Tree Based Methods

Machine Learning

Ignacio Sarmiento-Barbieri

Universidad de La Plata

Agenda

Recap

- Queremos predecir y en funcion de observables $(\mathbf{x}_i)$

$$y = f(\mathbf{x}_i) + u \tag{1}$$

- donde la estimación de f implica la que minimize el riesgo empírico (prediga mejor fuera de muestra):

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \tag{2}$$

Recap

- Por ejemplo en regresión $f(x_i) = \mathbf{x}_i\beta$ y minimizamos

$$\hat{\beta} = \underset{\tilde{\beta}}{\text{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{x}_i\beta)^2 \right\} \tag{3}$$

- Por ejemplo en clasificación logistica $p(x_i) = \frac{1}{1+e^{-\mathbf{x}_i\beta}}$

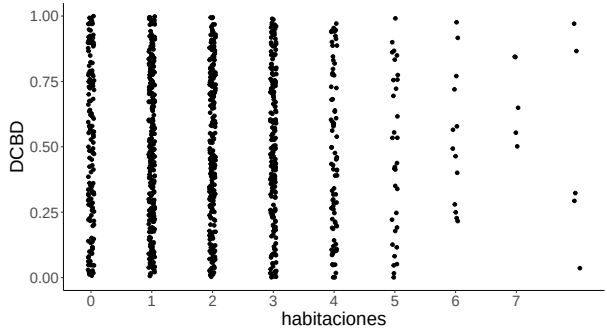
$$\hat{\beta}^{MLE} = \underset{\beta}{argmin} \; - \left[\sum_{i=1}^{n} log \left(\frac{p_i}{1-p_i} \right)^{y_i} + \sum_{i=1}^{n} log(1-p_i) \right] \tag{4}$$

Agenda

Motivación

- Queremos predecir:

$$Precio = f(habitaciones, Distancia\,al\,CBD) \tag{5}$$



Motivación

- Podemos asumir f es lineal

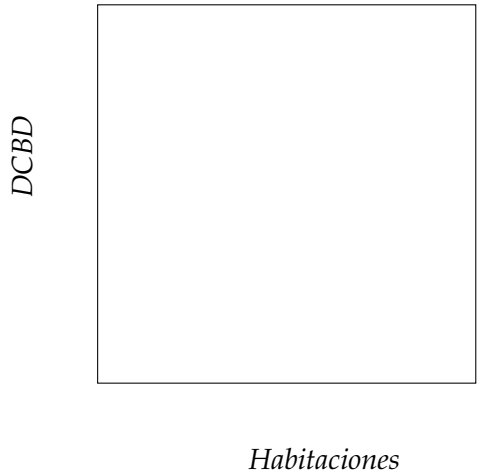
$$f(\mathbf{x}_i; \beta) = \beta_0 + \beta_1 Habitaciones_i + \beta_2 DCBD_i + u_i \tag{6}$$

- o ajustar un modelo flexible e interpretable como son los arboles

$$f(x_i; \{R_j, \gamma_j\}_1^J) = T(x_i; \{R_j, \gamma_j\}_1^J) \tag{7}$$

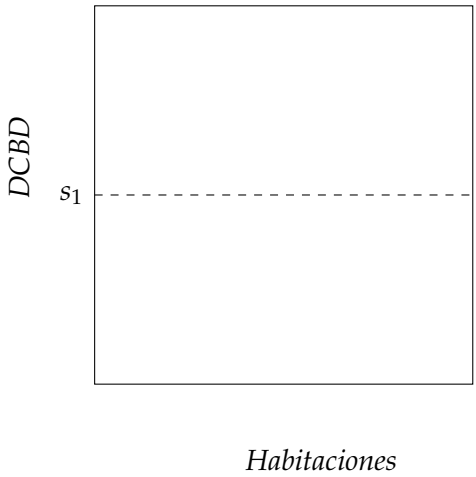
¿Qué hacen?

"Recursive binary splitting"



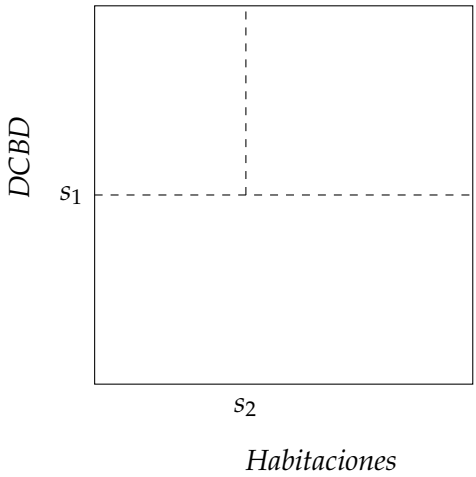
¿Qué hacen?

"Recursive binary splitting"

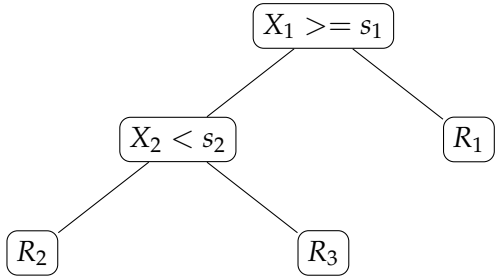


¿Qué hacen?

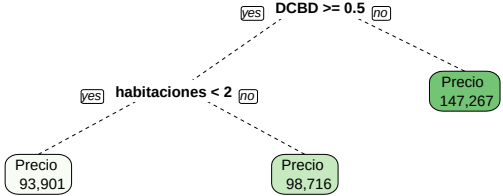
"Recursive binary splitting"

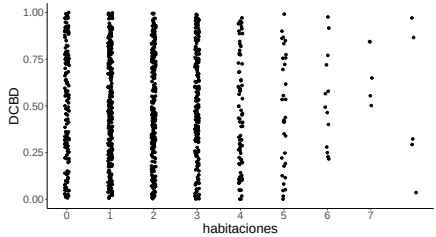


¿Qué hacen?



¿Qué hacen?


yes
DCBD >= 0.5
no
yes
habitaciones < 2
no
Precio
147,267
Precio
93,901
Precio
98,716


1.00
0.75
0.50
0.25
0.00
DCBD
0
1
2
3
4
5
6
7
habitaciones

¿Cómo construimos un árbol?

- El problema de optimización

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \tag{8}$$

- es ahora

$$\{\hat{R}_j, \hat{\gamma}_j\}_1^J = \underset{\{R_j, \gamma_j\}_1^J}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, T(\mathbf{x}_i; \{R_j, \gamma_j\}_1^J)) \right\} \tag{9}$$

¿Cómo construimos un árbol?

- Datos: $y_{n\times 1}$ y $X_{n\times k}$
- Definiciones
  - j es la variable que parte el espacio y s es el punto de partición
  - Defina los siguientes semiplanos

$$R_1(j,s) = \{X|X_j \leq s\} \;\; \& \;\; R_2(j,s) = \{X|X_j > s\} \tag{10}$$

  - El problema: usando una "perdida cuadrática" buscar la variable de partición X_j y el punto s de forma tal que:

$$\min_{j,s} \left[\min_{\gamma_{R_1}} \sum_{x_i \in R_1(j,s)} (y - \gamma_{R_1})^2 + \min_{\gamma_{R_2}} \sum_{x_i \in R_2(j,s)} (y - \gamma_{R_2})^2 \right] \tag{11}$$

¿Cómo construimos un árbol?

- ¿Cuál es la solución?

¿Cómo construimos un árbol?

photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/

¿Cómo construimos un árbol de decisión?

- Regiones lo más “puras” posibles
  - **Regresión**: minima varianza
  - **Clasificación**: ?

¿Cómo construimos un árbol de decisión?

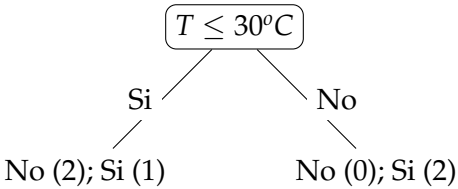
Problemas de clasificación

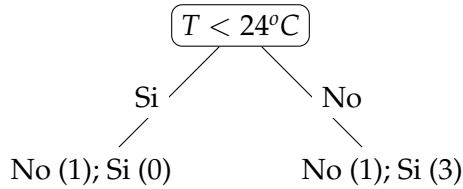
Temperatura °C	Llovió
23	NO
24	NO
29	SI
31	SI
33	SI

¿Cómo construimos un árbol de decisión?

Problemas de clasificación

- ¿Cuál de los dos cortes es mejor?





¿Cómo construimos un árbol de decisión?

Problemas de clasificación. Medidas de Impureza

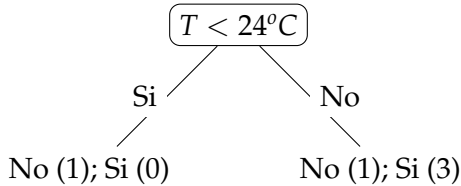
- Medidas de impureza dentro de cada hoja:
  - Índice de Gini : $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
  - Entropía : $-\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$
- Se define la impureza de un árbol por el promedio ponderado de las impurezas de cada hoja. El ponderador es la fracción de observaciones en cada hoja.

¿Cómo construimos un árbol de decisión?

Problemas de clasificación. Impureza

- ¿Cuál de los dos cortes es mejor?

Temperatura °C	Llovió
31	SI
24	NO
29	SI
33	SI
23	NO

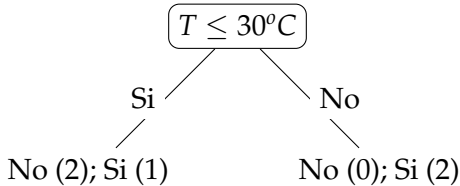


¿Cómo construimos un árbol de decisión?

Problemas de clasificación. Impureza

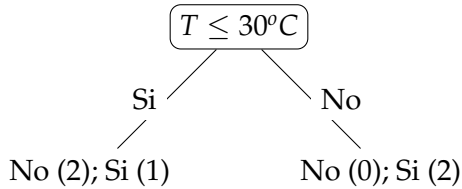
- ¿Cuál de los dos cortes es mejor?

Temperatura °C	Llovió
31	SI
24	NO
29	SI
33	SI
23	NO

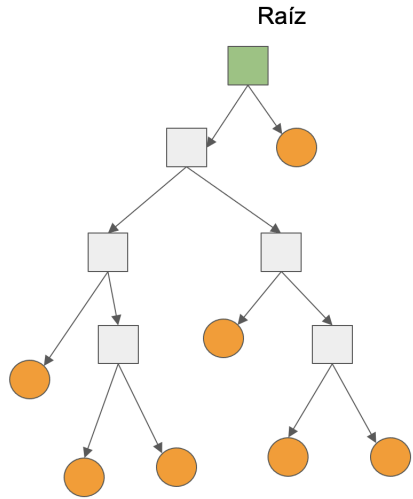


¿Cómo construimos un árbol de decisión?

Problemas de clasificación. Predicción



Sobreajuste



Sobreajuste. Algunas soluciones

- Fijar la profundidad del árbol.
- Fijar la mínima cantidad de datos que están contenidos dentro de cada hoja.
- Pruning (poda).
  - Dejar crecer un árbol muy grande T_0
  - Luego cortarlo obteniendo sub-árbol (*subtree*)
  - Como cortarlo?

Pruning (poda)

- No es posible calcular el error de predicción usando cross-validation para cada sub-árbol posible
- Solución: *Cost complexity pruning (cortar las ramas mas débiles)*
  - Indexamos los arboles con T.
  - Un sub-árbol $T \in T_0$ es un árbol que se obtuvo colapsando los nodos terminales de otro árbol (cortando ramas).
  - $[T]$ = número de nodos terminales del árbol T

Pruning (poda)

- Cost complexity del árbol T

$$C_\alpha(T) = \sum_{m=1}^{[T]} n_m Q_m(T) + \alpha[T] \tag{12}$$

  - donde $Q_m(T) = \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2$ para los árboles de regresión
  - $Q_m(T)$ penaliza la heterogeneidad dentro de la regresión y α el número de regiones
  - Objetivo: para un dado α, encontrar el pruning óptimo que minimice $C_\alpha(T)$

Pruning (poda)

- Mecanismo de búsqueda para T_α (pruning óptimo dado α).

  Resultado: para cada α hay un sub-árbol único T_α que minimiza $C\alpha$ (T).

  - Eliminar sucesivamente las ramas que producen un aumento mínimo en $\sum_{m=1}^{[T]} n_m Q_m(T)$

  - Se colapsa hasta el nodo inicial pero va a través de una sucesión de árboles
  - T_α pertenece a esta secuencia. (Breiman et al., 1984)

Pruning (poda)

Algoritmo Completo

1. Utilizamos particiones recursivas binarias para hacer crecer el árbol
2. Para un dado α, aplicamos *cost complexity pruning* al árbol para obtener la secuencia de los subarboles como α.
3. Utilizamos K-fold cross-validation para elegir α.
4. Tenemos entonces una secuencia de subarboles para distintos valores de α
5. Elegimos el α y el subárbol que tienen el menor error de predicción.

Ejemplo

photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/

Agenda

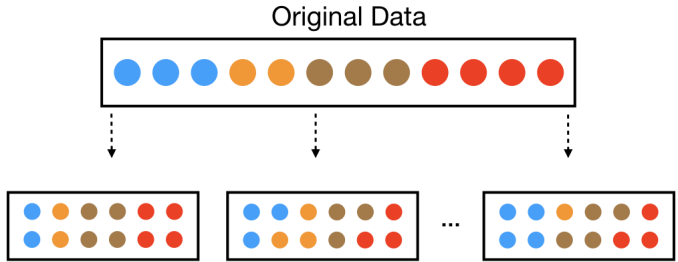
Bagging

- Problema con CART: pocos robustos.
- Idea: podemos mejorar mucho el rendimiento mediante la agregación

Bagging

- Bagging:
  - Obtenga muestras aleatorias $(X_i^b, Y_i^b)_{i=1}^N$ de la muestra observada (bootstrap).

Bagging



Bagging

- Bagging:
  - Obtenga muestras aleatorias $(X_i^b, Y_i^b)_{i=1}^N$ de la muestra observada (bootstrap).

Bagging

- Bagging:
  - Obtenga muestras aleatorias $(X_i^b, Y_i^b)_{i=1}^N$ de la muestra observada (bootstrap).
  - Para cada muestra, ajuste un árbol de regresión $\hat{f}^b(x)$
  - Promedie las muestras de bootstrap

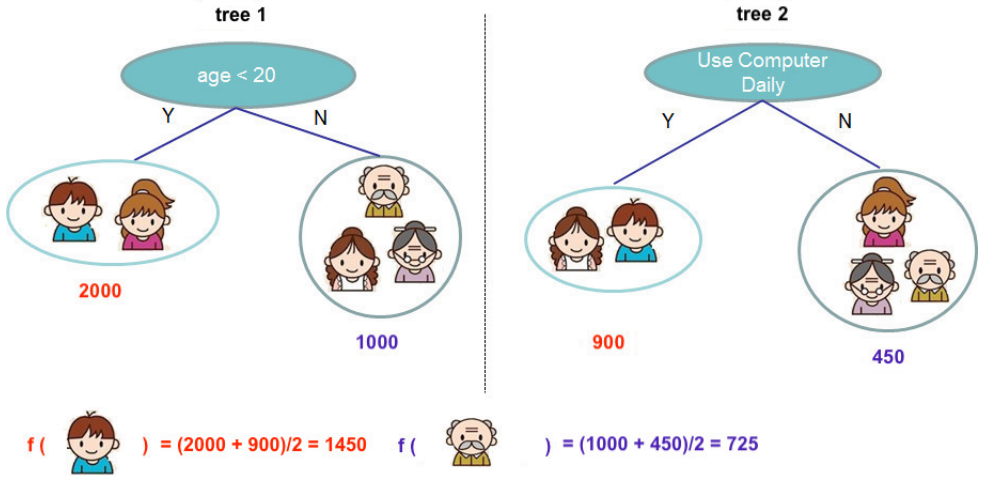
$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x) \tag{13}$$

Bagging

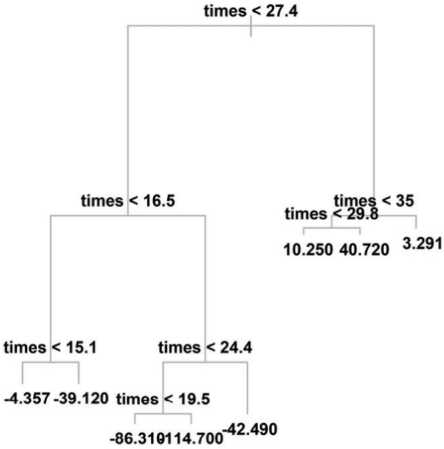
- Bagging:
  - Obtenga muestras aleatorias $(X_i^b, Y_i^b)_{i=1}^N$ de la muestra observada (bootstrap).
  - Para cada muestra, ajuste un árbol de regresión $\hat{f}^b(x)$
  - Promedie las muestras de bootstrap

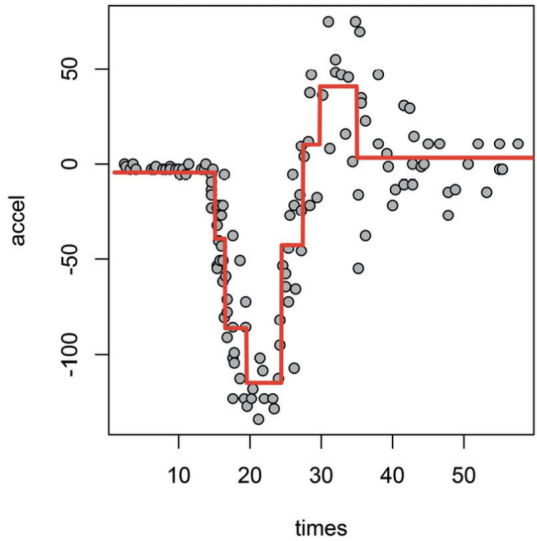
$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x) \tag{13}$$

Bagging

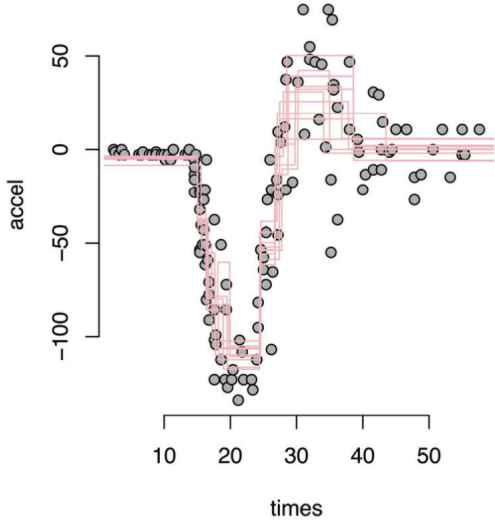

tree 1
age < 20
Y
N
2000
1000
tree 2
Use Computer Daily
Y
N
900
450
f ( ) = (2000 + 900)/2 = 1450
f ( ) = (1000 + 450)/2 = 725

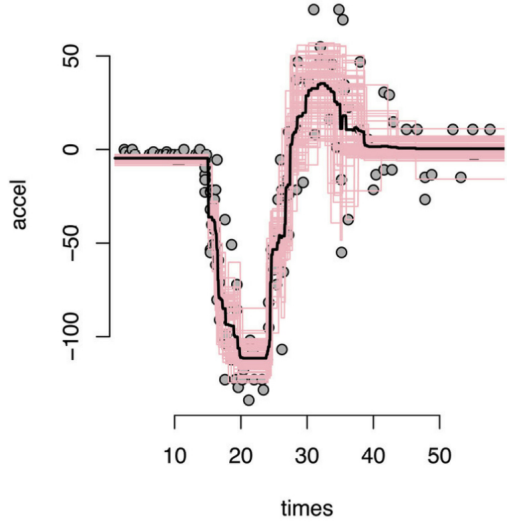
Bagging


times < 27.4
times < 16.5
times < 35
times < 29.8
10.250
40.720
3.291
times < 15.1
times < 24.4
-4.357
-39.120
times < 19.5
-86.310
-14.700
-42.490

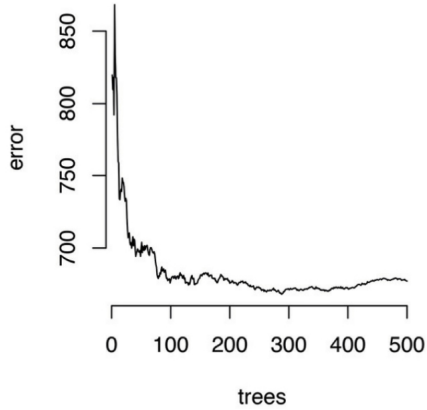

accel
50
0
-50
-100
10
20
30
40
50
times

Bagging


accel
50
0
−50
−100
10
20
30
40
50
times


accel
50
0
−50
−100
10
20
30
40
50
times

Bagging



Ejemplo

photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/

Bagging

Out-of-Bag Error: Intuición

- En Bagging, entrenamos múltiples modelos sobre muestras bootstrap.
- Cada muestra contiene $\sim$63% de los datos originales.
  - Porque: $1 - \left(1 - \frac{1}{n}\right)^n \approx 1 - e^{-1} \approx 0.63$
- El $\sim$37% restante no se usa en el entrenamiento del modelo correspondiente.
- Estos ejemplos no usados se llaman Out-of-Bag (OOB).
- ¡Sirven como un conjunto de validación natural!

Bagging

Out-of-Bag Error: Matemáticamente

Sean:

- $\{(x_i, y_i)\}_{i=1}^n$: conjunto de entrenamiento.
- $T_1, \ldots, T_B$: arboles entrenados en cada muestra bootstrap.
- $\mathcal{B}_i \subset \{1, \ldots, B\}$: índices de modelos donde x_i no fue usado (OOB).

Entonces, el error OOB es:

$$E_{OOB} = \frac{1}{n} \sum_{i=1}^{n} L \left(y_i, \frac{1}{|\mathcal{B}_i|} \sum_{b \in \mathcal{B}_i} T_b(x_i) \right)$$

- L función de pérdida (error cuadrático, $0-1$, etc.).
- Estima el **error de generalización, fuera de muestra** sin necesidad de un set de validación.

Bagging

Importancia de Variables por Permutación: Intuición

- En modelos tipo Random Forests, queremos saber:

  ¿Qué tan importante es cada variable para las predicciones?

- La idea:
  - Medimos cuánto empeora el desempeño del modelo si rompemos la relación entre una variable y el objetivo.
- ¿Cómo rompemos esa relación?
  - Permutando los valores de la variable, dejando las otras igual.
- Si la variable es importante, el error debería aumentar.

Bagging

Importancia por Permutación: Definición

- Sea E_{OOB}: error del modelo usando los datos OOB.
- Para cada variable j:
  - Permutamos la columna x_j en los datos OOB.
  - Calculamos el nuevo error: $E_{\text{perm}(j)}$.
- La importancia de la variable j es:

$$\text{VI}(j) = E_{\text{perm}(j)} - E_{\text{OOB}}$$

- Si x_j es irrelevante, el error no debería cambiar.
- Si x_j es clave, el error aumentará.

Random Forests

- Problema con el bagging: si hay un predictor dominante en la muestra de entrenamiento, diferentes árboles son muy similares entre sí.
- Bosques (forests): reduce la correlación entre los árboles en el boostrap.
- Si hay p predictores, en cada partición use solo $m < p$ predictores, elegidos al azar.
- Bagging es forests con $m = p$ (usando todo los predictores en cada partición).
- m es un hiper-parámetro, $m = \sqrt{(p)}$ es un benchmark

Ejemplo

photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/

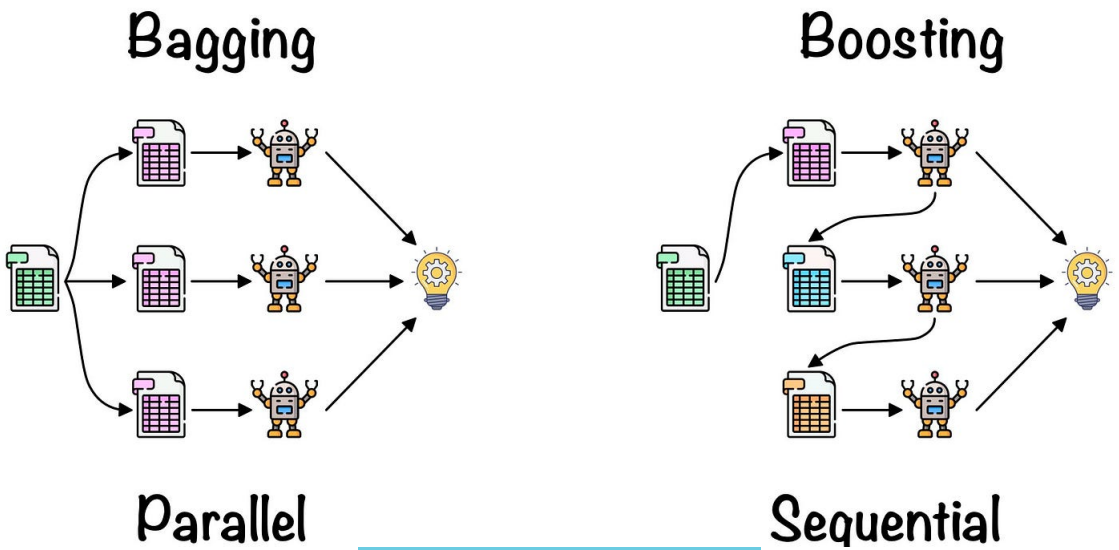
Agenda

1. Recap
2. Árboles
   - Árboles de Regresión
   - Árboles de Clasificación
   - Sobreajuste
3. Bagging and Forests
   - Bagging
   - Random Forests
4. Boosting
   - AdaBoost.M1
   - Boosting Regression Trees
   - Gradient Boosting Trees
     - XGBoost

Boosting: Motivation

- Problema con CART: varianza alta.
- Podemos mejorar mucho el rendimiento mediante la agregación
- El boosting toma esta idea pero lo ”encara” de una manera diferente $\rightarrow$ viene de la computación
- Va a usar arboles “pequeños” y “aprende” sequencialmente

Boosting: Motivation



Boosting: Motivation

- Boosting es una de las ideas de aprendizaje más poderosas introducidas en los últimos veinte años.
- Originalmente fue diseñado para problemas de clasificación,
- Pero también puede extenderse a la regresión.

Boosting: Motivation

- Intuitivamente lo que hacen es “aprender” de los errores lentamente.
- Esto lo hace “ajustando” árboles pequeños.
- Esto permite mejorar lentamente aprendiendo $f(.)$ en áreas donde no funciona bien.
- OJO: a diferencia de *bagging*, la construcción de cada árbol depende en gran medida de los árboles anteriores

AdaBoost.M1

- El AdaBoost.M1 es una implementación para clasificación

AdaBoost.M1

- El AdaBoost.M1 es una implementación para clasificación
- Como viene de la computación tiene un lenguaje ligeramente diferente.
- Vocabulario:
  - $y \in -1, 1$, X vector de predictores.
  - $\hat{y} = G(X)$ (clasificador)
  - $err = \frac{1}{N} \sum_i^N I(y_i \neq G(x_i))$

AdaBoost.M1

1. Comenzamos con ponderadores $w_i = 1/N$
2. Para m = 1 hasta M:
   1. Estimar $G_m(x)$ usando ponderadores w_i .
   2. Computar el error de predicción

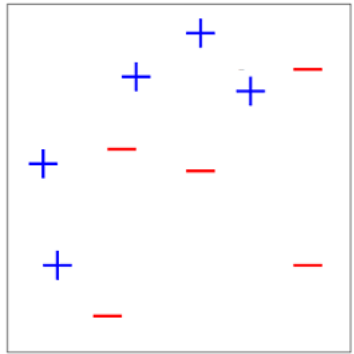
$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i} \tag{14}$$

   3. Obtener $\alpha_m = ln\left[\frac{(1-err_m)}{err_m}\right]$
   4. Actualizar los ponderadores : $w_i \leftarrow w_i c_i$

$$c_i = exp\left[\alpha_m I(yi \neq G_m(x_i))\right] \tag{15}$$

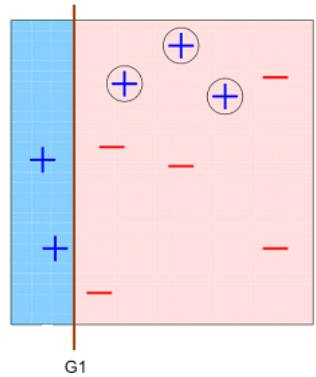
3. Resultado: $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$

AdaBoost.M1



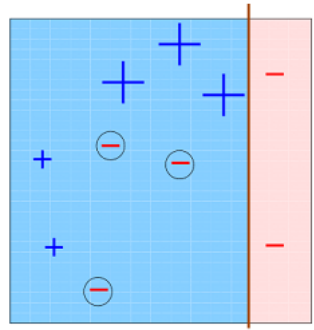
AdaBoost.M1

- Iteración 1



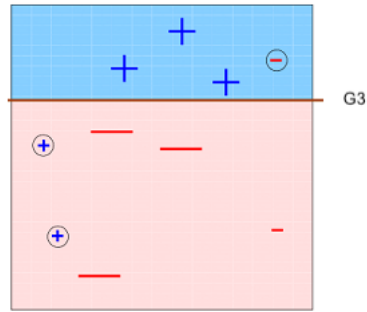
AdaBoost.M1

- Iteración 2



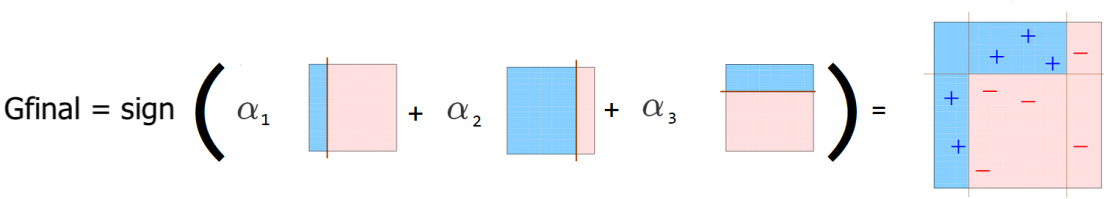
AdaBoost.M1

- Iteración 3



AdaBoost.M1

- Resultado Final



AdaBoost.M1

1. Comenzamos con ponderadores $w_i = 1/N$
2. Para m = 1 hasta M:
   1. Estimar $G_m(x)$ usando ponderadores w_i .
   2. Computar el error de predicción

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i} \tag{16}$$

   3. Obtener $\alpha_m = ln\left[\frac{(1-err_m)}{err_m}\right]$
   4. Actualizar los ponderadores : $w_i \leftarrow w_i c_i$

$$c_i = exp\left[\alpha_m I(yi \neq G_m(x_i))\right] \tag{17}$$

3. Resultado: $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$

AdaBoost.M1

- $c_i = exp\left[\alpha_m I(y_i \neq G_m(x_i))\right]$
- Si fue correctamente predicho, $c_i = 1$.
- En caso contrario, $c_i = exp(\alpha_m) = \frac{(1-err_m)}{err_m} > 1$
- En cada paso el algoritmo da mas importancia relativa a las predicciones incorrectas.
- Paso final: promedio ponderado de estos pasos

$$G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)] \tag{18}$$

Boosting Trees

Intuition

- Boosting combina muchos modelos débiles (por ejemplo, árboles pequeños), entrenados **secuencialmente**, donde cada nuevo modelo se ajusta para **corregir los errores del modelo anterior**.
  - Los modelos débiles por sí solos tienen bajo desempeño (alto sesgo).
  - Boosting reduce el error al combinar varios de estos modelos.
  - Cada nuevo modelo se entrena sobre los errores (residuos) del conjunto anterior.
  - El resultado final es un modelo fuerte con mejor capacidad predictiva.

"No empezamos de cero, ajustamos lo que ya hay."

Boosting Trees

Algoritmo

1. Iniciamos fijando $\hat{f}(x) = 0$ y $r_i = y_i$ para todos los i del training set
2. Para $m = 1, 2, ..., M$
   1. Ajustamos un árbol $\hat{f}^m(x) = 0$ con J bifurcaciones
   2. Actualizamos $\hat{f}(x)$ con una versión "shrunken" del nuevo árbol

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}^m(x) \tag{19}$$

   3. Actualizamos los residuales

$$r_i \leftarrow r_i - \hat{f}^m(x) \tag{20}$$

3. El modelo final es

$$\hat{f}_{boost} = \sum_{m=1}^{M} \hat{f}^m(x) \tag{21}$$

Boosting Trees: Hiperparámetros

- Cuantas iteraciones (M) usar?
  - Cada iteración generalmente reduce el error de ajuste, de modo que para M lo suficientemente grande este error puede hacerse arbitrariamente pequeño (sesgo se va a cero).
  - Sin embargo, ajustar demasiado bien los datos de entrenamiento puede llevar a overfit (sobreajuste)
  - Por lo tanto, hay un número óptimo $M^\star$ que minimiza el error fuera de muestra
  - Una forma conveniente de encontrar $M^\star$ con validación cruzada

Boosting Trees: Hiperparámetros

- Podemos utilizar "shrinkage"? $\Rightarrow$ Yes!
- Se escala la contribución de cada arbol por un factor $\lambda \in (0, 1)$

1. Iniciamos fijando $\hat{f}(x) = 0$ y $r_i = y_i$ para todos los i del training set
2. Para $m = 1, 2, ..., M$
   1. Ajustamos un árbol $\hat{f}^m(x) = 0$ con J bifurcaciones
   2. Actualizamos $\hat{f}(x)$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^m(x) \tag{22}$$

   3. Actualizamos los residuales

$$r_i \leftarrow r_i - \lambda \hat{f}^m(x) \tag{23}$$

3. El modelo final es

$$\hat{f}_{boost} = \sum_{m=1}^{M} \lambda \hat{f}^m(x) \tag{24}$$

Boosting Trees: Iteraciones

- Los otros hiperparámetros a fijar son
  - λ la tasa a la que aprende, los valores típicos son 0.01 o 0.001
  - El tamaño del árbol.
    - $4 \leq J \leq 8$ funciona bien
    - $J = 2$ suele ser insuficiente
    - $J > 10$ rara vez se utiliza

De Boosting a Gradient Boosting

- Boosting: modelos secuenciales que corrigen errores del anterior.
- AdaBoost: ajusta pesos a las observaciones mal clasificadas.
- ¿Y si en lugar de ajustar pesos, minimizamos directamente una función de pérdida?

Del ajuste heurístico a la optimización formal.

Gradient Boosting Trees

- El objetivo es

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \tag{25}$$

- Boosting

$$f(x_i; \{R_j, \gamma_j\}_1^J, M) = \sum_{m=1}^{M} T(x_i; \Theta_m) \tag{26}$$

- el problema es

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} T(x_i; \Theta_m)) \right\} \tag{27}$$

Gradient Boosting Trees

- El problema

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} T(x_i; \Theta_m)) \right\} \tag{28}$$

- Se puede aproximar por

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f_{m-1} + T(x_i; \Theta_m)) \right\} \tag{29}$$

- Ejemplo para una función de perdida cuadrática

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - f_{m-1} - T(x_i; \Theta_m))^2 \right\} \tag{30}$$

Gradient Boosting Trees

- For **arbitrary** loss functions this "simplified" problem

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L\left(y_i, f_{m-1} + T(x_i; \Theta_m)\right) \right\} \tag{31}$$

- is still a tricky problem
- We can solve it by using an implementation of gradient descent

$$f_m = f_{m-1} - \epsilon \sum_{i=1}^{n} \nabla_{f_{m-1}} L(y_i, f_{m-1}(x)) \tag{32}$$

- How do we implement this search?

Gradient Boosting Trees

Algoritmo

1. Inicializar:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. Para $m = 1$ hasta M:
   1. Calcular los pseudo-residuos:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   2. Ajustar un árbol de regresión a los r_{im}, obteniendo regiones terminales R_{jm}, $j = 1, \ldots, J_m$.
   3. Para cada región R_{jm}, calcular:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   4. Actualizar el modelo:

$$f_m(x) = f_{m-1}(x) + \lambda \sum_{j=1}^{J_m} \gamma_{jm} \cdot I(x \in R_{jm})$$

3. El modelo final es:

$$\hat{f}(x) = f_M(x)$$

XGBoost: Motivation

- XGBoost is an implementation was specifically engineered for optimal performance and speed.
- Why talk about XGBoost?
  - Among the 29 challenge winning solutions published in Kaggle's blog during 2015, 17 solutions used XGBoost.
  - Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. (The second most popular method, deep neural nets, was used in 11 solutions)
  - The success of the system was also witnessed in 2015 Data Mining and Knowledge Discovery competition organized by ACM (KDD Cup) , where XGBoost was used by every winning team in the top-10.
  - Historically, XGBoost has performed quite well for structured, tabular data. But, if you are dealing with non-structured data such as images, neural networks are usually a better option (more on this later)

XGBoost

- Cual es la innovación? Penaliza la optimización, en cada paso la función objetivo para construir el árbol es

$$\mathcal{L} = \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} T(x_i; \Theta_m)) + \sum_{m=1}^{M} \Omega(T(x_i; \Theta_m)) \tag{33}$$

- El segundo termino penaliza la complejidad del modelo,

$$\Omega(.) = \zeta J + \frac{1}{2} \varphi \sum_{j=1}^{J} \gamma_j^2 \tag{34}$$

Example

photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/