) &\propto D_i(\ell) \exp(-\alpha_i y^{(\ell)} f_i(x^{(\ell)})) \end{aligned}$$

Adaboost desde AnyBoost

- Para el caso en que las hipótesis son discretas (+1,-1)

$$\begin{aligned}\sum_{\ell} D_i(\ell) y^{(\ell)} f(x^{(\ell)}) &= \sum_{y^{(\ell)} \neq f(x^{(\ell)})} -D_i(\ell) + \sum_{y^{(\ell)} = f(x^{(\ell)})} D_i(\ell) \\ &= \sum_{y^{(\ell)} \neq f(x^{(\ell)})} -D_i(\ell) + \left(1 - \sum_{y^{(\ell)} \neq f(x^{(\ell)})} D_i(\ell)\right)\end{aligned}$$

por lo tanto

$$\begin{aligned}\sum_{\ell} D_i(\ell) y^{(\ell)} f(x^{(\ell)}) &= -2 \sum_{y^{(\ell)} \neq f(x^{(\ell)})} D_i(\ell) + 1 \\ &= -2 \sum_{\ell} D_i(\ell) I(y^{(\ell)} \neq f(x^{(\ell)})) + 1\end{aligned}$$

por lo tanto

$$\max_f \sum_{\ell} D_i(\ell) y^{(\ell)} f(x^{(\ell)}) \Leftrightarrow \min_f \sum_{\ell} D_i(\ell) I(y^{(\ell)} \neq f(x^{(\ell)}))$$

Re-sampling versus Re-weighting

- Este análisis muestra que el objetivo del i-ésimo learner es efectivamente minimizar la probabilidad de clasificar incorrectamente un dato con respecto a la distribución de esa ronda.

$$\sum_{\ell} D_i(\ell) I(y^{(\ell)} \neq f_i(x^{(\ell)})) = P_{D_i}(y^{(\ell)} \neq f_i(x^{(\ell)}))$$

- Este objetivo puede implementarse de modo directo, es decir, modificando la función objetivo del learner base para que asigne mayor peso a algunos ejemplos.
- Si denotamos por (x^*, y^*) un par obtenido vía re-muestreo con una cierta distribución de probabilidad D , tenemos que

$$\mathbb{E}(I(y^* \neq f_i(x^*))) = \sum_{\ell} D(\ell) I(y^{(\ell)} \neq f_i(x^{(\ell)}))$$

Adaboost desde AnyBoost

- Como ya se había demostrado, los pesos de Adaboost discreto son óptimos para la función objetivo

$$\sum_{\ell} D_i(\ell) \exp(-\alpha_i y^{(\ell)} f_i(x^{(\ell)}))$$

- Y por lo tanto son óptimos para

$$\sum_{\ell} \exp(-\alpha_i y^{(\ell)} F_i(x^{(\ell)}))$$

que es simplemente el valor del objetivo después de agregar la última hipótesis aprendida.

Adaboost desde AnyBoost

- Es interesante notar que todo este análisis sugiere que en la versión continua de Adaboost, cada learner debiese entrenarse con el criterio

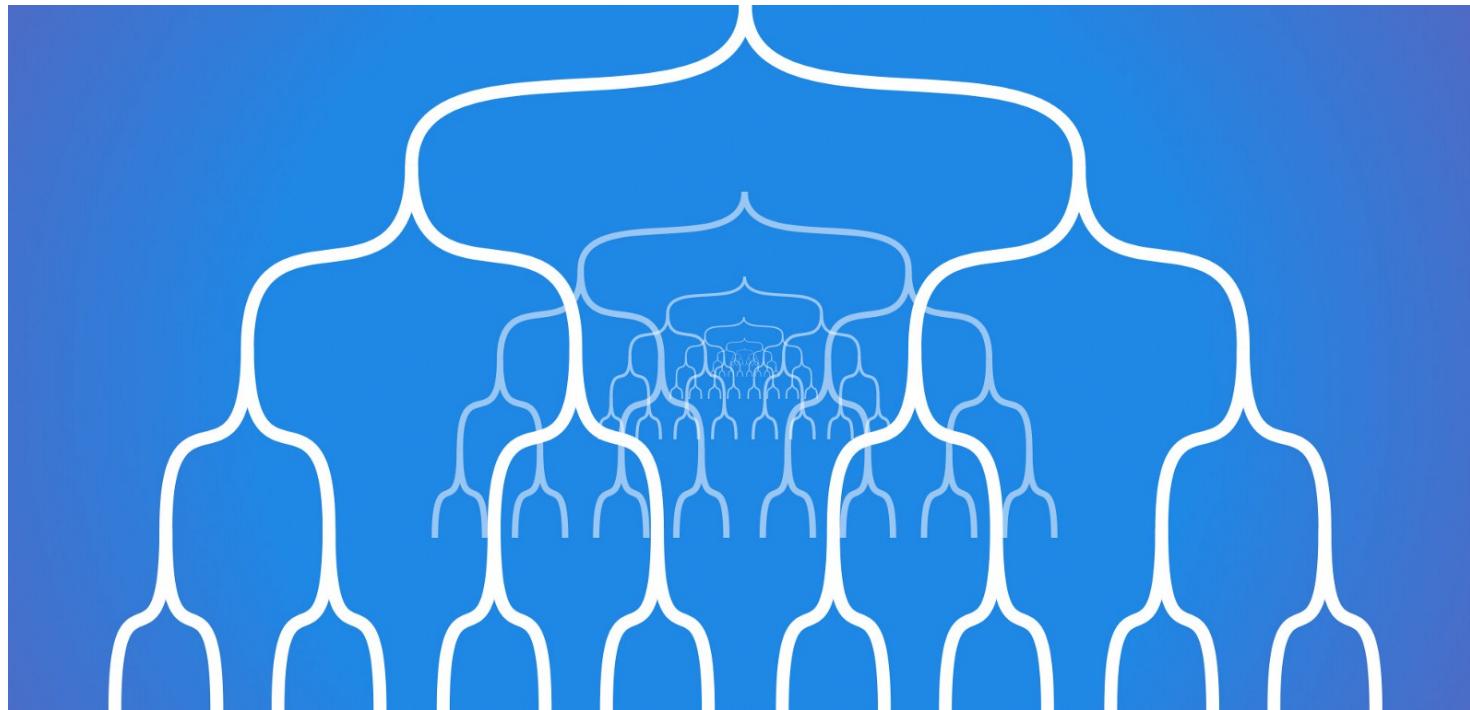
$$\max_f \sum_{\ell} D_i(\ell) y^{(\ell)} f(x^{(\ell)})$$

que en este caso no coincide con el criterio de Schapire & Singer

$$\min_f \sum_{\ell} D_i(\ell) I(y^{(\ell)} \neq f(x^{(\ell)}))$$

Gradient Tree Boosting

- En los últimos años, se ha popularizado una versión del método anterior especializada en árboles (XGBoost).



Gradient Tree Boosting

- Una de las diferencias acá es el modo en que se hace la aproximación del gradiente deseado en cada iteración.
- Primero, se considera una versión más general de la función de entrenamiento (no necesariamente función del margen)

$$J(f) = \sum_{\ell} L(y^{(\ell)}, f(x^{(\ell)}))$$

$$-\frac{\partial J(f)}{\partial f}(x) = \begin{cases} -\frac{\partial L(y^{(\ell)}, f(x^{(\ell)}))}{\partial f(x^{(\ell)})} & x = x^{(\ell)} \\ 0 & \text{en otro caso} \end{cases}$$

Gradient Tree Boosting

- En vez de aproximar de este gradiente maximizando el alineamiento del predictor i -ésimo con esa función en cada iteración

$$\left\langle f_i, - \frac{\partial J(f)}{\partial f} \Big|_{f=F_{i-1}} \right\rangle$$

se define

$$r_i^{(\ell)} = - \frac{\partial J(f)}{\partial f} \Big|_{f=F_{i-1}} (x^{(\ell)}) = - \frac{\partial L(y^{(\ell)}, f(x^{(\ell)}))}{\partial f(x^{(\ell)})}$$

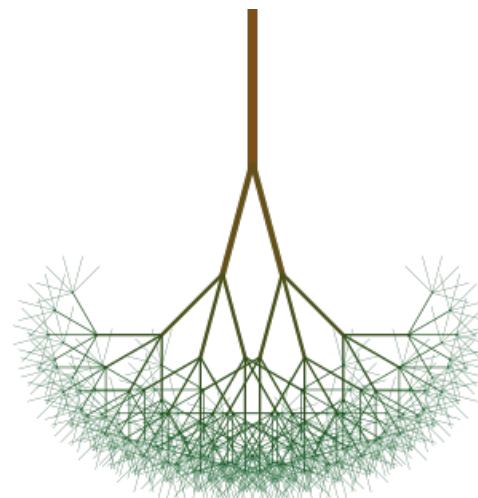
y se entrena el learner para resolver un problema de regresión con el conjunto de entrenamiento modificado:

$$\{(x^{(\ell)}, r_i^{(\ell)})\}_{\ell=1}^n$$

Gradient Tree Boosting

- Esto tiene la ventaja de que se puede utilizar un learner base genérico sin necesidad de “intervenir” su función objetivo. Típicamente se emplean árboles de regresión, que aprenderán a particionar el espacio para aproximar el gradiente deseado, usando como ejemplos los gradientes requeridos en los datos de entrenamiento.

$$\{(x^{(\ell)}, r_i^{(\ell)})\}_{\ell=1}^n$$



Gradient Tree Boosting

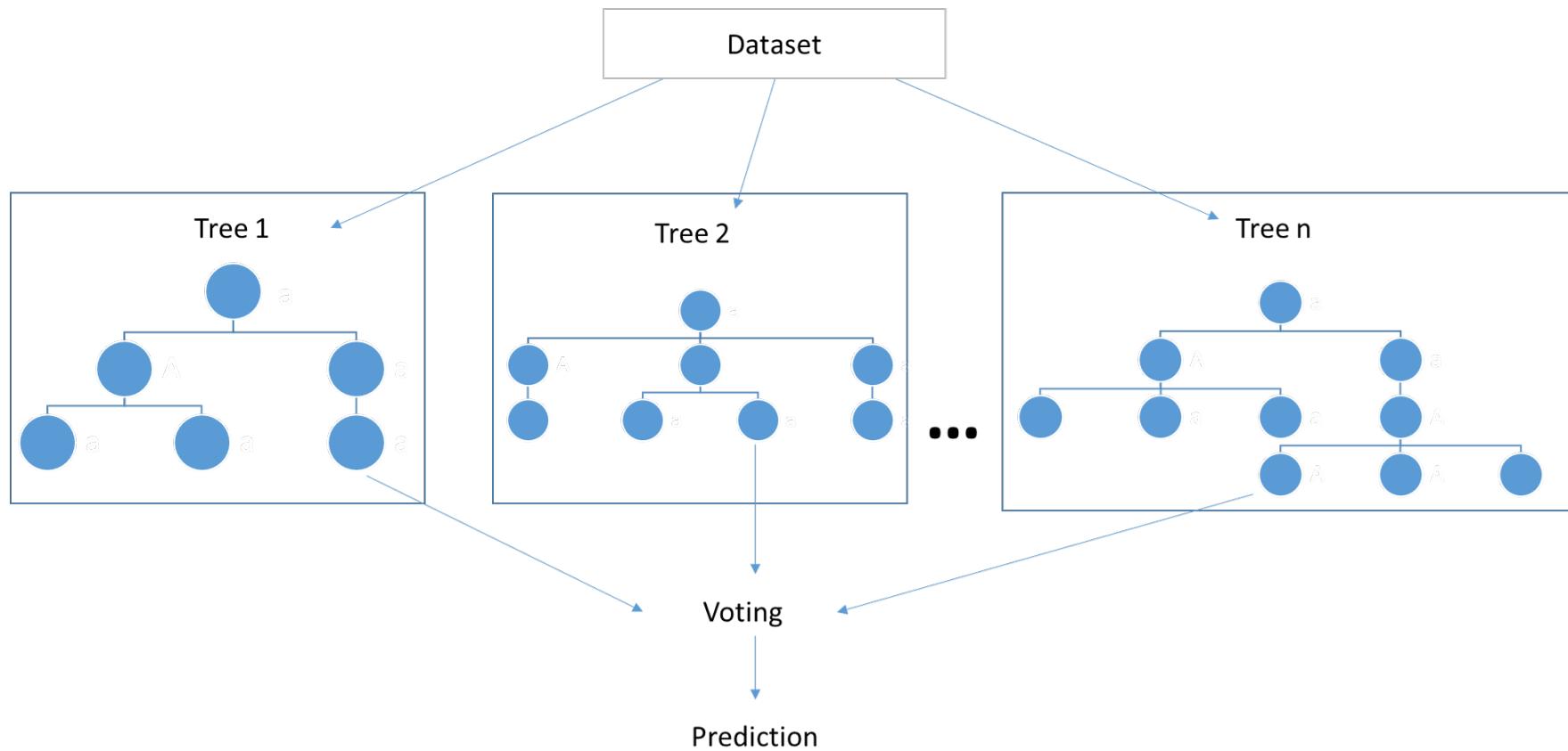
- El tipo de partición del espacio de características que induce un árbol, permite hacer una optimización adicional: una vez que se han inferido las hojas (regiones) del árbol i -ésimo $\{R_{im}\}$, la predicción γ_{im} correspondiente a cada una de ellas se reemplaza por la solución del siguiente problema

$$\gamma_{im} = \arg \min_{\gamma} \sum_{\ell} L(y^{(\ell)}, F_{i-1}(x^{(\ell)}) + \gamma) I(x^{(\ell)} \in R_{im})$$

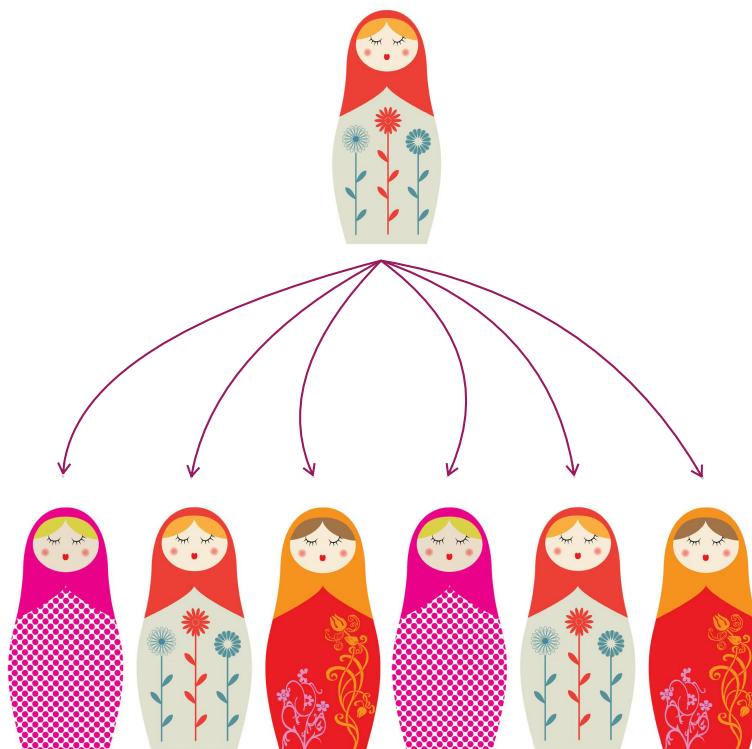
- En otras palabras, cada hoja, se re-entrena para corregir o complementar las predicciones del ensemble actual sobre los puntos que pertenecen a esa hoja, asumiendo que los árboles se ensamblarán aditivamente

$$F_T(x) = \sum_{i=1}^T \sum_m \gamma_{im} I(x \in R_{im})$$

Gradient Tree Boosting



Mezclas de Expertos



Mezclas de Expertos

Under review as a conference paper at ICLR 2017

OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer¹, Azalia Mirhoseini^{*†1}, Krzysztof Maziarz^{*2}, Andy Davis¹, Quoc Le¹, Geoffrey Hinton¹ and Jeff Dean¹

¹Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com

²Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

Learning Factored Representations in a Deep Mixture of Experts

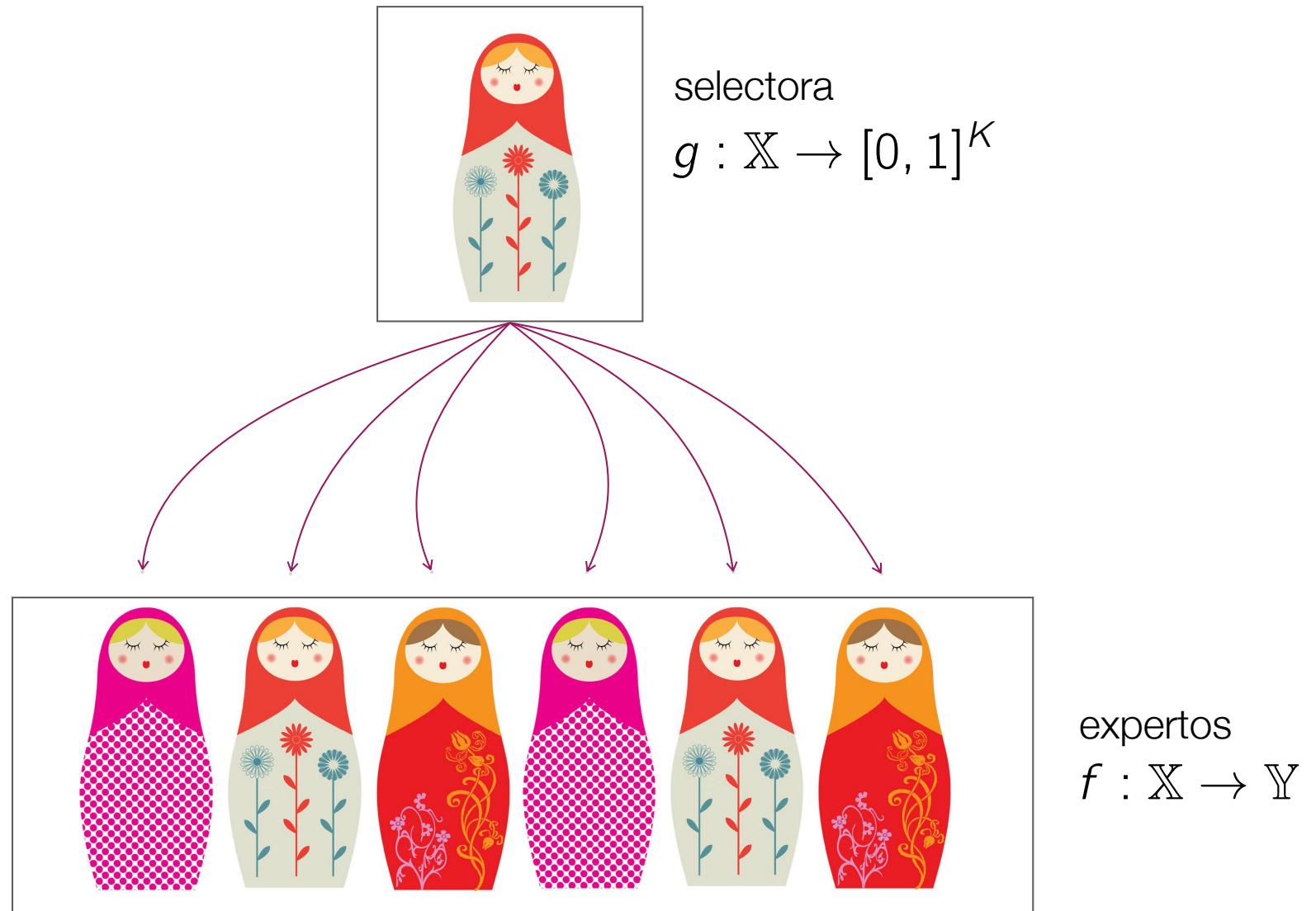
David Eigen ^{1,2} **Marc'Aurelio Ranzato** ^{1 *} **Ilya Sutskever** ¹

¹ Google, Inc.

² Dept. of Computer Science, Courant Institute, NYU

deigen@cs.nyu.edu ranzato@fb.com ilyasu@google.com

Idea



Posibles Formulaciones

- Modelo

$$F(x) = \sum_i g_i(x) f_i(x)$$

- Selector Softmax

$$g_i(x) = \frac{\exp(w_i^T x)}{\sum_i \exp(w_i^T x)}$$

- Expertos regresores

$$f_i(x) = \theta_i^T x \approx \mathbb{E}(y|x)$$

Posibles Formulaciones

- Modelo

$$F(x) = \sum_i g_i(x) f_i(x)$$

- Selector Softmax

$$g_i(x) = \frac{\exp(w_i^T x)}{\sum_i \exp(w_i^T x)}$$

- Expertos clasificadores

$$\begin{aligned}f_i(x) &= q_i(y|x; \theta_i) \approx P(y = 1|x) \\&= \sigma(\theta_i^T x) = \frac{1}{1 + \exp(-\theta_i^T x)}\end{aligned}$$

Posibles Formulaciones

- Modelo

$$F(x) = \sum_i g_i(x) f_i(x)$$

- Selector Softmax

$$g_i(x) = \frac{\exp(w_i^T x)}{\sum_i \exp(w_i^T x)}$$

- Expertos no-supervisados (e.g., gausianos)

$$\begin{aligned}f_i(x) &= p_i(x) \approx P(x) \\&= \kappa_i \exp(-\gamma_i(x - \mu_i)^2)\end{aligned}$$

Entrenamiento

- Dado un conjunto de ejemplos independientes

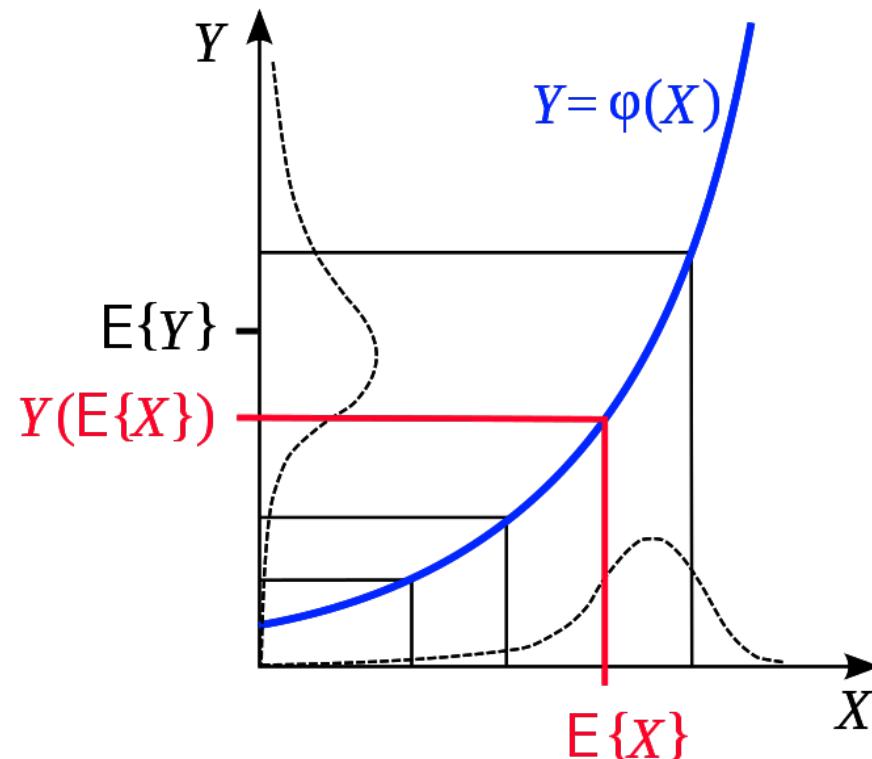
$$\begin{aligned} I(\Theta) &= \sum_{\ell} \ln P(y^{(\ell)} | x^{(\ell)}; \Theta) \\ &= \sum_{\ell} \ln \left(\sum_i g_i(x^{(\ell)}) q_i(y^{(\ell)} | x^{(\ell)}) \right) \end{aligned}$$

- Función objetivo muy difícil de optimizar. No convexa.
- Si la selectora debe elegir un experto, el problema es NP completo.

Jensen

- Si φ es convexa:

$$\varphi(\mathbb{E}(X)) \leq \mathbb{E}(\varphi(X))$$



Jensen

- Si φ es concava:

$$\varphi(\mathbb{E}(X)) \geq \mathbb{E}(\varphi(X))$$

- La igualdad vale si y sólo si (con probabilidad 1)

$$X = \text{cte}$$

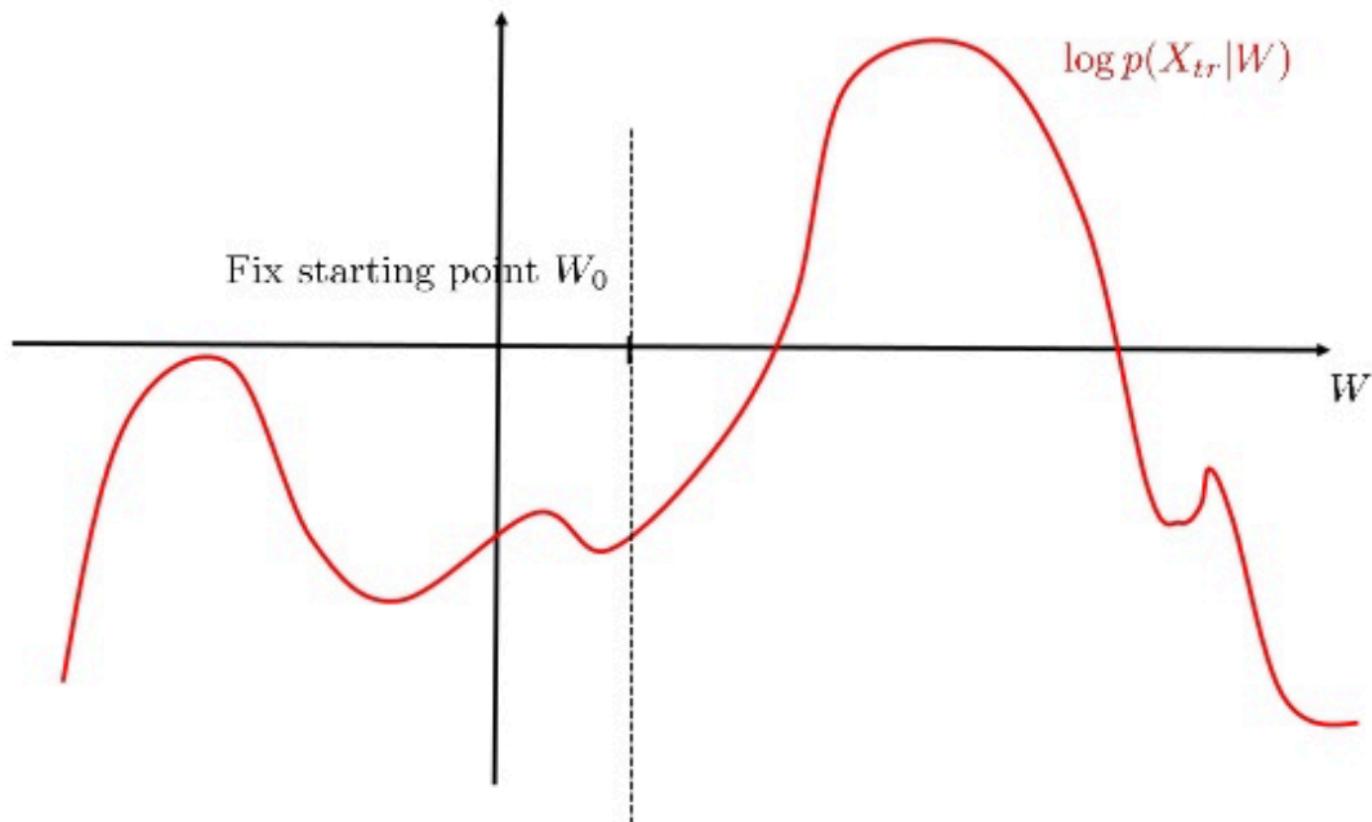
Algoritmo EM

- Para cualquier distribución Q sobre i

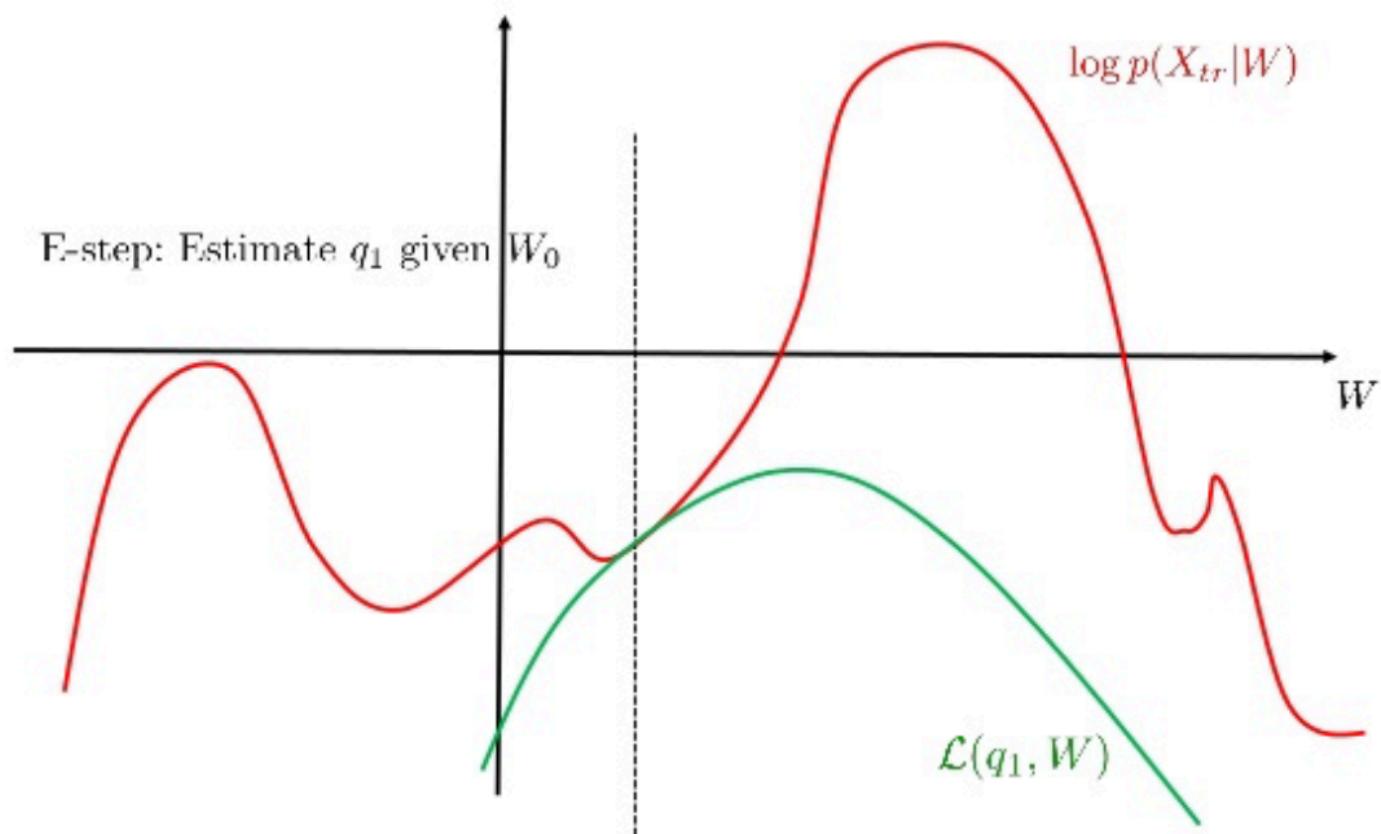
$$\begin{aligned} I(\Theta) &= \sum_{\ell} \ln \left(\sum_i Q(i|x^{(\ell)}) \frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{Q(i|x^{(\ell)})} \right) \\ &\geq \sum_{\ell} \sum_i Q(i|x^{(\ell)}) \ln \left(\frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{Q(i|x^{(\ell)})} \right) \end{aligned}$$

- Idea: algoritmo iterativo en dos pasos
 - **Paso E:** Elegir Q para que la cota es buena
 - **Paso M:** Maximizar la cota inferior

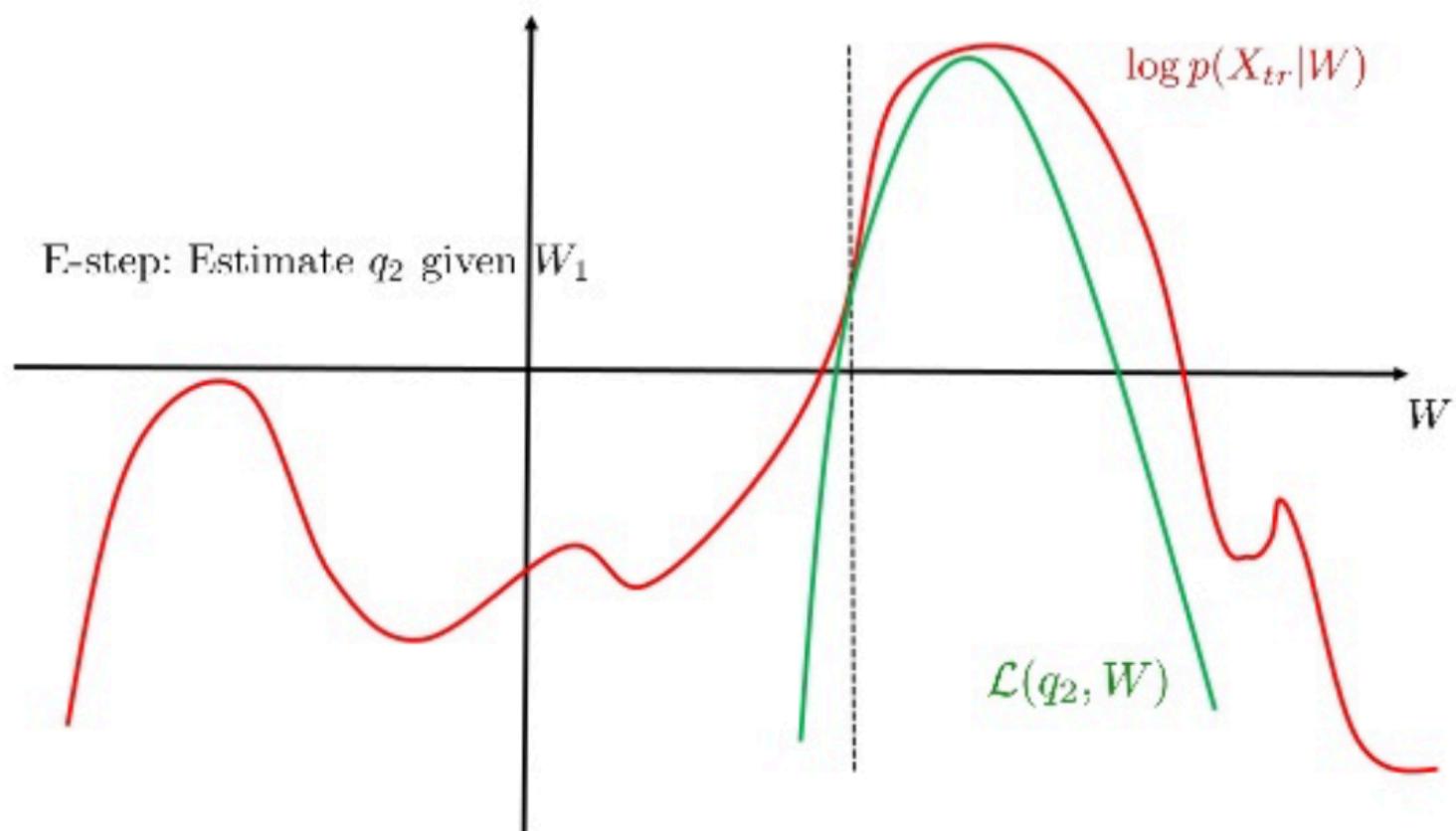
Algoritmo EM



Algoritmo EM



Algoritmo EM



Algoritmo EM para ME

- **Paso E.** En Jensen, la igualdad vale si y sólo si (con probabilidad 1)

$$X = \text{cte}$$

- Podemos elegir Q para “X” sea constante

$$\frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{Q(i|x^{(\ell)})} = \text{cte}$$

- Esto implica que

$$Q(i|x^{(\ell)}) \propto g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})$$

$$Q(i|x^{(\ell)}) = \frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{\sum_i g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}$$

Algoritmo EM para ME

- **Paso M.** Maximizamos la cota.

$$l'(\Theta) = \sum_{\ell} \sum_i Q(i|x^{(\ell)}) \ln \left(\frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{Q(i|x^{(\ell)})} \right)$$

- Para la red selectora

$$\operatorname{argmax}_W l_{\text{inf}}(\Theta) = \operatorname{argmax}_W \sum_{\ell} \sum_i Q(i|x^{(\ell)}) \ln g_i(x^{(\ell)})$$

- Esta optimización es muy sencilla. Permite entrenamiento vía gradiente y soluciones exactas en casos sencillos.

Algoritmo EM para ME

- **Paso M.** Maximizamos la cota.

$$l'(\Theta) = \sum_{\ell} \sum_i Q(i|x^{(\ell)}) \ln \left(\frac{g_i(x^{(\ell)}) q_i(y^{(\ell)}|x^{(\ell)})}{Q(i|x^{(\ell)})} \right)$$

- Para el i-ésimo experto

$$\operatorname{argmax}_W l_{\text{inf}}(\Theta) = \operatorname{argmax}_W \sum_{\ell} Q(i|x^{(\ell)}) \ln q_i(y^{(\ell)}|x^{(\ell)})$$

- Esta optimización es muy sencilla. Permite entrenamiento vía gradiente y soluciones exactas en casos sencillos. En la formulación presentada para clasificación, se trata de una simple regresión logística con pesos.

Garantías

- Sea $\Theta^{t+1/2}$ la solución obtenida después de ejecutar el t-ésimo paso E
- Sea Θ^{t+1} la solución obtenida después de ejecutar el t-ésimo paso M

$$I(\Theta^t) = I_{\inf}(\Theta^{t+1/2}) \leq I_{\inf}(\Theta^{t+1}) \leq I(\Theta^{t+1})$$

Teorema

El algoritmo de entrenamiento obtenido es monótono (no puede empeorar la solución).

- Implica esto convergencia al óptimo?