Tema 3 – Combinación de estimadores

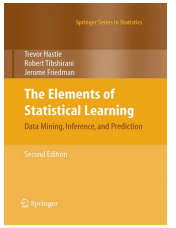
Aprendizaje Automático II - Grado en Inteligencia Artificial
Universidad Rey Juan Carlos

Iván Ramírez Díaz
ivan.ramirez@urjc.es

José Miguel Buenaposada Biencinto
josemiguel.buenaposada@urjc.es

Bibliografía

- The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Edition). Capítulo 10. Trevor Hastie, Robert Tibshirani, Jerome Friedman. Springer.



3. Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

3. Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

Combinación por votación de la mayoría

- Entrenamos K clasificadores. Clasificamos con los K clasificadores y la clase ganadora es la que más votos recibe de los K posibles.

¿Cuándo y por qué funciona?

- Supongamos K=25 clasificadores con errores estadísticamente independientes.
- Supongamos tasa de error de cada clasificador err = 0,35
- La probabilidad de que más de la mitad de los clasificadores se equivoquen:

$$P(X > 12) = \sum_{i=13}^{25} \binom{25}{i} err^i (1 - err)^{25-i} = 0,06$$

Nº Combinaciones de i clasificadores (de los 25)

Probabilidad de i clasificadores se equivocan

Probabilidad de 25-i clasificadores aciertan

ejemplo de Evgueni Smirnov

Combinación por votación de la mayoría

- Entrenamos K clasificadores. Clasificamos con los K clasificadores y la clase ganadora es la que más votos recibe de los K posibles.

¿Cuándo y por qué funciona?

- Supongamos K=25 clasificadores con errores estadísticamente independientes.
- Supongamos tasa de error de cada clasificador err = 0,35

La clave es que los clasificadores sean estadísticamente independientes en sus respuestas:
La respuesta de un clasificador no depende de lo que haya dicho el otro y no tienen por qué equivocarse en los mismos ejemplos.

ejemplo de Evgueni Smirnov

3 Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

Árboles de Decisión y/o regresión

- Se particiona el espacio de características en un conjunto de regiones { R_i }
- En cada región se devuelve siempre la misma respuesta, γ_i, con lo que estima el árbol es:

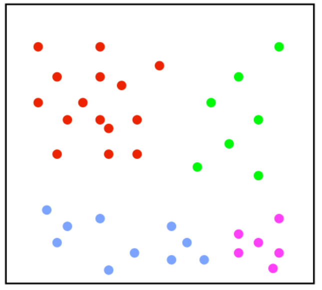
$$T(\boldsymbol{x};\theta) = \sum_{j=1}^{N_R} \gamma_j \cdot I(x \in R_j)$$

donde

$$\theta = \{(R_j, \gamma_i)\}_{j=1}^{N_R}$$

Recordatorio Árboles de Decisión: Entrenamiento

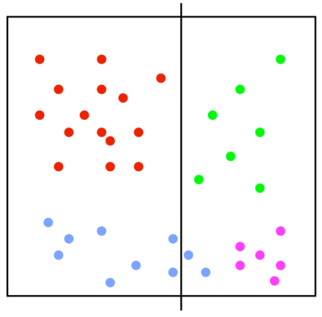
- Se particiona recursivamente el conjunto de entrenamiento con test de respuesta binaria sobre los vectores de características.
- Hasta que cada subcobjunto contiene instancias de una sola clase.



(transparencia David Capel)

Recordatorio Árboles de Decisión: Entrenamiento

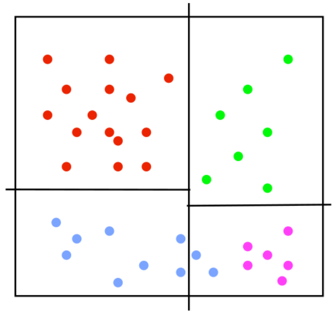
- Se particiona recursivamente el conjunto de entrenamiento con test de respuesta binaria sobre los vectores de características.
- Hasta que cada subcobjunto contiene instancias de una sola clase.



(transparencia David Capel)

Recordatorio Árboles de Decisión: Entrenamiento

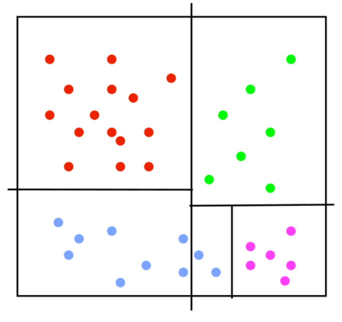
- Se particiona recursivamente el conjunto de entrenamiento con test de respuesta binaria sobre los vectores de características.
- Hasta que cada subcobjunto contiene instancias de una sola clase.



(transparencia David Capel)

Recordatorio Árboles de Decisión: Entrenamiento

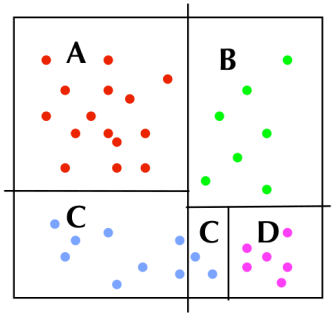
- Se particiona recursivamente el conjunto de entrenamiento con test de respuesta binaria sobre los vectores de características.
- Hasta que cada subcobjunto contiene instancias de una sola clase.



(transparencia David Capel)

Recordatorio Árboles de Decisión: Entrenamiento

- Se particiona recursivamente el conjunto de entrenamiento con test de respuesta binaria sobre los vectores de características.
- Hasta que cada subcobjunto contiene instancias de una sola clase.

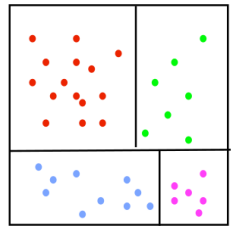


(transparencia David Capel)

Recordatorio Árboles de Decisión: Reglas de decisión

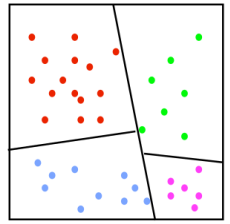
- Cortes con hiperplanos paralelos a los ejes: Poner umbral a una única característica en cada nodo. Rápida de evaluar.

$$x_i > T$$



- Cortes con hiperplanos sin resticciones. Poner umbral a una combinación lineal de características. Más costosa pero produce árboles más pequeños.

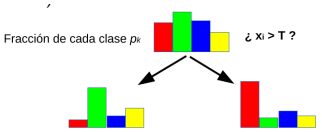
$$\boldsymbol{w}^T \boldsymbol{x} > T$$



(transparencia David Capel)

Recordatorio Árboles de Decisión: Elección de la mejor partición

- Lugar del corte elegido para optimizar alguna medida de rendimiento en los subconjuntos de datos que quedan en los hijos.
- Se buscan subconjuntos en los nodos hijo con menor entropía (desorden o confusión):



Proporción de datos de entrenamiento de la clase k en el conjunto A

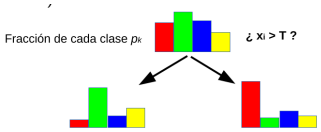
Entropía de un conjunto de datos A: $H(A) = -\sum_k (p_k^A \log(p_k^A))$

La ganancia de información mide la disminución de la entropía (del desorden) al dividir los datos en dos subconjutos. Una distribución uniforme tiene más desorden que una en que una de las clases tiene toda la probabilidad.

(transparencia David Capel)

Recordatorio Árboles de Decisión: Elección de la mejor partición

- Lugar del corte elegido para optimizar alguna medida de rendimiento en los subconjuntos de datos que quedan en los hijos.
- Se buscan subconjuntos en los nodos hijo con menor entropía (desorden o confusión):



Entropía de un conjunto de datos A: $H(A) = -\sum_k (p_k^A \log(p_k^A))$

Ganancia de información:

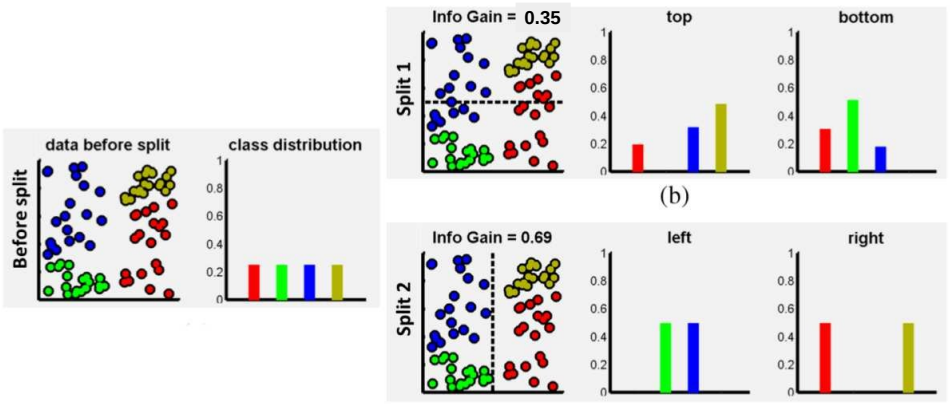
Nº de datos de entrenamiento en el nodo actual

Nº de datos de entrenamiento en el subárbol derecho

$$I_E = H(S) - \frac{|S_{izq}|}{|S|} H(S_{izq}) - \frac{|S_{der}|}{|S|} H(S_{der})$$

(transparencia David Capel)

Recordatorio Árboles de Decisión: Elección de la mejor partición



Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning andSemi-Supervised Learning
Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. 2012)

Árboles de Regresión

- Lugar del corte elegido para optimizar alguna medida de rendimiento en los subconjuntos de datos que quedan en los hijos.
- Se busca particionar el espacio de características en 2 subcjtos:

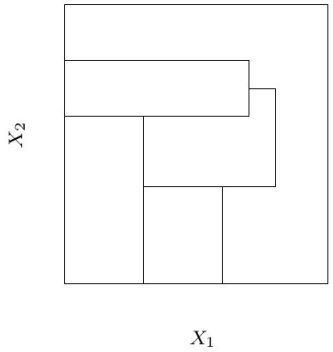
$$R_1(j,s) = \{X | X_j \leq s\} \qquad R_2(j,s) = \{X | X_j > s\}$$

tal que en los nodos hijo se tenga el menor error cuadrático:

$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right]$$

y donde la respuesta del árbol, c_m, en cada región R_m sea la media de los valores de los y_i de entrenamiento que caen en ella.

Árboles de Regresión



Esta partición del espacio de características no se puede aprender con la estrategia de partición recursiva anterior

Árboles de Regresión

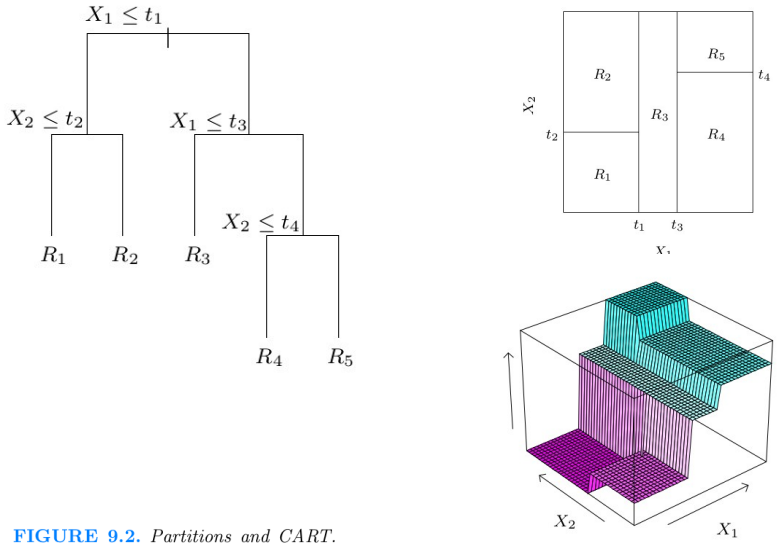


FIGURE 9.2. *Partitions and CART.*

Árboles de Decisión: Discusión

- Ventajas:
  - Entrenamiento rápido y fácil de implementar
  - Se puede manejar fácilmente gran número de características
- Desventajas:
  - Siempre se puede construir un árbol de decisión con 100% de acierto en el conjunto de datos de entrenamiento (sesgo bajo)
  - Tienden al sobreaprendizaje y no generalizan bien (sesgo bajo, varianza alta).

¿Qué se puede hacer para mejorar las limitaciones de los Árboles de Decisión?

(transparencia David Capel)

Combinación de clasificadores: Bagging

- Algoritmo **B**ootstrapping **Agg**regat**ing** (Leo Breiman 1996)
  - Para $i = 1 \ldots M$
    - Obtener $n^* < n$ muestra de $\mathcal{D}$ con reemplazamiento.
    - Aprender un clasificador C_i sobre la nueva muestra.
  - El clasificador final es una votación de los $C_i, \ldots, C_M$.
- El clasificador resultante del Bagging:
  - Incrementa la estabilidad del clasificador
  - Reduce la varianza:
    - Mejora las técnicas de varianza alta (Redes de neuronas, Árboles de decisión)
    - No mejora las técnicas de varianza baja (Un clasificador lineal)

Random Forest (= Bagging de árboles)

- Combinación de M árboles de decisión binarios mediante votación de la mayoría
- Introducir aleatoriedad en el aprendizaje (entrenar árboles estadísticamente independientes):
  - Mediante Bagging
  - Mediante la elección aleatoria de subconjuntos de características
  - Seleccionando varios umbrales aleatoriamente y eligiendo el de mayor ganancia de información, I_E (Extremely Randomized Trees)

(transparencia David Capel)

Radom Forest: Discusión

- Reduce el problema de la varianza en los árboles individuales con M árboles con errores estadísticamente independientes.
- No reduce el sesgo por lo que hay que utilizar árboles con un número de niveles, L, adecuado al problema.
- El entrenamiento y la ejecución del Random Forest se puede paralelizar (cada árbol por separado).

Parámetros relacionados y críticos:
N.º de árboles, T (reducción de la varianza), y el número máximo de niveles de los árboles, L (sesgo)

Random Forest: Ejemplos

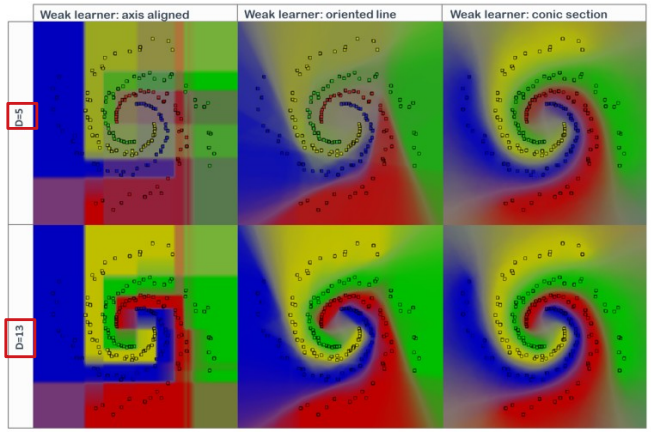


Fig. 3.7 The effect of the weak learner model. The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. For fixed weak learner deeper trees produce larger confidence. For constant D nonlinear weak learners produce the best results. In fact, an axis-aligned weak learner model produces blocky artifacts while the curvilinear model tends to extrapolate the shape of the spiral arms in a more natural way. Training has been achieved with $\rho = 500$ for all split nodes. The forest size is kept fixed at $T = 400$.

En el ejemplo D es la profundidad de los árboles (depth), T es el número de árboles y ρ es el número de características aleatorias usadas en el entrenamiento de cada árbol.

Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning andSemi -Supervised Learning
Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. 2012)

Random Forest: Ejemplos

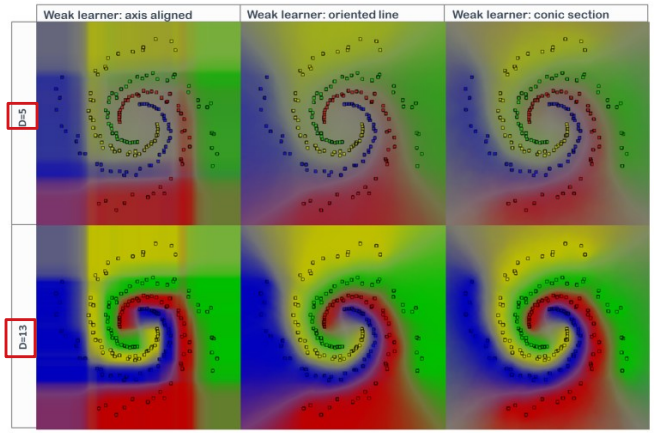


Fig. 3.8 The effect of randomness. The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. This experiment is identical to that in Figure 3.7 except that we have used much more training randomness. In fact $\rho = 5$ for all split nodes. The forest size is kept fixed at $T = 400$. More randomness reduces the artifacts of the axis-aligned weak learner a little, as well as reducing overall prediction confidence too. See text for details.

En el ejemplo D es la profundidad de los árboles (depth), T es el número de árboles y ρ es el número de características aleatorias usadas en el entrenamiento de cada árbol.

Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning andSemi -Supervised Learning
Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. 2012)

3. Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

Combinación de clasificadores: Boosting

- Idea: aprender un clasificador fuerte combinado clasificadores básicos o débiles (un poco mejores que el azar).
- Construiremos una combinación de clasificadores (*Ensemble*) donde cada clasificador contribuye según su peso (estimado con el entrenamiento del Boosting).
- AdaBoost, *Adaptive Boosting* (Freund y Shapire 1995)

AdaBoost discreto: planteamiento

- Problema de clasificación biclase
- Datos de entrenamiento, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ con
  - $\mathbf{x}_i$ vector de características,
  - $y \in \{-1, 1\}$ etiqueta de clase.
- Y obtenemos un clasificador fuerte: $G(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$.

Algoritmo Adaboost discreto

Asumiendo la condición de clasificador-débil: $err_m \leq 1/2$

1. Inicializar los pesos $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. Para $m = 1, \ldots, M$
   1. Entrenar un clasificador básico, $G_m(\mathbf{x})$, con los datos y sus pesos w_i.
   2. Estimar el error de $G_m(\mathbf{x})$: $err_m = \sum_{i=1}^{N}(w_i I(y_i \neq G_m(\mathbf{x}_i))$.
   3. Calcular la importancia de $G_m(\mathbf{x})$: $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
   4. Establecer los nuevos pesos: $w_i \leftarrow w_i e^{(\alpha_m I(y_i \neq G_m(\mathbf{x}_i))},\ i = 1, 2, \ldots, N$.
   5. Normalizar los pesos $w_i \leftarrow \frac{w_i}{\sum_{i=1}^{N} w_i}$.
3. Devolver: $G(\mathbf{x}) = signo\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right)$

Algoritmo Adaboost discreto

Importancia de cada G_m (paso 2.3 del algoritmo):

$$\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$$

- Si $err_m \leq ½ \rightarrow \alpha_m \geq 0$
- Cuanto más pequeño es err_m, más grade es $\alpha_m \rightarrow \alpha_m$ modela "lo bueno" que es cada Gm clasificando ("su importancia").

Algoritmo Adaboost discreto

Cálculo de los w_i (paso 2.4 del algoritmo):

$$w_i \leftarrow w_i e^{(\alpha_m I(y_i \neq G_m(\mathbf{x}_i))},\ i = 1, 2, \ldots, N$$

$$e^{(\alpha_m I(y_i \neq G_m(\mathbf{x}_i))} = \begin{cases} 1, & y_i = G_m(\mathbf{x}_i) \\ e^{\alpha_m}, & y_i \neq G_m(\mathbf{x}_i) \end{cases}$$

- Incrementa el peso de los ejemplos mal clasificados
- Deja sin cambiar el peso de ejemplos bien clasificados
- Lo "aprendido" sobre ejemplos en iteraciones anteriores está en w_i.

Entrenar un clasificador débil con pesos

Entrada

Datos de entrenamiento: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
Pesos: $(w_1, \ldots, w_N)$

Dos opciones

1. Utilizar un clasificador capaz de usar la distribución de pesos w_i (p.ej. árboles de decisión).

Entrenar un clasificador débil con pesos

Entrada

Datos de entrenamiento: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
Pesos: $(w_1, \ldots, w_N)$

Dos opciones

2. Obtener una muestra $(\hat{\mathbf{x}}_1, y_1), \ldots, (\hat{\mathbf{x}}_n, y_n)$, $(n < N)$, utilizando la distribución de pesos w_i:
   - Los datos con más peso se usarán con mayor probabilidad en el entrenamiento.
   - Un valor de $n = 0{,}1 \cdot N$ (10 % de N) es un buen compromiso.
   - El entrenamiento se hará sin pesos pero en la nueva muestra (obtenida con pesos).

Clasif. débil elegido de un conjunto predefinido

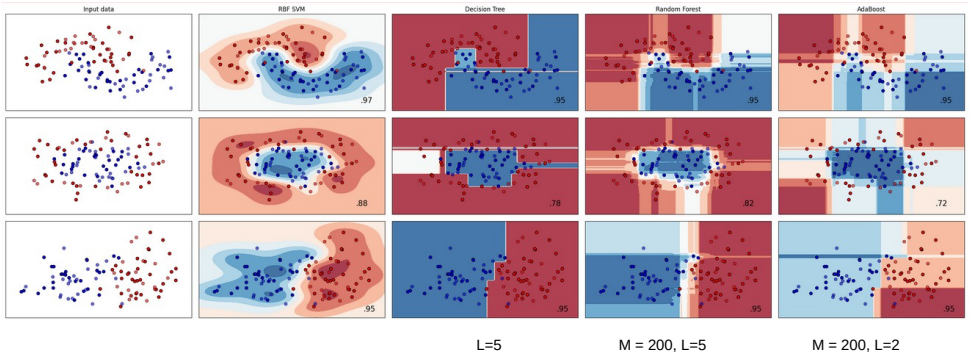
Entrada

Datos de entrenamiento: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
Pesos: $(w_1, \ldots, w_N)$
Conjunto de clasificadores: $\{h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})\}$

Procedimiento

- Clasificar los datos de entrenamiento con cada $h_k(\mathbf{x})$.
- Calcular el error de cada $h_k(\mathbf{x})$: $err_k = \sum_{i=1}^{N} (w_i I(y_i \neq h_k(\mathbf{x}_i))$.
- Elegir el clasificador entrenado con el menor error, err_k.

Ejemplos AdaBoost



AdaBoost ajusta un modelo aditivo

- Ajusta un modelo aditivo con una “base” (los $G_m(\mathbf{x})$)
- La aproximación de la función $f(\mathbf{x})$ será:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m),$$

- Para ajustar la aproximación se minimiza la pérdida esperada:

$$\text{mín}_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i, \gamma_m)\right)$$

Problema: para muchas $L(y, f(\mathbf{x}))$ y/o $b(\mathbf{x}, \gamma)$ encontrar el mínimo es costoso.

AdaBoost ajusta un modelo aditivo

Es más fácil resolver el problema para una única $b(\mathbf{x}, \gamma)$:

$$\text{mín}_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, \beta b(\mathbf{x}_i, \gamma))$$

Algoritmo de modelado aditivo iterativo:

1. Inicializar $f_0(\mathbf{x}) = 0$.
2. Para $m = 1 \ldots M$
   - Calcular:
   $$(\beta_m, \gamma_m) = arg\ \text{mín}_{\beta_m,\gamma_m} \sum_{i=1}^{N} L(y, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$
   - Establecer:
   $$f_m(\mathbf{x}) = f_{m-1} + \beta_m b(\mathbf{x}, \gamma_m)$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

¿Por qué la función exponencial?

Porque el f(x) que minimiza la pérdida esperada condicionada a x:

$$f(\boldsymbol{x}) = arg\,min_{f(x)} E_{P(y|x)}[e^{-yf(\boldsymbol{x})}] = \frac{1}{2}\ \log\left(\frac{Pr(y=1 \mid \boldsymbol{x})}{Pr(y=-1 \mid \boldsymbol{x})}\right)$$

Cuando la probabilidad a posteriori de la clase 1 es mayor que la de la clase -1, el resultado es positivo. Y negativo en caso contrario → Igual que el clasificador Estadístico de Bayes.

Sec. 10.5 libro Hastie y Tibshirani

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(\mathbf{x}_i) + \beta G(\mathbf{x}_i))}$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(\mathbf{x}_i)+\beta G(\mathbf{x}_i))}$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \sum_{i=1}^{N} \underbrace{e^{-y_i f_{m-1}(\mathbf{x}_i)}}_{w_i^{(m)}} \cdot e^{-y_i \beta G(\mathbf{x}_i)}$$

$w_i^{(m)}$ no depende de $G(\mathbf{x})$ ni de β, depende de f_{m-1}.

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \sum_{i=1}^{N} w_i^{(m)} \cdot e^{-y_i \beta G(\mathbf{x}_i)}$$

$$\forall \beta > 0, \; -y_i \beta G(\mathbf{x}_i) = \begin{cases} < 0, & si \;\; acierto \\ > 0, & si \;\; fallo \end{cases}$$

Por tanto, buscamos G_m que minimiza la tasa de error (con pesos):

$$G_m = arg \min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i))$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \sum_{i=1}^{N} w_i^{(m)} \cdot e^{-y_i \beta G(\mathbf{x}_i)}$$

$$e^{-y_i \beta G(\mathbf{x}_i)} = \begin{cases} e^{-\beta}, & si \;\; acierto \\ e^{\beta}, & si \;\; fallo \end{cases}$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \left(e^{-\beta} \sum_{y_i = G(\mathbf{x})} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(\mathbf{x})} w_i^{(m)} \right)$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \underbrace{\left((e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} \right)}_{J(\beta)}$$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

$$(\beta_m, G_m) = arg \min_{\beta_m, G_m} \underbrace{\left((e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} \right)}_{J(\beta)}$$

$\frac{\partial J(\mathbf{x})}{\partial \beta} = (e^{\beta} + e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) - e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}$

Igualando a 0 para encontrar el mínimo:

$(e^{\beta} + e^{-\beta}) e^{\beta} = \frac{1}{err_m} \Rightarrow e^{2\beta} + 1 = \frac{1}{err_m} \Rightarrow \beta_m = \frac{1}{2} log \left(\frac{1 - err_m}{err_m} \right)$

AdaBoost y la pérdida exponencial

AdaBoost es un algoritmo de modelado aditivo iterativo que utiliza la función de pérdida exponencial:

$$L(y, f(x)) = e^{-yf(\mathbf{x})}$$

Usando los clasificadores débiles $G_m(\mathbf{x}) : \mathbb{R}^n \to \{-1, 1\}$

- $G_m = arg \min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i))$.
- $\beta_m = \frac{1}{2} log \left(\frac{1 - err_m}{err_m} \right)$.
- $err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i))}{\sum_{i=1}^{N} w_i^{(m)}}$.

$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m G_m(\mathbf{x})$

Y los pesos a: $w_i^{(m+1)} = w_i^{(m)} e^{-\beta_m y_i G_m(\mathbf{x}_i)} = w_i^{(m)} e^{\alpha_m I(y_i \neq G_m(\mathbf{x}_i))} e^{-\beta_m}$:

- $\alpha_m = 2\beta_m$
- $e^{-\beta_m}$ multiplica a todos los pesos $\Rightarrow$ se puede quitar.

AdaBoost Discreto (AdaBoost.M1)

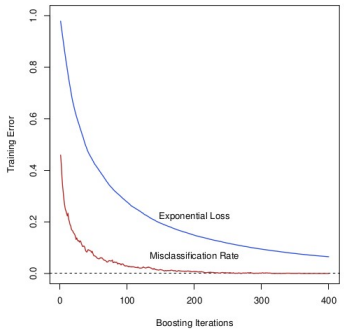


FIGURE 10.3. *Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss: $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$. After about 250 iterations, the misclassification error is zero, while the exponential loss continues to decrease.*

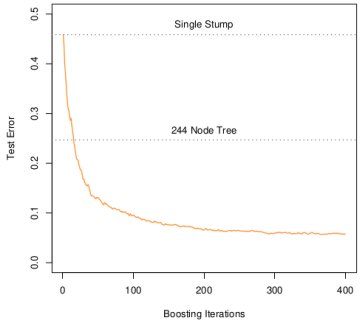
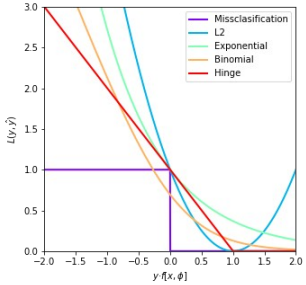


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Funciones de pérdida y el margen: y·f(x)

- Error de clasificación:

  $L(y, f(x)) = I(signo(f(x) \neq y))$
- La exponencial y binomial aproximaciones continuas al error de clasificación
- Binomial penaliza menos las observaciones con margen muy negativo (mal clasificadas) que la exponencial.
- Si las clases no son perfectamente separables AdaBoost tendrá peor comportamiento.
- Error cuadrático da importancia a observaciones bien y mal clasificadas.



AdaBoost ...

- Decrece exponencialmente el error de entrenamiento en el número de iteraciones
- Funciona bien si los clasificadores básicos tienen sesgo alto y baja varianza (“no muy complejos”)
- Reduce el sesgo del error de los clasificadores básicos y si se regulariza puede reducir la varianza

Otros algoritmos de Boosting

Utilizan otra función de pérdida diferente de la exponencial:

- Logistic Regression Boosting, LogitBoost (Friedman et al., Collins et al., 2000),
$$L(y, f(x)) = log(1 + e^{-2yf(\mathbf{x})}).$$
- Least Square Boosting (Friedman, 1999),
$$L(y, f(x)) = (y - f(\mathbf{x}))^2.$$
- Boosting with Soft Margins (Rätsch et al., 2001a, 2001b)
$$L(y, f(x)) = log(1 + e^{1-yf(\mathbf{x})}).$$

… y otros muchos “especializados” para una L concreta.

3. Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

SAMME (AdaBoost Multiclase): Planteamiento

SAMME = Stagewise Additive Modeling using Multi-class Exponential loss function

Ji Zhu, Hui Zou, Saharon Rosset, Trevor Hastie. "Multi-class AdaBoost". Journal of Statistics and Its Interface. Volumen 2, 2009. Págs 349-360

SAMME (AdaBoost Multiclase): Planteamiento

- Problema de clasificación con K clases
- Datos de entrenamiento, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ con
  - $\mathbf{x}_i$ vector de características,
  - $\mathbf{y}_i = (y_1, y_2, \ldots, y_K)^\top$ es un vector que representa la clase:

$$y_c = \begin{cases} 1, & si \quad c = K \\ -\frac{1}{K-1}, & si \quad c \neq K \end{cases}$$

- Ahora buscamos $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}))^\top$ tal que

$$\begin{aligned} &\text{mín}_{\mathbf{f}(\mathbf{x})} \sum_{i=1}^{N} L(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i)) \\ &\text{sujeto a: } f_1(\mathbf{x}) + \ldots + f_K(\mathbf{x}) = 0 \end{aligned}$$

- La función de pérdida es la exponencial multiclase:

$$L(\mathbf{y}, \mathbf{f}) = e^{(-\frac{1}{K}(y_1 f_1 + \ldots + y_K f_K))} = e^{(-\frac{1}{K}\mathbf{y}^\top \mathbf{f})}$$

SAMME (AdaBoost Multiclase): Algoritmo

Asumiendo la condición de clasificador-débil: $(1 - err_m) > 1/K$

1. Inicializar los pesos $w_i = 1/N, \ i = 1, 2, \ldots, N$.
2. Para $m = 1, \ldots, M$
   1. Entrenar un clasificador básico multiclase, $G_m(\mathbf{x})$, con los datos y sus pesos w_i.
   2. Estimar el error de $G_m(\mathbf{x})$: $err_m = \sum_{i=1}^{N} (w_i I(y_i \neq G_m(\mathbf{x}_i)))$
   3. Calcular la importancia de $G_m(\mathbf{x})$: $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right) + log(K-1)$
   4. Establecer los nuevos pesos: $w_i \leftarrow w_i e^{(\alpha_m I(y_i \neq G_m(\mathbf{x}_i)))}, \ i = 1, 2, \ldots, N$.
   5. Normalizar los pesos $w_i \leftarrow \frac{w_i}{\sum_{i=1}^{N} w_i}$..
3. Devolver: $G(\mathbf{x}) = arg\ \text{máx}_k \left(\sum_{m=1}^{M} \alpha_m I(G_m(\mathbf{x}) = k)\right)$

3. Combinación de estimadores

- Votación de la mayoría
- Bagging: Random Forest
- Boosting Biclase: AdaBoost
- Boosting Multiclase: SAMME
- Gradient Boosting

Gradient Boosting ajusta un modelo aditivo

- Ajusta un modelo aditivo con una "base" (los b(***x***; γ_m))
- La aproximación de la función ***f(x)*** será:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m),$$

- Para ajustar la aproximación se minimiza la pérdida esperada:

$$\text{mín}_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i, \gamma_m)\right)$$

Gradient Boosting ajusta un modelo aditivo

Es más fácil resolver el problema para una única *b(**x**, γ):*

$$(\beta_m, \gamma_m) = arg\ \text{mín}_{\beta_m,\gamma_m} \sum_{i=1}^{N} L(y, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$

Algoritmo de modelado aditivo iterativo:

1. Inicializar $f_0(\mathbf{x}) = 0$.
2. Para $m = 1 \ldots M$
   - Calcular:
   $$(\beta_m, \gamma_m) = arg\ \text{mín}_{\beta_m,\gamma_m} \sum_{i=1}^{N} L(y, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$
   - Establecer:
   $$f_m(\mathbf{x}) = f_{m-1} + \beta_m b(\mathbf{x}, \gamma_m)$$

Gradient Boosting ajusta un modelo aditivo

Descenso de gradiente:

Inicializar $f_0(\mathbf{x})$

Para todo m=1...M

Calcular gradiente $\boldsymbol{g_m} = \left.\frac{\partial L(y, f(x))}{\partial f(x)}\right|_{f=f_{m-1}}$

$f_m(x) = f_{m-1}(x) - \alpha \cdot \boldsymbol{g_m}$

Algoritmo de modelado aditivo iterativo:

1. Inicializar $f_0(\mathbf{x}) = 0$.
2. Para $m = 1 \ldots M$
   - Calcular:
   $$(\beta_m, \gamma_m) = arg\ \text{mín}_{\beta_m,\gamma_m} \sum_{i=1}^{N} L(y, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$
   - Establecer:
   $$f_m(\mathbf{x}) = f_{m-1} + \beta_m b(\mathbf{x}, \gamma_m)$$

Para optimizar una función de pérdida dada L ¡El estimador básico debe de estimar el gradiente de L cambiado de signo!

Gradient Boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Gradient Boosting

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance Binary cross-entropy	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Gradient Boosting: Clasificación

- En clasificación con la cross-entropy (multinomial deviance) se aprenden K árboles, uno por cada clase.
- Es decir, tendremos una f(x) para cada clase que será proporcional a la probabilidad a posteriori de cada clase (antes de pasar por la soft-max).

Gradient Boosting: profundidad del árbol

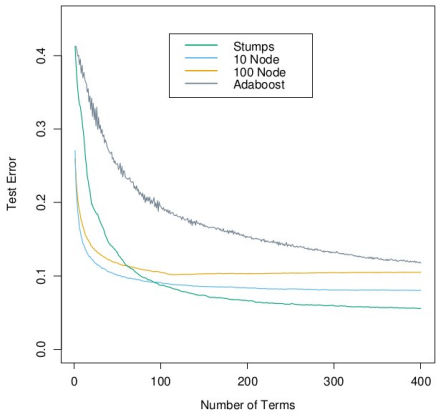
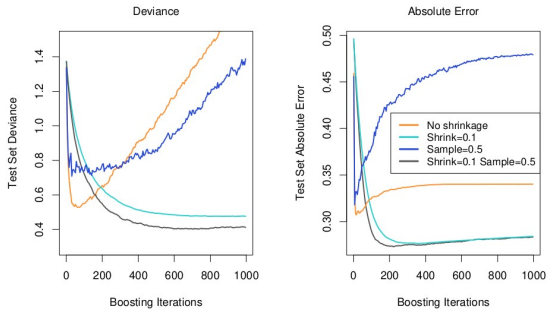


FIGURE 10.9. *Boosting with different sized trees, applied to the example (10.2) used in Figure 10.2. Since the generative model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost Algorithm 10.1.*

Regularización de Gradient Boosting

- Learning rate o Shrinkage. Factor 0 < s ≤ 1 que se multiplica por el parámetro β_i cada *weak learner* entrenado.
- Sampling rate. Factor 0 < r ≤ 1 que representa la proporción de datos de entrenamiento obtenidos por muestreo *sin reemplazamiento* para entrenar cada *weak learner*.

Binary cross-entropy



Deep Learning vs Gradient Boosting

- Para datos tabulados con “pocos datos” (10000):

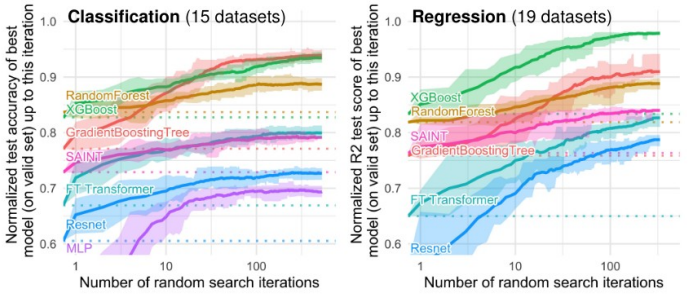


Figure 1: **Benchmark on medium-sized datasets, with only numerical features**. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

Soma Onishi, Masahiro Nishimura, Ryota Fujimura, Yoichi Hayashi: Why Do Tree Ensemble Approximators Not Outperform the Recursive-Rule eXtraction Algorithm? Mach. Learn. Knowl. Extr. 6(1): 658-678 (2024)