

- finish talking about random forests (choice of m)
- boosting
- PPR
- Neural Networks

Choice of $m \rightarrow$ cross-validation
(bootstrapping, 0.632 + bootstrapping)

$\searrow$ default: $\lfloor P/3 \rfloor$ regression
 $\lfloor \sqrt{P} \rfloor$ classification

in the example

$m = \sqrt{P}$ default

$(2, 5) \rightarrow p = 7 \quad m = \lfloor \sqrt{7} \rfloor = 2$

$$1 - P_M(0) = 1 - \frac{\binom{2}{0} \binom{5}{2}}{\binom{7}{2}} \approx 0.52$$

Gradient Boosting

• forward stagewise additive model

$$y = f(x) + \varepsilon$$

1) initialize $f(x) = 0$ tuning parameter: number of boosting steps

2) for $m = 1, \dots, M$

(a) find the direction in which I can improve my model the most

$$u_m = - \frac{\partial \mathcal{L}(y, f(x))}{\partial f(x)}$$

negative gradient

model (linear model, tree, splines)

(b) fit a base learner on this direction

$$h_m(u_m, x) \leftarrow \begin{matrix} \text{spline} \\ \text{tree} \\ \text{lin. regr.} \end{matrix}$$

$$X\hat{\beta} = X(X^T X)^{-1} X^T u_m$$

(c) update the model $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \eta h_m(u_m, x)$

tuning parameter:
boosting step size

3) output $\hat{f}(x) = \hat{f}_M(x)$

The step 2b can be performed globally (using all x) or componentwise (x_j by x_j). In the latter case, we need a further step to decide which x_j I want to use
 $\rightarrow$ component-wise gradient boosting

Boosting relies on the choice of the number of ^{iterations} boosting steps

→ larger M , larger complexity (lower bias, higher variance)

- componentwise boosting: including more variables into the model
 - ↳ including not relevant variables if M is too large

↳ overfit if M is too large

→ smaller M , lower complexity (higher bias, lower variance)

- componentwise boosting: including less variables into the model
 - ↳ losing important variables if M is too small

↳ underfit if M is too small

→ M is found by cross-validation (0.632+ bootstrapping)

Other parameter λ (boosting step size)

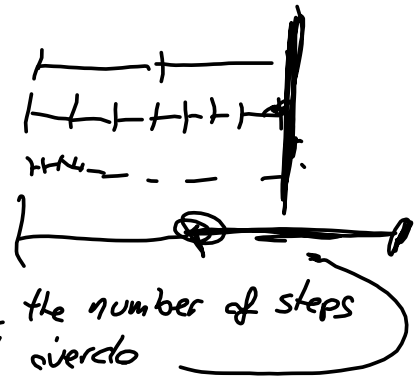
normally it is fixed to 0.1
 it's connected to that of the number of steps:
 optimal value

larger λ , less boosting steps to reach the best performance

smaller λ , more boosting steps

So, the tactic is to fix the step size and tune the number of steps

IMPORTANT: it should be small enough, to not overdo
 better too small than too big!



Projection pursuit regression

→ extension (G)AM

$$f(x_1, \dots, x_p) = \sum_{m=1}^M g_m(w_m^T X)$$

p-dim.

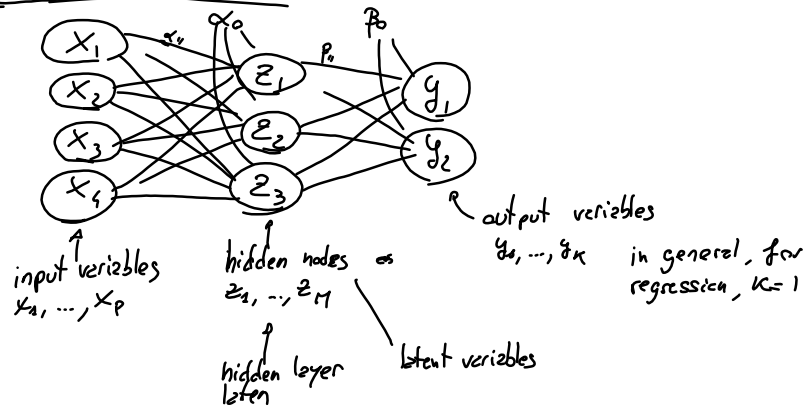
where w_m , $m=1, \dots, M$ are p-dimensional vectors

- the derived features $V_m = w_m^T X$ are projection of X on the unit vectors of w_m , chosen in a way that the model fits well (pursuit)
- w_m are the projection vectors (must be estimated)
- $g_m(\cdot)$ are called ridge functions (are constant ~~over~~ in all dimensions of X but those defined by w_m)
- M is the number of projection (computed by cross-validation) ^{may be}
- in general, $M \neq p$ (in contrast to additive models)

IMPORTANT: for M large enough and an appropriate choice of g_m , the PPR model can approximate arbitrarily well any continuous function in $\mathbb{R}^p$

↓
universal approximator

Neural Networks (just an introduction)



We want to relate our p covariates X to the outcomes $y_1, \dots, y_k$ through M latent variables Z + some constants

A neural network is essentially a two-stage regression scheme

$$z_m = \sigma \left(\alpha_{0m} + \sum_{j=1}^p \alpha_{jm} x_j \right) \quad m=1, \dots, M$$

$$y_\ell = g \left(\beta_{0\ell} + \sum_{m=1}^M \beta_{m\ell} z_m \right) \quad \ell=1, \dots, k$$

where $\alpha_{0m}, \alpha_{jm}, \beta_{0\ell}, \beta_{m\ell}$, $j=1, \dots, p$, $m=1, \dots, M$, $\ell=1, \dots, k$ are parameters (called weights) to be estimated

$$M \quad M \times p \quad k \quad k \times M \rightarrow M(p+1) + k(M+1) \text{ parameters to estimate}$$

$\sigma(\cdot)$ is the activation function, and g is a function that links the hidden variables to the response

g for regression: $g(x) = x$ identity function

g for classification: soft max

$\sigma(\cdot)$ is a non-linear function, typical choices are:

Sigmoid: $\sigma(x) = \frac{1}{1 + e^{-x}}$

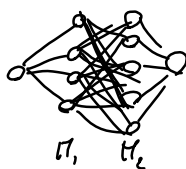
ReLU: $\sigma(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$

tanh: $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

ReLU: Rectified Linear Unit

if $\sigma(x) = x$, then we go back to linear regression model

We need to define the number of hidden variables M (and, in the case of deep-learning, the number of hidden layers)



architecture

not easy solution

try and fail

To fit the NN means estimating the weights

$$\min R(\theta) = \sum_{\ell=1}^k \sum_{i=1}^n (y_{i\ell} - f_\ell(x_i))^2$$

where $f_\ell(x_i) = g_\ell \left(\beta_{0\ell} + \sum_{m=1}^M \beta_{m\ell} \sigma \left(\alpha_{0j} + \sum_{j=1}^p \alpha_{jm} x_j \right) \right)$ $\ell=1, \dots, k$