Boosting

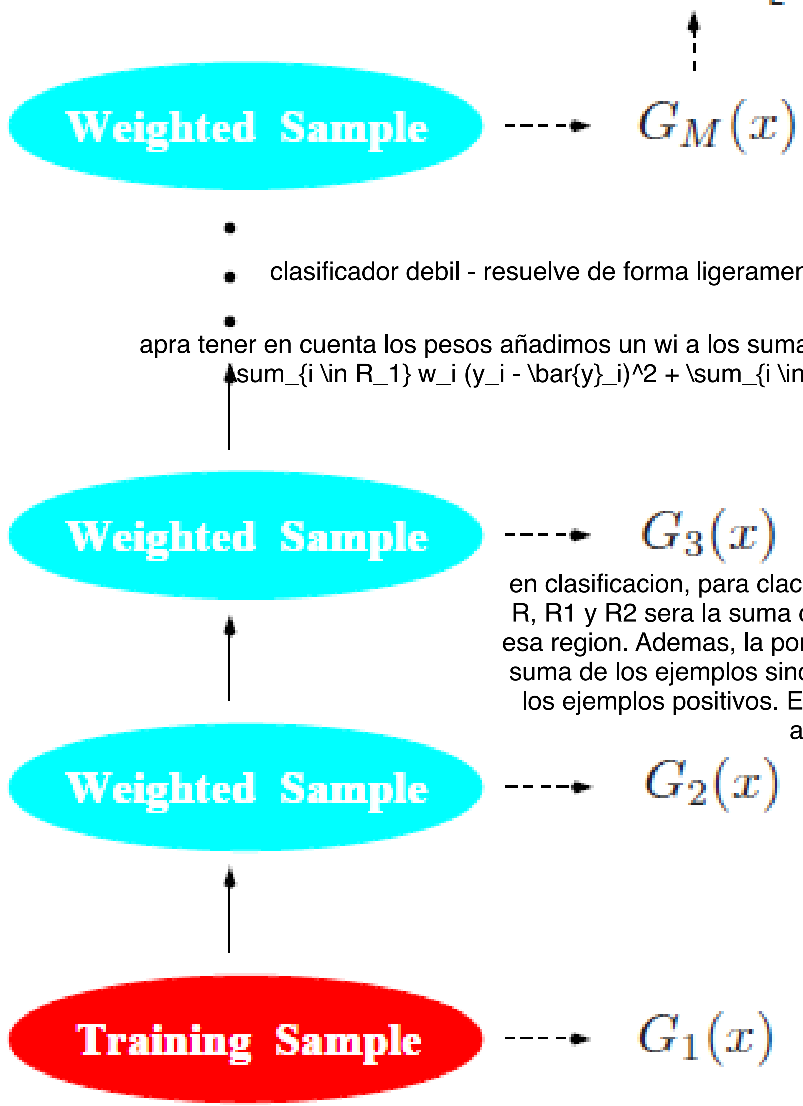
University of Santiago de Compostela
Manuel Mucientes

AdaBoost

- *Boosting idea: combine the outputs of many "weak" classifiers to produce a powerful "committee"*
- *AdaBoost.M1 (Freund and Schapire, 1997)*
  - Two-class problem: output variable in {-1, 1}

modificamos el conjutno fijandonos en los ejemplos donde fallamos la clasificacion. Aumentamos el peso a esos ejemplos, lo que fuerza al sigueinte arbol a centrarse mas en esos ejemplos (equivale a tener ese ejemplo replicado varias veces). Viendo los fallos del arbol 2 hacemos lo mismo hasta llegar al arbol final. Hiperparametro prinicpal: numero de arboles (a mas arboles, mas sobreaprendizaje).

$$G(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$



clasificador debil - resuelve de forma ligeramente superior al aleatorio (55-60)

apra tener en cuenta los pesos añadimos un wi a los sumatorios de las regionesen regresion \sum_{i \in R_1} w_i (y_i - \bar{y}_i)^2 + \sum_{i \in R_2} w_i (y_i - \bar{y}_i)^2

en clasificacion, para clacular los tamaño de las regioens, R, R1 y R2 sera la suma de los pesos de los ejemplos de esa region. Ademas, la porbabilidad positiva no ira como la suma de los ejemplos sino como la suma de los pesos de los ejemplos positivos. El algoritmo en su no cambia en absoluto

AdaBoost

Algorithm 10.1 *AdaBoost.M1.*

ej: error tendiendo a cero, el valord e alpha sera muy grande, y su logaritmo muy grande por tanto alpha muy grande (clasificador casi perfecto.
Si acierta mitad (en suma de pesos) log(0.5/0.5)=0 por lo que no contribuye ese arbol.
Si el error es cercano a 1, alpha sera muy grande y negativo hara que tengamos en cuenta lo contrario a su respuesta. MAginutd = confianza, signo = clasificacion

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M: hiperparametros principal: numero de arboles: M
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute

   cuando se equivoca en un ejemplo

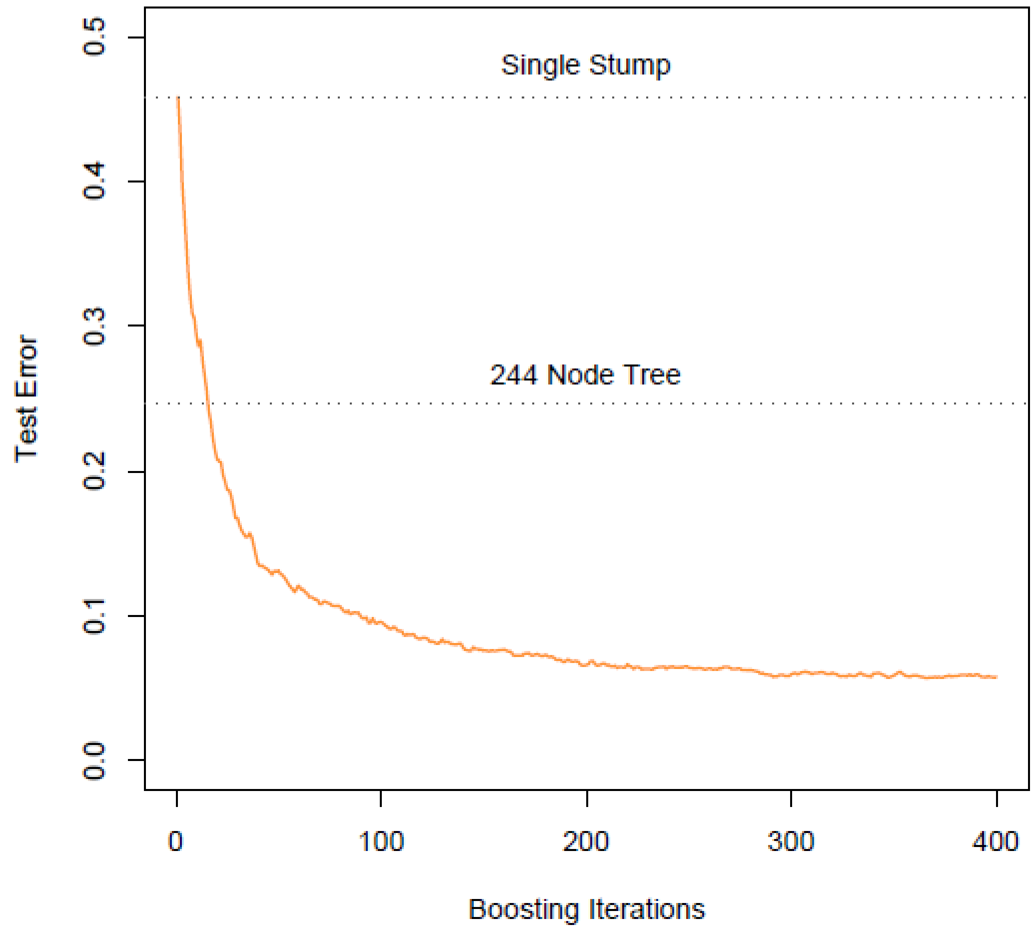
$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

AdaBoost

- Example:
  - Target: $Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5), \\ -1 & \text{otherwise.} \end{cases}$
  - Ten independent Gaussian features
  - Training: 2,000 cases
  - Test: 10,000 cases
  - Weak classifier: stump (two-terminal node tree)
    - Single stump: 45.8% test error

single stump: arbol con solo dos nodos hoja



Boosting Fits an Additive Model

- Boosting is a way of fitting an additive expansion in a set of elementary basis functions:

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

- Loss function: solo se resuelve analiticamente en casos muy sencillos. VEremos aproximaciones de esto

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

- For some *L* and *b* this is computationally expensive

Forward Stagewise Additive Modeling

- Approximates $$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$.

Forward Stagewise Additive Modeling

- For squared-error loss:

$$L(y, f(x)) = (y - f(x))^2$$

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\ &= (r_{im} - \beta b(x_i;\gamma))^2, \end{aligned}$$

  - Fit base learner to current residuals

- Adaboost: forward stagewise additive modeling with exponential loss: (Hastie, sec. 10.4)

cuando el signo de y y f(x) difieran, habra un coste alto. En el resto de caso el coste sera bajo

$$L(y, f(x)) = \exp(-y\, f(x))$$

con esta funcion de coste se obtieene el algoritmo adaboost

  - Simple algorithm for additive modeling
    - Weighted fit (samples are weighted) of the base learner

“Off-the-Shelf” Procedures for Data Mining

por que? funciona bien bagging y boosting con arboles?

importante tener calras estas propiedades para los disitntos metodos

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of “mixed” type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

cualitativas - cuantitativas

outlier en svm se convertira en un vec de soporte

missing data en arbol. Supongamos que falta un valor de x1, ese ejemplo va por ambas ramas. Si es un propblema de regresion se da la media de los nodos hoja a los que ha llegado. En clasificacion, hay un vec de prob para cada nodo hoja, sumamos todas esas prob y dividimos entre el numero de nodos hoja donde esta y lsito

segunda mas (Interpretability)

la mas importante (Predictive power)

- MARS (Multivariate Adaptive Regression Splines)

Boosting trees

- Single trees:

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \qquad \hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

- Boosted tree model: $f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)\right) \qquad (1)$$

$$\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m} \qquad \hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$$

  - Finding the regions is difficult: simplifies for special cases

Numerical Optimization via Gradient Boosting

- Fast approximate algorithms to solve (1) with any differentiable loss criterion

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)\right) \quad (1)$$

- Numerical optimization: $\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m \,, \quad \mathbf{h}_m \in \mathbb{R}^N$
  - Initial guess: $\mathbf{f}_0 = \mathbf{h}_0$
  - $\mathbf{f}_m$ induced based on $\mathbf{f}_{m-1}$

gradiente negativo (hay que moverse en la direccion contraria)

- Calculate the step ($\mathbf{h}_m$) with steepest descent: $\mathbf{h}_m = -\rho_m \, \mathbf{g}_m$

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \qquad \rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho\mathbf{g}_m)$$

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

Gradient Boosting

- Gradient only defined on training points: need to generalize
- Induce tree with predictions close to negative gradient:

ahora la salida es el -gradiente del ejemplo M, no la salida como en el caso de regrsiosn de arboles

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \left(-g_{im} - T(x_i; \Theta)\right)^2$$

  - Squared error to build tree
  - Approximation to: $$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$
    - Regions not equal but similar
  - Output for each region:

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

- For squared error loss: $-g_{im} = y_i - f_{m-1}(x_i)$
  - Equivalent to least squares tree boosting

Gradient Boosting for Regression

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

- Tree size: restrict all trees to be the same size
  - Cross-validation to select *J* seldom improves over *J*=6 (Hastie et al., 2009, p. 363)
  - In real problems larger *J* might be necessary

- Optimal number of trees (*M*): validation sample

- Shrinkage: another way to regularize
  - Scale the contribution of each tree: line 2(d) of gradient boosting

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

Gradient Boosting

- Subsampling: Stochastic gradient boosting (Friedman, 1999)
  - At each iteration sample a fraction (η) of the training set without replacement

- Four hyper-parameters: *J*, *M*, ν, η
  - Determine suitable values for *J*, $\nu\,(< 0.1)$, η (0.5)
  - Pick *M* through validation

Gradient Boosting

- For multinomial deviance loss (logistic loss) or cross-entropy loss:
  - *K* least squares trees per iteration (1 for *K*=2)

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$$

$$\begin{aligned} L(y, p(x)) &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) \log p_k(x) \\ &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) f_k(x) + \log\left(\sum_{\ell=1}^{K} e^{f_\ell(x)}\right) \end{aligned}$$

$$\begin{aligned} -g_{ikm} &= - \frac{\partial L\left(y_i, f_{1m}(x_i), \ldots, f_{1m}(x_i)\right)}{\partial f_{km}(x_i)} \\ &= I(y_i = \mathcal{G}_k) - p_k(x_i), \end{aligned}$$

Gradient Boosting

Algorithm 10.4 *Gradient Boosting for K-class Classification.*

1. Initialize $f_{k0}(x) = 0,\ k = 1, 2, \ldots, K$.
2. For m=1 to M:
   (a) Set
   $$p_k(x) = \frac{e^{f_k(x)}}{\sum_{\ell=1}^{K} e^{f_\ell(x)}},\ k = 1, 2, \ldots, K.$$
   (b) For $k = 1$ to K:
      i. Compute $r_{ikm} = y_{ik} - p_k(x_i),\ i = 1, 2, \ldots, N$.
      ii. Fit a regression tree to the targets $r_{ikm},\ i = 1, 2, \ldots, N$, giving terminal regions $R_{jkm},\ j = 1, 2, \ldots, J_m$.
      iii. Compute
      $$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1 - |r_{ikm}|)},\ j = 1, 2, \ldots, J_m.$$
      iv. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$.
3. Output $\hat{f}_k(x) = f_{kM}(x),\ k = 1, 2, \ldots, K$.

Variable importance of additive trees

- Contribution of each input variable in predicting the response

- For a single tree (Breiman et al., 1984):

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 \, I(v(t) = \ell)$$

  - J-1: number of internal nodes
  - $\hat{\imath}_t^2$: improvement of RSS (regression), Gini index or cross-entropy (classification)

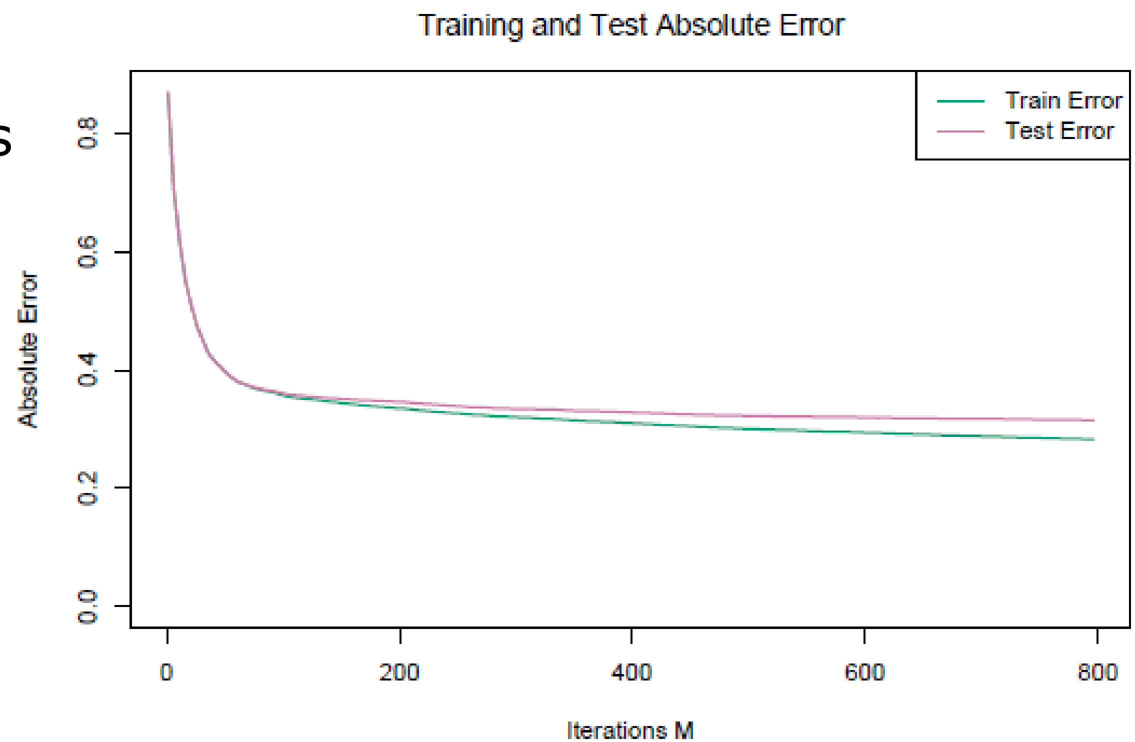
- For additive trees: $$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m)$$

  - More reliable than for a single tree

Example: California Housing

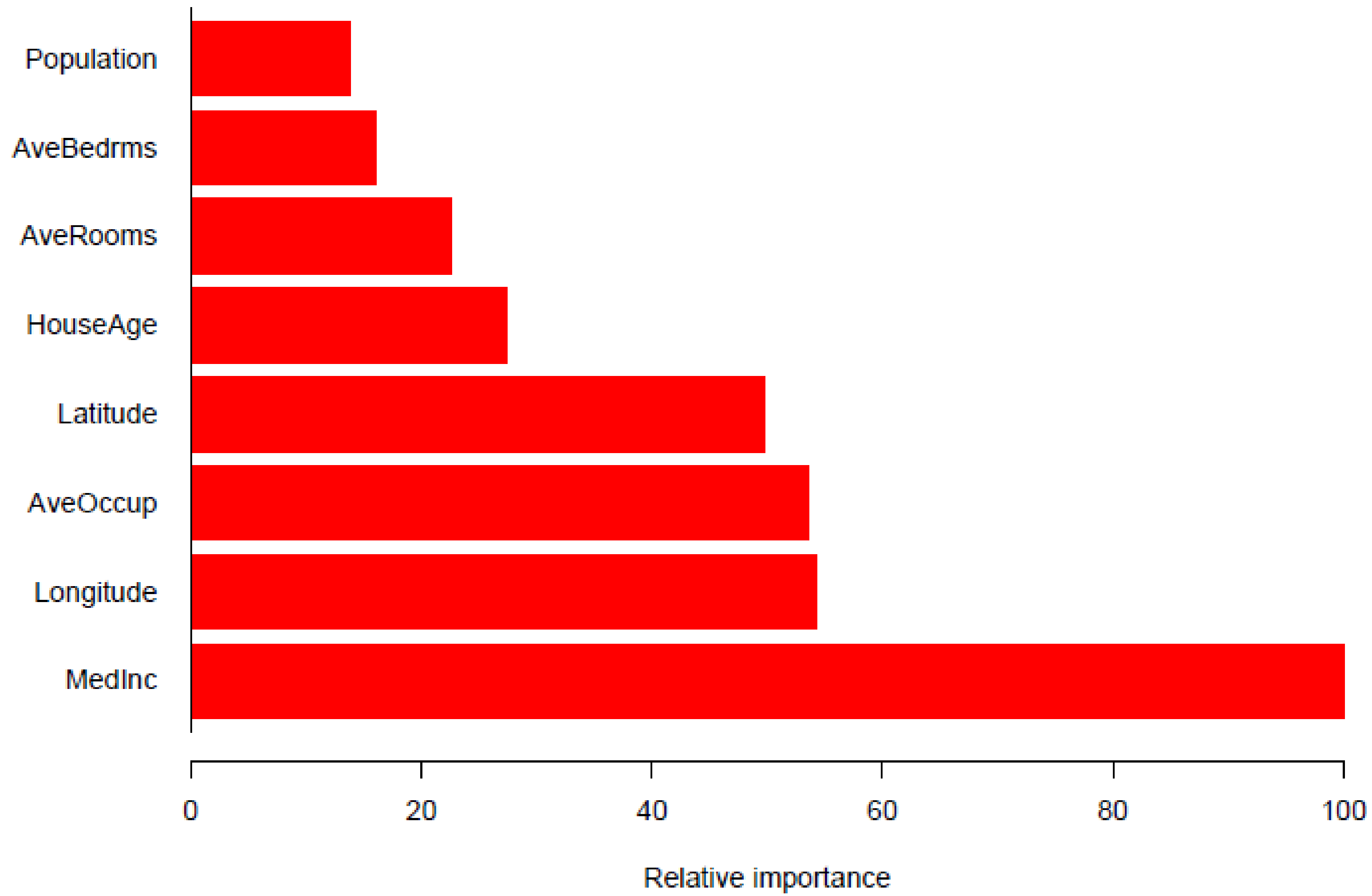
- Pace and Barry, 1997. StatLib repository
- 20,460 neighborhoods in California: 80% training, 20% test
- Response variable: median house value in each neighborhood in units of $100,000
- Eight numerical predictors: median income (MedInc), etc.
- Gradient boosting with

  $J=6$, $\nu=0.1$, Huber loss



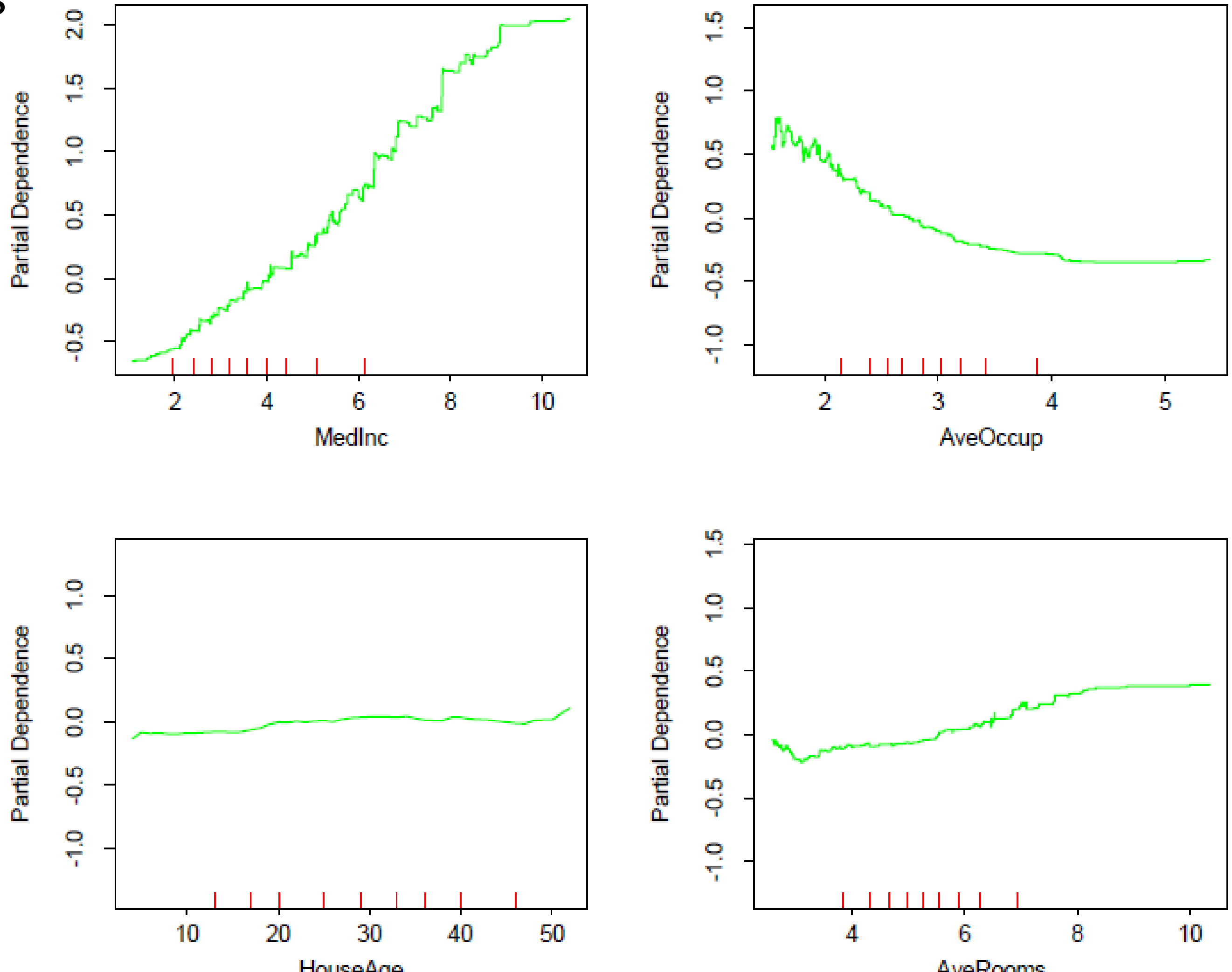
Example: California Housing (ii)

- Variable importance



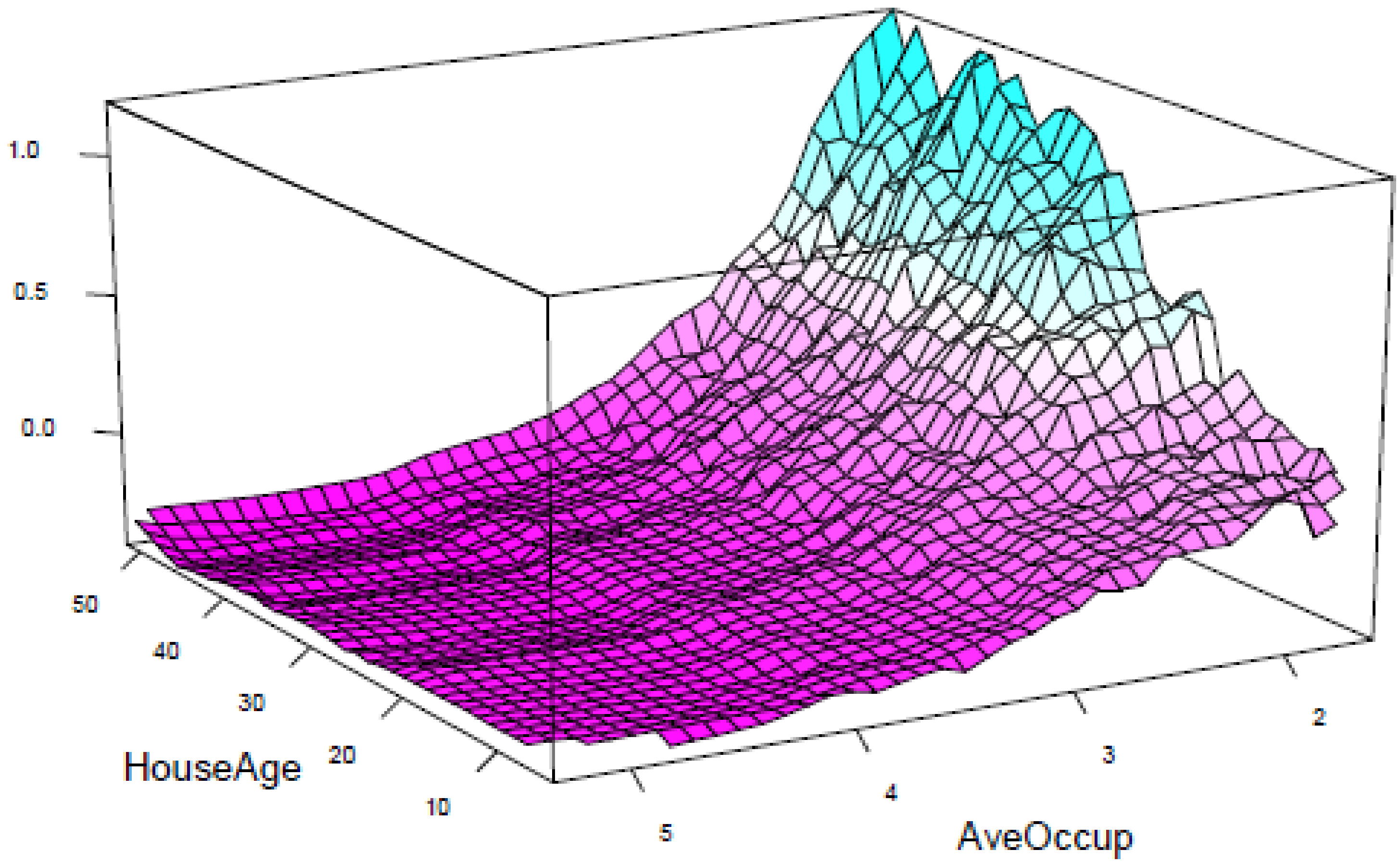
Example: California Housing (iii)

- Partial dependence plots (one variable)
  - Effect of a variable taking into account the (average) effects of the other variables



Example: California Housing (iv)

- Partial dependence plot (two variables)



Bibliography

- T. Hastie, R. Tibshirani, y J. Friedman, The elements of statistical learning. Springer, 2009.
  - Chapter 10

- G. James, D. Witten, T. Hastie, y R. Tibshirani, An Introduction to Statistical Learning with Applications in R. Springer, 2021.
  - Chapter 8, Sec. 8.2.3