Big Data and Machine Learning with an Actuarial Perspective

A. Charpentier (UQAM & Université de Rennes 1)

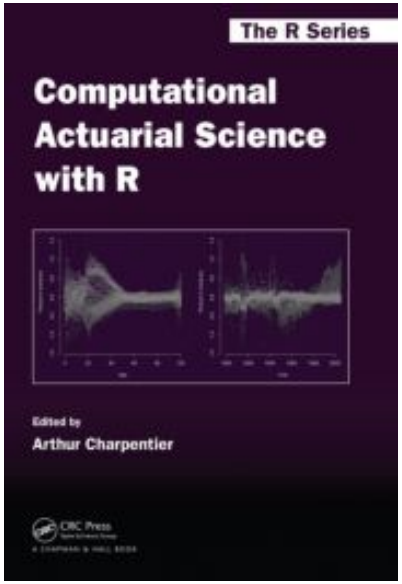
IA | BE Summer School, Louvain-la-Neuve, September 2015.

http://freakonometrics.hypotheses.org

A Brief Introduction to Machine Learning and Data Science for Actuaries

A. Charpentier (UQAM & Université de Rennes 1)

Professor of Actuarial Sciences, Mathematics Department, UQàM
(previously Economics Department, Univ. Rennes 1 & ENSAE Paristech
actuary in Hong Kong, IT & Stats FFSA)
PhD in Statistics (KU Leuven), Fellow Institute of Actuaries
MSc in Financial Mathematics (Paris Dauphine) & ENSAE
Editor of the freakonometrics.hypotheses.org's blog
Editor of Computational Actuarial Science, CRC



Agenda

1. Introduction to Statistical Learning
2. Classification $y_i \in \{0, 1\}$, or $y_i \in \{\bullet, \bullet\}$
3. Regression $y_i \in \mathbb{R}$ (possibly $y_i \in \mathbb{N}$)
4. Model selection, feature engineering, etc

All those topics are related to computational issues, so R codes will be mentioned

Inside Black boxes

The goal of the course is to describe philosophical difference between machine learning techniques, and standard statistical / econometric ones, to describe algorithms used in machine learning, but also to see them in action.
A machine learning technique is

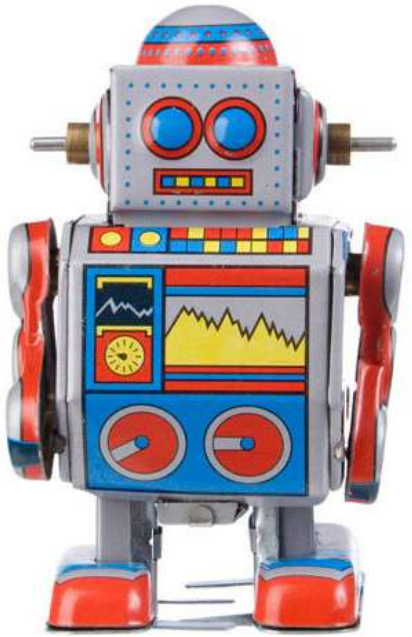
- an algorithm
- a code (implementation of the algorithm)

Prose and Verse (Spoiler)

MAÎTRE DE PHILOSOPHIE: Sans doute. Sont-ce des vers que vous lui voulez écrire?
MONSIEUR JOURDAIN: Non, non, point de vers.
MAÎTRE DE PHILOSOPHIE: Vous ne voulez que de la prose?
MONSIEUR JOURDAIN: Non, je ne veux ni prose ni vers.
MAÎTRE DE PHILOSOPHIE: Il faut bien que ce soit l'un, ou l'autre.
MONSIEUR JOURDAIN: Pourquoi?
MAÎTRE DE PHILOSOPHIE: Par la raison, Monsieur, qu'il n'y a pour s'exprimer que la prose, ou les vers.
MONSIEUR JOURDAIN: Il n'y a que la prose ou les vers?
MAÎTRE DE PHILOSOPHIE: Non, Monsieur: tout ce qui n'est point prose est vers; et tout ce qui n'est point vers est prose.
MONSIEUR JOURDAIN: Et comme l'on parle qu'est-ce que c'est donc que cela?
MAÎTRE DE PHILOSOPHIE: De la prose.
MONSIEUR JOURDAIN: Quoi? quand je dis: "Nicole, apportez-moi mes pantoufles, et me donnez mon bonnet de nuit" , c'est de la prose?
MAÎTRE DE PHILOSOPHIE: Oui, Monsieur.
MONSIEUR JOURDAIN: Par ma foi! il y a plus de quarante ans que je dis de la prose sans que j'en susse rien, et je vous suis le plus obligé du monde de m'avoir appris cela. Je voudrais donc lui mettre dans un billet: Belle Marquise, vos beaux yeux me font mourir d'amour; mais je voudrais que cela fût mis d'une manière galante, que cela fût tourné gentiment.

'*Le Bourgeois Gentilhomme* ', Molière (1670)

Part 1.
Statistical/Machine Learning



Statistical Learning and Philosophical Issues

From *Machine Learning and Econometrics*, by Hal Varian :

"**Machine learning** use data to predict some variable as a function of other covariables,

- may, or may not, care about insight, importance, patterns
- may, or may not, care about inference (how y changes as some $\boldsymbol{x}$ change)

Econometrics use statistical methodes for prediction, inference and causal modeling of economic relationships

- hope for some sort of insight (inference is a goal)
- in particular, causal inference is goal for decision making."

$\rightarrow$ machine learning, 'new tricks for econometrics'

Statistical Learning and Philosophical Issues

Remark machine learning can also learn from econometrics, especially with non i.i.d. data (time series and panel data)

Remark machine learning can help to get better predictive models, given good datasets. No use on several data science issues (e.g. selection bias).

Statistical Learning and Philosophical Issues

"*Ceteris Paribus*: causal effect with other things being held constant; partial derivative

Mutatis mutandis: correlation effect with other things changing as they will; total derivative

Passive observation: If I observe price change of dx_j, how do I expect quantity sold y to change?

Explicit manipulation: If I explicitly change price by dx_j, how do I expect quantity sold y to change?"

Non-Supervised and Supervised Techniques

Just $\boldsymbol{x}_i$'s, here, no y_i: unsupervised.

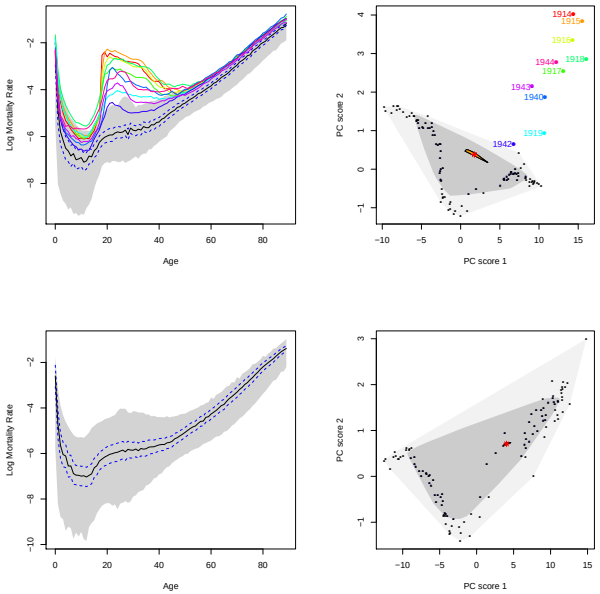
Use principal components to reduce dimension: we want d vectors $\boldsymbol{z}_1, \cdots, \boldsymbol{z}_d$ such that

$$\boldsymbol{x}_i \sim \sum_{j=1}^{d} \omega_{i,j} \boldsymbol{z}_j \text{ or } \boldsymbol{X} \sim \boldsymbol{Z}\boldsymbol{\Omega}^\mathsf{T}$$

where $\boldsymbol{\Omega}$ is a $k \times d$ matrix, with $d < k$.

First Compoment is $\boldsymbol{z}_1 = \boldsymbol{X}\boldsymbol{\omega}_1$ where

$$\boldsymbol{\omega}_1 = \underset{\|\boldsymbol{\omega}\|=1}{\text{argmax}} \left\{ \|\boldsymbol{X} \cdot \boldsymbol{\omega}\|^2 \right\} = \underset{\|\boldsymbol{\omega}\|=1}{\text{argmax}} \left\{ \boldsymbol{\omega}^\mathsf{T} \boldsymbol{X}^\mathsf{T} \boldsymbol{X} \boldsymbol{\omega} \right\}$$



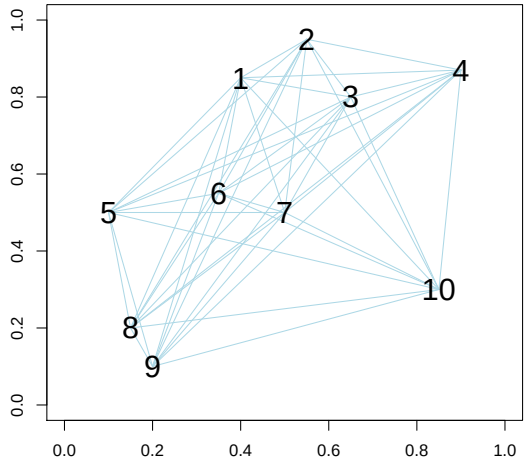
Second Compoment is $\boldsymbol{z}_2 = \boldsymbol{X}\boldsymbol{\omega}_2$ where

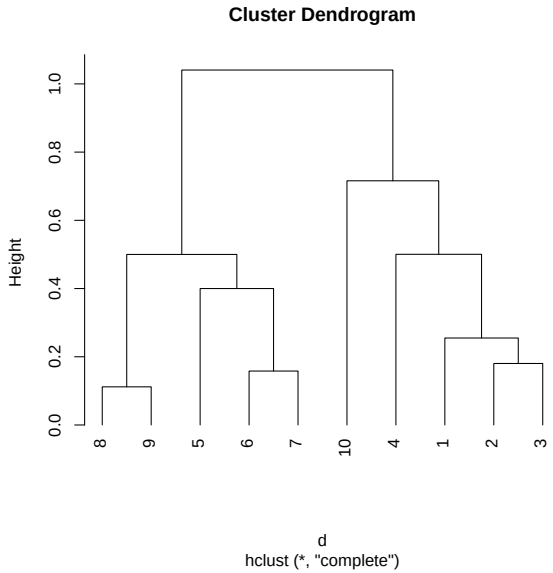
$$\boldsymbol{\omega}_2 = \underset{\|\boldsymbol{\omega}\|=1}{\text{argmax}} \left\{ \|\widetilde{\boldsymbol{X}}^{(1)} \cdot \boldsymbol{\omega}\|^2 \right\} \text{ where } \widetilde{\boldsymbol{X}}^{(1)} = \boldsymbol{X} - \underbrace{\boldsymbol{X}\boldsymbol{\omega}_1}_{\boldsymbol{z}_1} \boldsymbol{\omega}_1^\mathsf{T}$$

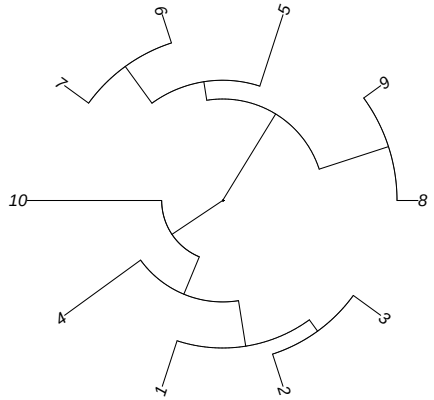
Non-Supervised and Supervised Techniques

... etc, see Galton (1889) or MacDonell (1902).

k-means and hierarchical clustering can be used to get clusters of the n observations.







Datamining, Explantory Analysis, Regression, Statistical Learning, Predictive Modeling, etc

In statistical learning, data are approched with little priori information.

In regression analysis, see Cook & Weisberg (1999)

> The primary goal in a regression analysis is to understand, as far as possible with the available data, how the conditional distribution of the response y varies across subpopulations determined by the possible values of the predictor or predictors. Since this is the central idea, it will be helpful to have a conve-

i.e. we would like to get the distribution of the response variable Y conditioning on one (or more) predictors $\boldsymbol{X}$.

Consider a regression model, $y_i = m(\boldsymbol{x_i}) + \varepsilon_i$, where ε_i 's are i.i.d. $\mathcal{N}(0, \sigma^2)$, possibly linear $y_i = \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta} + \varepsilon_i$, where ε_i's are (somehow) unpredictible.

Machine Learning and 'Statistics'

Machine learning and statistics seem to be very similar, they share the same goals—they both focus on data modeling—but their methods are affected by their cultural differences.

"The goal for a statistician is to predict an interaction between variables with some degree of certainty (we are never 100% certain about anything). Machine learners, on the other hand, want to build algorithms that predict, classify, and cluster with the most accuracy, see Why a Mathematician, Statistician & Machine Learner Solve the Same Problem Differently

Machine learning methods are about algorithms, more than about asymptotic statistical properties.

Machine Learning and 'Statistics'

See also nonparametric inference: "Note that the non-parametric model is not none-parametric: parameters are determined by the training data, not the model. [...] non-parametric covers techniques that do not assume that the structure of a model is fixed. Typically, the model grows in size to accommodate the complexity of the data." see wikipedia

Validation is not based on mathematical properties, but on properties out of sample: we must use a training sample to train (estimate) model, and a testing sample to compare algorithms.

Goldilock Principle: the Mean-Variance Tradeoff

In statistics and in machine learning, there will be parameters and meta-parameters (or tunning parameters. The first ones are estimated, the second ones should be chosen.

See Hill estimator in extreme value theory. X has a Pareto distribution above some threshold u if

$$\mathbb{P}[X > x | X > u] = \left(\frac{u}{x}\right)^{\frac{1}{\xi}} \text{ for } x > u.$$

Given a sample $\boldsymbol{x}$, consider the Pareto-QQ plot, i.e. the scatterplot

$$\left\{ -\log\left(1 - \frac{i}{n+1}\right), \log x_{i:n} \right\}_{i=n-k,\cdots,n}$$

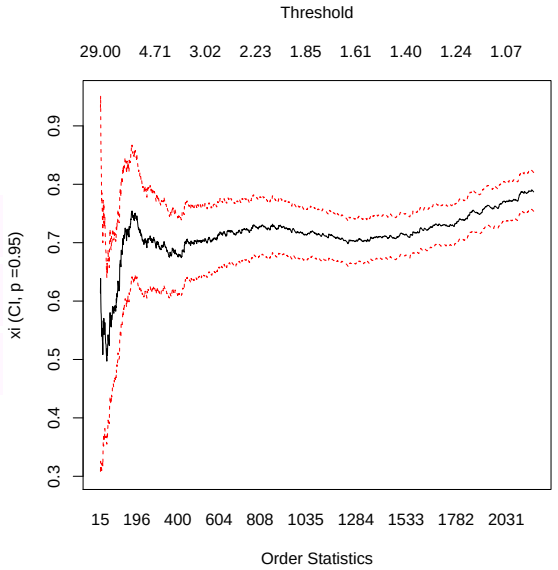
for points exceeding $X_{n-k:n}$. The slope is ξ, i.e.

$$\log X_{n-i+1:n} \approx \log X_{n-k:n} + \xi\left(-\log\frac{i}{n+1} - \log\frac{n+1}{k+1}\right)$$

Goldilock Principle: the Mean-Variance Tradeoff

Hence, consider estimator $\widehat{\xi}_k = \dfrac{1}{k}\displaystyle\sum_{i=0}^{k-1} \log x_{n-i:n} - \log x_{n-k:n}$.

```
> library(evir)
> data(danish)
> hill(danish, "xi")
```



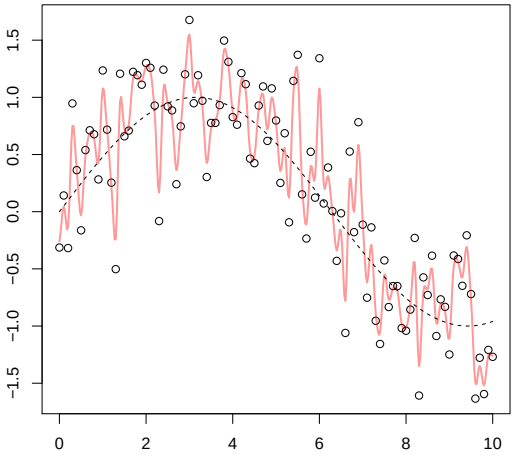
Standard mean-variance tradeoff,

- k large: bias too large, variance too small
- k small: variance too large, bias too small

Goldilock Principle: the Mean-Variance Tradeoff

Same holds in kernel regression, with bandwidth h (length of neighborhood)

```
> library(np)
> nw <- npreg(y ~ x, data=db, bws=h,
+ ckertype= "gaussian")
```

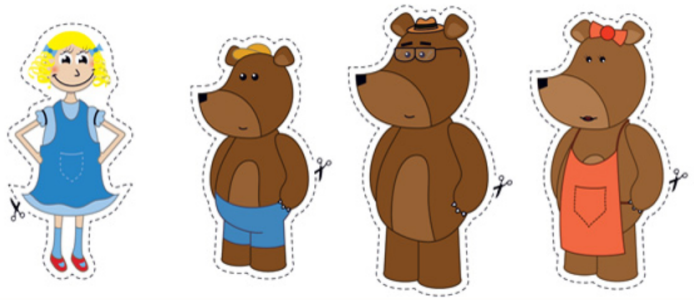


Standard mean-variance tradeoff,

- h large: bias too large, variance too small
- h small: variance too large, bias too small

Goldilock Principle: the Mean-Variance Tradeoff

More generally, we estimate $\widehat{\boldsymbol{\theta}}_h$ or $\widehat{m}_h(\cdot)$
Use the mean squared error for $\widehat{\boldsymbol{\theta}}_h$



$$\mathbb{E}\left[\left(\boldsymbol{\theta}-\widehat{\boldsymbol{\theta}}_h\right)^2\right]$$

or mean integrated squared error $\widehat{m}_h(\cdot)$,

$$\mathbb{E}\left[\int\left(m(\boldsymbol{x})-\widehat{m}_h(\boldsymbol{x})\right)^2 d\boldsymbol{x}\right]$$

In statistics, derive an asymptotic expression for these quantities, and find $h^\star$ that minimizes those.

Goldilock Principle: the Mean-Variance Tradeoff

For kernel regression, the MISE can be approximated by

$$\frac{h^4}{4}\left(\int x^2K(\boldsymbol{x})d\boldsymbol{x}\right)^2\int\left(m''(\boldsymbol{x})+2m'(\boldsymbol{x})\frac{f'(\boldsymbol{x})}{f(\boldsymbol{x})}\right)d\boldsymbol{x}+\frac{1}{nh}\sigma^2\int K^2(x)dx\int\frac{d\boldsymbol{x}}{f(\boldsymbol{x})}$$

where f is the density of $\boldsymbol{x}$'s. Thus the optimal h is

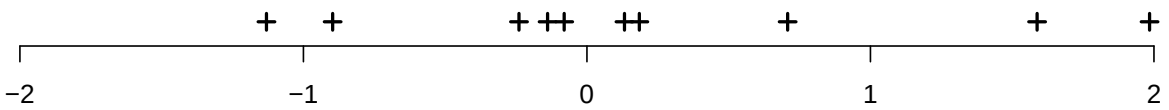
$$h^\star=n^{-\frac{1}{5}}\left(\frac{\sigma^2\int K^2(x)dx\int\frac{d\boldsymbol{x}}{f(\boldsymbol{x})}}{\left(\int x^2K(\boldsymbol{x})d\boldsymbol{x}\right)^2\int\left(\int m''(\boldsymbol{x})+2m'(\boldsymbol{x})\frac{f'(\boldsymbol{x})}{f(\boldsymbol{x})}\right)^2d\boldsymbol{x}}\right)^{\frac{1}{5}}$$

(hard to get a simple rule of thumb... up to a constant, $h^\star\sim n^{-\frac{1}{5}}$)

Use bootstrap, or cross-validation to get an optimal h

Randomization is too important to be left to chance!

Bootstrap (resampling) algorithm is very important (nonparametric monte carlo)



→ data (and not model) driven algorithm

Randomization is too important to be left to chance!

Consider some sample $\boldsymbol{x} = (x_1, \cdots, x_n)$ and some statistics $\widehat{\theta}$. Set $\widehat{\theta}_n = \widehat{\theta}(\boldsymbol{x})$

Jackknife used to reduce bias: set $\widehat{\theta}_{(-i)} = \widehat{\theta}(\boldsymbol{x}_{(-i)})$, and $\tilde{\theta} = \dfrac{1}{n}\sum_{i=1}^{n} \widehat{\theta}_{(-i)}$

$$\text{If } \mathbb{E}(\widehat{\theta}_n) = \theta + O(n^{-1}) \text{ then } \mathbb{E}(\tilde{\theta}_n) = \theta + O(n^{-2}).$$

See also leave-one-out cross validation, for $\widehat{m}(\cdot)$

$$\text{mse} = \frac{1}{n}\sum_{i=1}^{n}[y_i - \widehat{m}_{(-i)}(x_i)]^2$$

Boostrap estimate is based on bootstrap samples: set $\widehat{\theta}_{(b)} = \widehat{\theta}(\boldsymbol{x}_{(b)})$, and $\tilde{\theta} = \dfrac{1}{n}\sum_{i=1}^{n} \widehat{\theta}_{(b)}$, where $\boldsymbol{x}_{(b)}$ is a vector of size n, where values are drawn from $\{x_1, \cdots, x_n\}$, with replacement. And then use the law of large numbers...

See Efron (1979).

Statistical Learning and Philosophical Issues

From $(y_i, \boldsymbol{x}_i)$, there are different stories behind, see Freedman (2005)

- the causal story : $x_{j,i}$ is usually considered as independent of the other covariates $x_{k,i}$. For all possible $\boldsymbol{x}$, that value is mapped to $m(\boldsymbol{x})$ and a noise is atatched, ε. The goal is to recover $m(\cdot)$, and the residuals are just the difference between the response value and $m(\boldsymbol{x})$.
- the conditional distribution story : for a linear model, we usually say that Y given $\boldsymbol{X} = \boldsymbol{x}$ is a $\mathcal{N}(m(\boldsymbol{x}), \sigma^2)$ distribution. $m(\boldsymbol{x})$ is then the conditional mean. Here $m(\cdot)$ is assumed to really exist, but no causal assumption is made, only a conditional one.
- the explanatory data story : there is no model, just data. We simply want to summarize information contained in $\boldsymbol{x}$'s to get an accurate summary, close to the response (i.e. $\min\{\ell(\boldsymbol{y}_i, m(\boldsymbol{x}_i))\}$) for some loss function ℓ.

Machine Learning vs. Statistical Modeling

In **machine learning**, given some dataset $(\boldsymbol{x}_i, y_i)$, solve

$$\widehat{m}(\cdot) = \underset{m(\cdot)\in\mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \ell(y_i, m(\boldsymbol{x}_i)) \right\}$$

for some loss functions $\ell(\cdot,\cdot)$.

In **statistical modeling**, given some probability space $(\Omega, \mathcal{A}, \mathbb{P})$, assume that y_i are realization of i.i.d. variables Y_i (given $\boldsymbol{X}_i = \boldsymbol{x}_i$) with distribution F_i. Then solve

$$\widehat{m}(\cdot) = \underset{m(\cdot)\in\mathcal{F}}{\operatorname{argmin}} \left\{ \log \mathcal{L}(m(\boldsymbol{x}); \boldsymbol{y}) \right\} = \underset{m(\cdot)\in\mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \log f(y_i; m(\boldsymbol{x}_i)) \right\}$$

where $\log \mathcal{L}$ denotes the log-likelihood.

Loss Functions

Fitting criteria are based on loss functions (also called cost functions). For a quantitative response, a popular one is the quadratic loss, $\ell(y, m(\boldsymbol{x})) = [y - m(\boldsymbol{x})]^2$.

Recall that

$$\begin{cases} \mathbb{E}(Y) = \underset{m\in\mathbb{R}}{\text{argmin}}\{\|Y - m\|_{\ell_2}\} = \underset{m\in\mathbb{R}}{\text{argmin}}\{\mathbb{E}\left([Y - m]^2\right)\} \\ \text{Var}(Y) = \underset{m\in\mathbb{R}}{\min}\{\mathbb{E}\left([Y - m]^2\right)\} = \mathbb{E}\left([Y - \mathbb{E}(Y)]^2\right) \end{cases}$$

The empirical version is

$$\begin{cases} \overline{y} = \underset{m\in\mathbb{R}}{\text{argmin}}\{\sum_{i=1}^{n} \frac{1}{n}[y_i - m]^2\} \\ s^2 = \underset{m\in\mathbb{R}}{\min}\{\sum_{i=1}^{n} \frac{1}{n}[y_i - m]^2\} = \sum_{i=1}^{n} \frac{1}{n}[y_i - \overline{y}]^2 \end{cases}$$

Loss Functions

Robust estimation is based on a different loss function, $\ell(y, m(\boldsymbol{x})) = |y - m(\boldsymbol{x})|$.

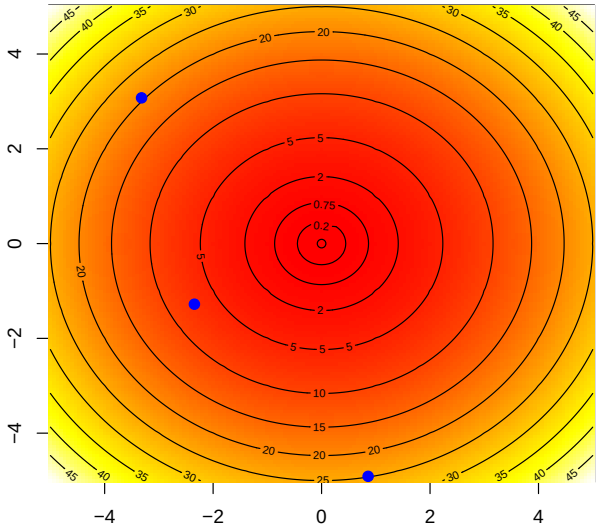
In the context of classification, we can use a misclassification indicator, $\ell(y, m(\boldsymbol{x})) = \mathbf{1}(y \neq m(\boldsymbol{x}))$

Note that those loss functions have symmetric weighting.

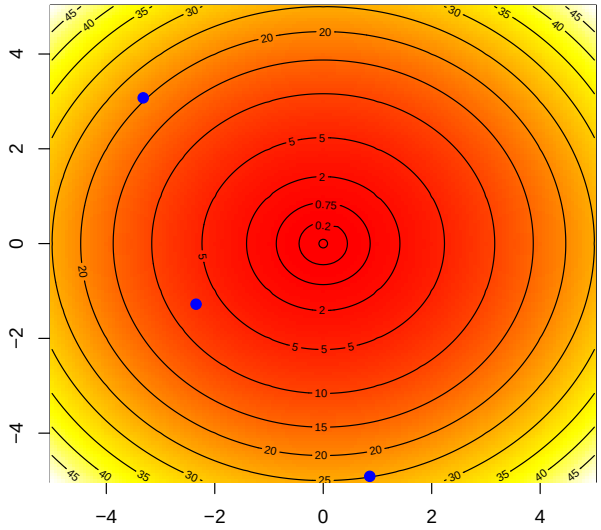
Computational Aspects: Optimization

Econometrics, Statistics and Machine Learning rely on the same object: optimization routines.

A gradient descent/ascent algorithm



A stochastic algorithm



Linear Predictors

In the linear model, least square estimator yields

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}} = \underbrace{\boldsymbol{X}[\boldsymbol{X}^\mathsf{T}\boldsymbol{X}]^{-1}\boldsymbol{X}^\mathsf{T}}_{\boldsymbol{H}}\boldsymbol{Y}$$

We have a linear predictor if the fitted value $\widehat{y}$ at point $\boldsymbol{x}$ can be written

$$\widehat{y} = \widehat{m}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{S}_{\boldsymbol{x},i} y_i = \boldsymbol{S}_{\boldsymbol{x}}^\mathsf{T}\boldsymbol{y}$$

where $\boldsymbol{S}_{\boldsymbol{x}}$ is some vector of weights (called smoother vector), related to a $n \times n$ smoother matrix,

$$\widehat{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{y}$$

where prediction is done at points $\boldsymbol{x}_i$'s.

Degrees of Freedom and Model Complexity

E.g.

$$\boldsymbol{S}_{\boldsymbol{x}} = \boldsymbol{X}[\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X}]^{-1}\boldsymbol{x}$$

that is related to the hat matrix, $\widehat{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y}$.

Note that

$$T = \frac{\|\boldsymbol{S}\boldsymbol{Y} - \boldsymbol{H}\boldsymbol{Y}\|}{\text{trace}([\boldsymbol{S} - \boldsymbol{H}]^{\mathsf{T}}[\boldsymbol{S} - \boldsymbol{H}])}$$

can be used to test a linear assumtion: if the model is linear, then T has a Fisher distribution.

In the context of linear predictors, trace($\boldsymbol{S}$) is usually called equivalent number of parameters and is related to $n-$ effective degrees of freedom (as in Ruppert *et al.* (2003)).

Model Evaluation

In linear models, the R^2 is defined as the proportion of the variance of the the response y that can be obtained using the predictors.

But maximizing the R^2 usually yields overfit (or unjustified optimism in Berk (2008)).

In linear models, consider the adjusted R^2,

$$\overline{R}^2 = 1 - [1 - R^2]\frac{n-1}{n-p-1}$$

where p is the number of parameters (or more generally trace($\boldsymbol{S}$)).

Model Evaluation

Alternatives are based on the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC), based on a penalty imposed on some criteria (the logarithm of the variance of the residuals),

$$AIC = \log\left(\frac{1}{n}\sum_{i=1}^{n}[y_i - \widehat{y}_i]^2\right) + \frac{2p}{n}$$

$$BIC = \log\left(\frac{1}{n}\sum_{i=1}^{n}[y_i - \widehat{y}_i]^2\right) + \frac{\log(n)p}{n}$$

In a more general context, replace p by $\text{trace}(\boldsymbol{S})$

Model Evaluation

One can also consider the expected prediction error (with a probabilistic model)

$$\mathbb{E}[\ell(Y, \widehat{m}(\boldsymbol{X})]$$

We cannot claim (using the law of large number) that

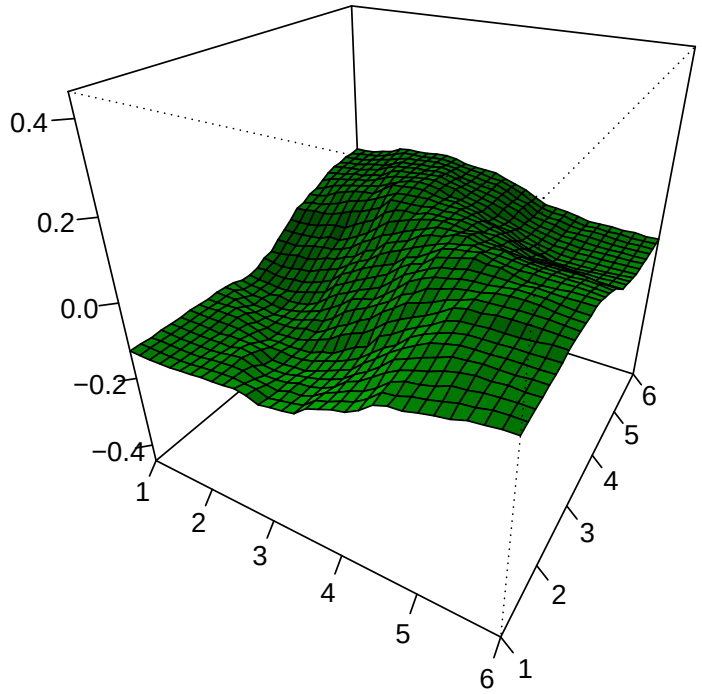
$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \widehat{m}(\boldsymbol{x}_i)) \overset{a.s.}{\not\to} \mathbb{E}[\ell(Y, m(\boldsymbol{X})]$$

since $\widehat{m}$ depends on $(y_i, \boldsymbol{x}_i)$'s.

Natural option : use two (random) samples, a training one and a validation one.

Alternative options, use cross-validation, leave-one-out or k-fold.

Underfit / Overfit and Variance - Mean Tradeoff



Underfit / Overfit and Variance - Mean Tradeoff

Goal in predictive modeling: reduce uncertainty in our predictions.

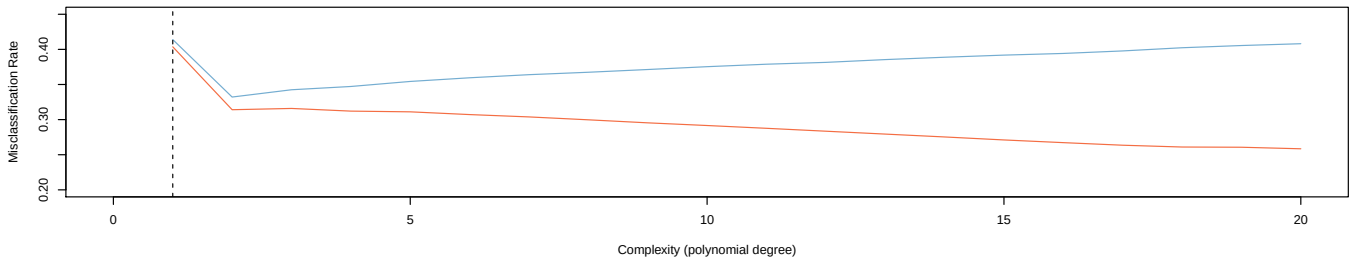
Need more data to get a better knowledge.

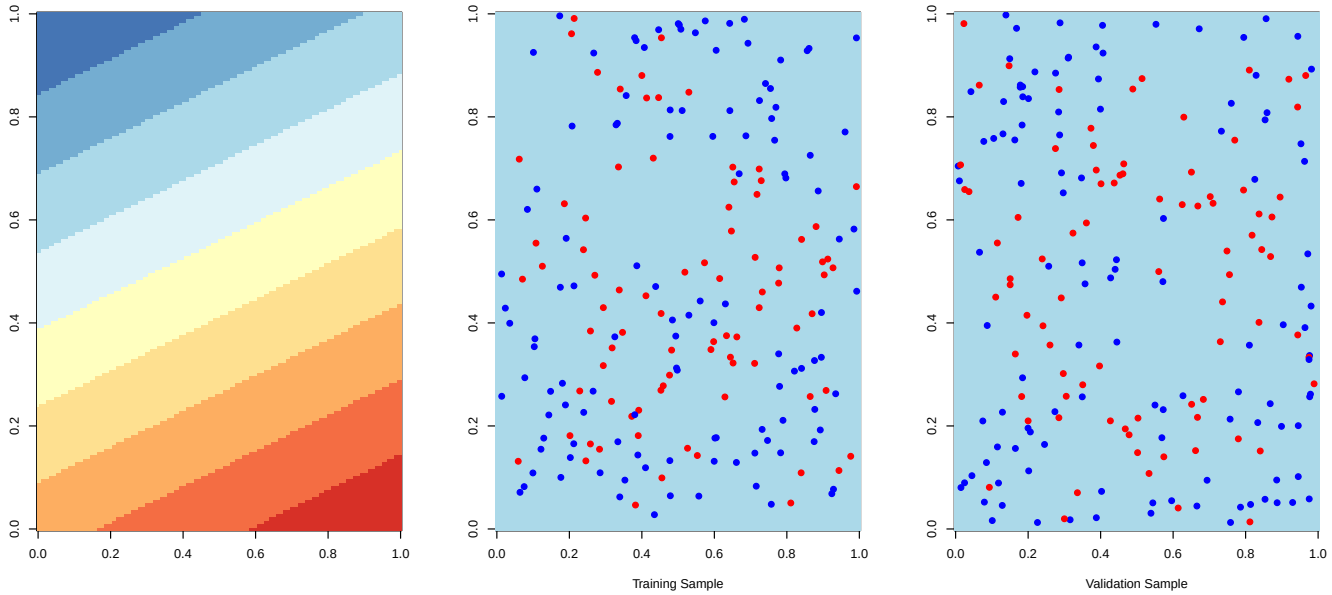
Unfortunately, reducing the error of the prediction on a dataset does not generally give a good generalization performance

$\longrightarrow$ need a training and a validation dataset

Overfit, Training vs. Validation and Complexity (Vapnik Dimension)

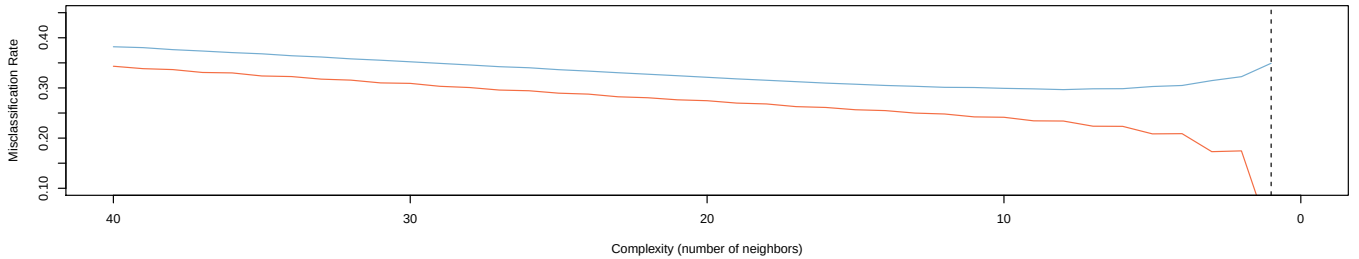
complexity $\longleftrightarrow$ polynomial degree

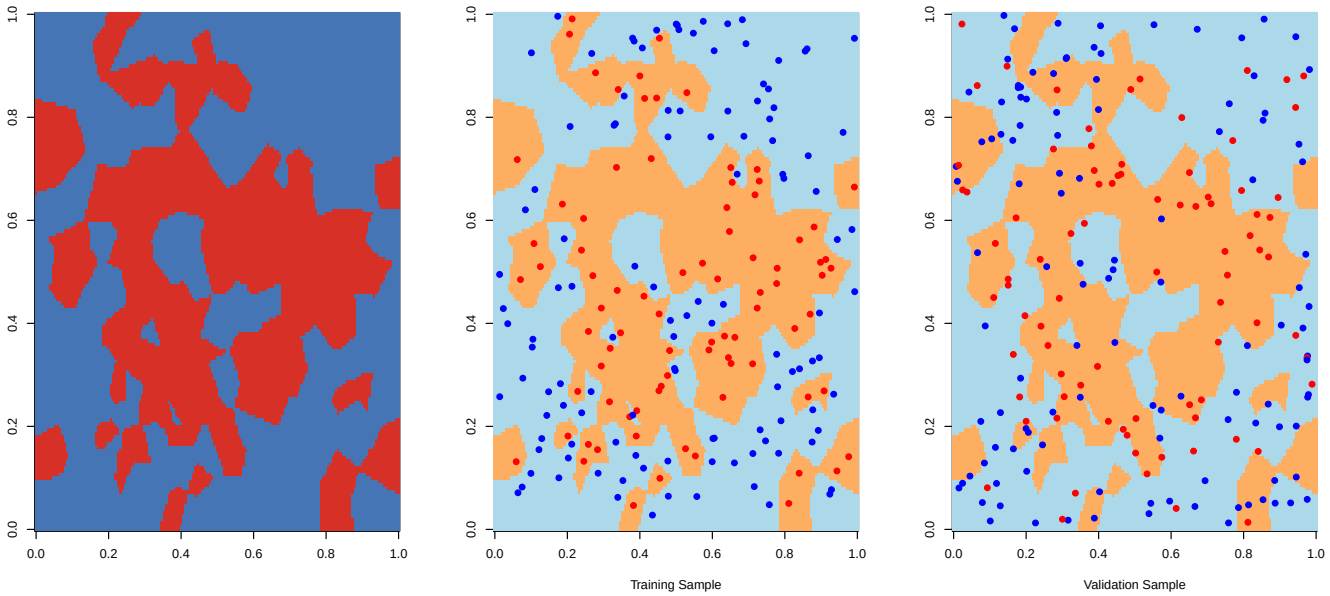




Overfit, Training vs. Validation and Complexity (Vapnik Dimension)

complexity $\longleftrightarrow$ number of neighbors (k)





Themes in Data Science

Predictive Capability we want here to have a model that predict well for new observations

Bias-Variance Tradeoff A very smooth prediction has less variance, but a large bias. We need to find a good balance between the bias and the variance

Loss Functions In machine learning, goodness of fit is discussed based on disparities between predicted values, and observed one, based on some loss function

Tuning or Meta Parameters Choice will be made in terms of tuning parameters

Interpretability Does it matter to have a good model if we cannot interpret it ?

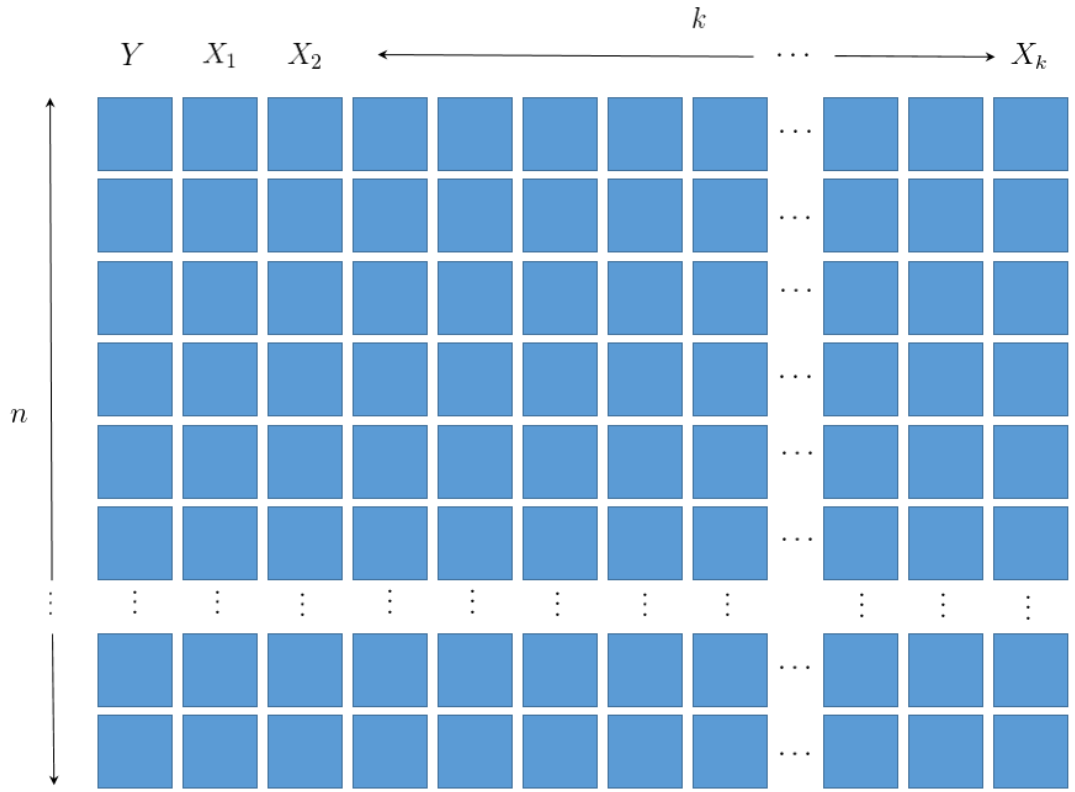
Coding Issues Most of the time, there are no analytical expression, just an alogrithm that should converge to some (possibly) optimal value

Data Data collection is a crucial issue (but will not be discussed here)

Scalability Issues

Dealing with big (or massive) datasets, large number of observations (n) and/or large number of predictors (features or covariates, k).

Ability to parallelize algorithms might be important (map-reduce).

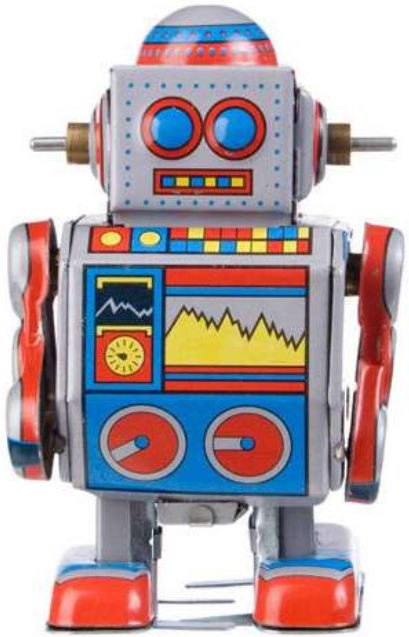


n can be large, but limited (portfolio size)
large variety k
large volume nk

$\rightarrow$ Feature Engineering

Part 2.

Classification, $y \in \{0, 1\}$



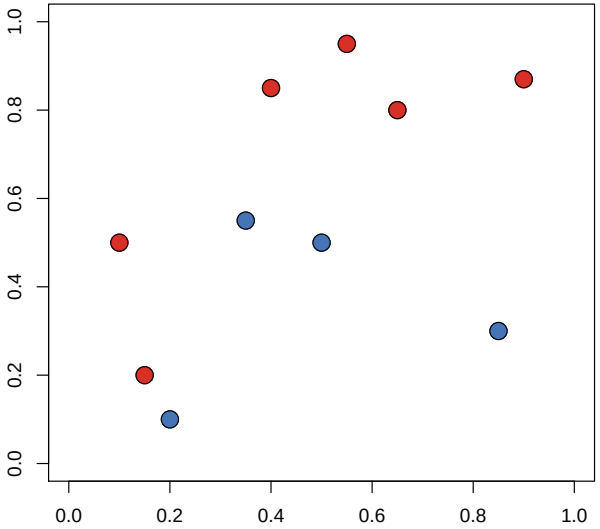
Classification?

Example: Fraud detection, automatic reading (classifying handwriting symbols), face recognition, accident occurence, death, purchase of optinal insurance cover, etc

Here $y_i \in \{0, 1\}$ or $y_i \in \{-1, +1\}$ or $y_i \in \{\bullet, \bullet\}$.

We look for a (good) predictive model here.
There will be two steps,

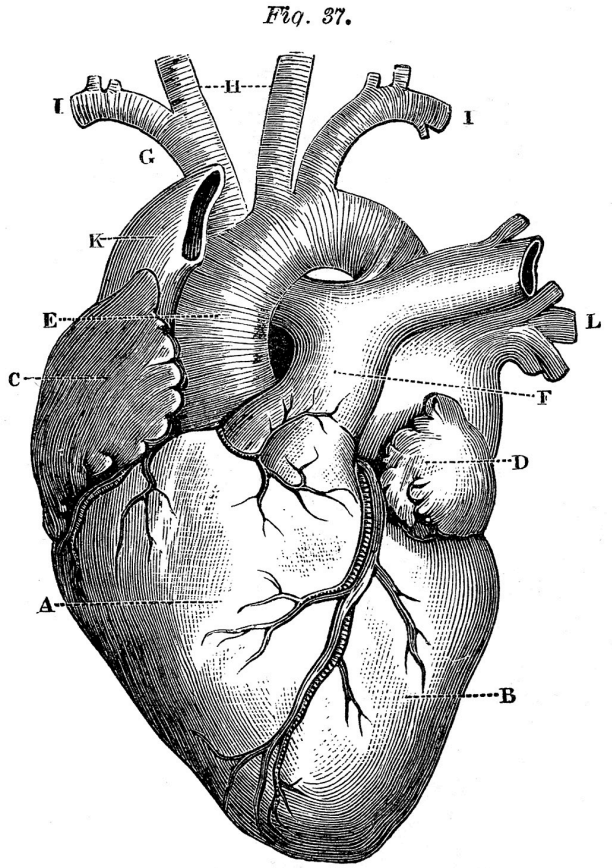
- the score function, $s(\boldsymbol{x}) = \mathbb{P}(Y = 1|\boldsymbol{X} = \boldsymbol{x}) \in [0, 1]$
- the classification function $s(\boldsymbol{x}) \to \widehat{Y} \in \{0, 1\}$.



Modeling a 0/1 random variable

Myocardial infarction of patients admited in E.R.

- heart rate (FRCAR),
- heart index (INCAR)
- stroke index (INSYS)
- diastolic pressure (PRDIA)
- pulmonary arterial pressure (PAPUL)
- ventricular pressure (PVENT)
- lung resistance (REPUL)
- death or survival (PRONO)

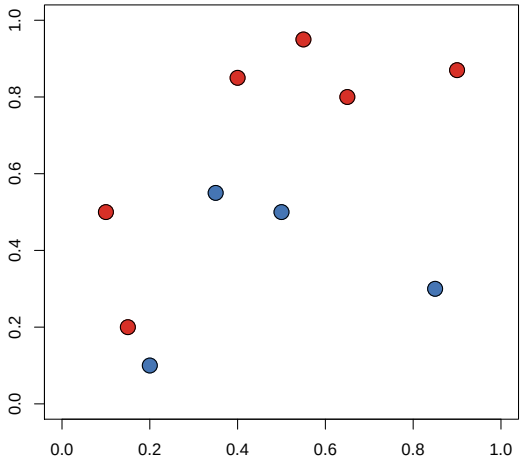



```
> myocarde=read.table("http://freakonometrics.free.fr/myocarde.csv",
    head=TRUE,sep=";")
```

Logistic Regression

Assume that $\mathbb{P}(Y_i = 1) = \pi_i$,

$$\text{logit}(\pi_i) = \boldsymbol{X}_i'\boldsymbol{\beta}, \text{ where } \text{logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right),$$



or

$$\pi_i = \text{logit}^{-1}(\boldsymbol{X}_i'\boldsymbol{\beta}) = \frac{\exp[\boldsymbol{X}_i'\boldsymbol{\beta}]}{1+\exp[\boldsymbol{X}_i^{\mathsf{T}}\boldsymbol{\beta}]}.$$

The log-likelihood is

$$\log\mathcal{L}(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i\log(\pi_i)+(1-y_i)\log(1-\pi_i) = \sum_{i=1}^{n} y_i\log(\pi_i(\boldsymbol{\beta}))+(1-y_i)\log(1-\pi_i(\boldsymbol{\beta}))$$

and the first order conditions are solved numerically

$$\frac{\partial\log\mathcal{L}(\boldsymbol{\beta})}{\partial\beta_k} = \sum_{i=1}^{n} X_{k,i}[y_i - \pi_i(\boldsymbol{\beta})] = 0.$$

Logistic Regression, Output (with R)

```
> logistic <- glm(PRONO~. , data=myocarde, family=binomial)
> summary(logistic)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -10.187642  11.895227  -0.856    0.392
FRCAR         0.138178   0.114112   1.211    0.226
INCAR        -5.862429   6.748785  -0.869    0.385
INSYS         0.717084   0.561445   1.277    0.202
PRDIA        -0.073668   0.291636  -0.253    0.801
PAPUL         0.016757   0.341942   0.049    0.961
PVENT        -0.106776   0.110550  -0.966    0.334
REPUL        -0.003154   0.004891  -0.645    0.519

(Dispersion parameter for binomial family taken to be 1)

Number of Fisher Scoring iterations: 7
```

Logistic Regression, Output (with R)

```
> library(VGAM)
> mlogistic <- vglm(PRONO~. , data=myocarde, family=multinomial)
> summary(mlogistic)

Coefficients:
              Estimate Std. Error   z value
(Intercept) 10.1876411 11.8941581  0.856525
FRCAR       -0.1381781  0.1141056 -1.210967
INCAR        5.8624289  6.7484319  0.868710
INSYS       -0.7170840  0.5613961 -1.277323
PRDIA        0.0736682  0.2916276  0.252610
PAPUL       -0.0167565  0.3419255 -0.049006
PVENT        0.1067760  0.1105456  0.965901
REPUL        0.0031542  0.0048907  0.644939

Name of linear predictor: log(mu[,1]/mu[,2])
```

Logistic (Multinomial) Regression

In the Bernoulli case, $y \in \{0, 1\}$,

$$\mathbb{P}(Y=1) = \frac{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}}}{1+e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}}} = \frac{p_1}{p_0+p_1} \propto p_1 \text{ and } \mathbb{P}(Y=0) = \frac{1}{1+e^{\boldsymbol{X}^\intercal}} = \frac{p_0}{p_0+p_1} \propto p_0$$

In the multinomial case, $y \in \{A, B, C\}$

$$\mathbb{P}(X=A) = \frac{p_A}{p_A+p_B+p_C} \propto p_A \text{ i.e. } \mathbb{P}(X=A) = \frac{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_A}}{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_B}+e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_B}+1}$$

$$\mathbb{P}(X=B) = \frac{p_B}{p_A+p_B+p_C} \propto p_B \text{ i.e. } \mathbb{P}(X=B) = \frac{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_B}}{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_A}+e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_B}+1}$$

$$\mathbb{P}(X=C) = \frac{p_C}{p_A+p_B+p_C} \propto p_C \text{ i.e. } \mathbb{P}(X=C) = \frac{1}{e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_A}+e^{\boldsymbol{X}^\intercal\boldsymbol{\beta}_B}+1}$$

Logistic Regression, Numerical Issues

The algorithm to compute $\widehat{\boldsymbol{\beta}}$ is

1. start with some initial value $\boldsymbol{\beta}_0$
2. define $\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} - H(\boldsymbol{\beta}_{k-1})^{-1}\nabla \log \mathcal{L}(\boldsymbol{\beta}_{k-1})$

where $\nabla \log \mathcal{L}(\boldsymbol{\beta})$is the gradient, and $H(\boldsymbol{\beta})$ the Hessian matrix, also called Fisher's score.

The generic term of the Hessian is

$$\frac{\partial^2 \log \mathcal{L}(\boldsymbol{\beta})}{\partial \beta_k \partial \beta_\ell} = \sum_{i=1}^{n} X_{k,i} X_{\ell,i} [y_i - \pi_i(\boldsymbol{\beta})]$$

Define $\boldsymbol{\Omega} = [\omega_{i,j}] = \text{diag}(\widehat{\pi}_i(1-\widehat{\pi}_i))$ so that the gradient is writen

$$\nabla \log \mathcal{L}(\boldsymbol{\beta}) = \frac{\partial \log \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}'(\boldsymbol{y} - \boldsymbol{\pi})$$

Logistic Regression, Numerical Issues

and the Hessian

$$H(\boldsymbol{\beta}) = \frac{\partial^2 \log \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} = -\boldsymbol{X}'\boldsymbol{\Omega}\boldsymbol{X}$$

The gradient descent algorithm is then

$$\boldsymbol{\beta}_k = (\boldsymbol{X}'\boldsymbol{\Omega}\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{\Omega}\boldsymbol{Z} \text{ where } \boldsymbol{Z} = \boldsymbol{X}\boldsymbol{\beta}_{k-1} + \boldsymbol{X}'\boldsymbol{\Omega}^{-1}(\boldsymbol{y} - \boldsymbol{\pi}),$$

From maximum likelihood properties,

$$\sqrt{n}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \xrightarrow{\mathcal{L}} \mathcal{N}(\mathbf{0}, I(\boldsymbol{\beta})^{-1}).$$

From a numerical point of view, this asymptotic variance $I(\boldsymbol{\beta})^{-1}$ satisfies $I(\boldsymbol{\beta})^{-1} = -H(\boldsymbol{\beta})$.

Logistic Regression, Numerical Issues

```
> X=cbind(1,as.matrix(myocarde[,1:7]))
> Y=myocarde$PRONO=="Survival"
> beta=as.matrix(lm(Y~0+X)$coefficients,ncol=1)
> for(s in 1:9){
+     pi=exp(X%*%beta[,s])/(1+exp(X%*%beta[,s]))
+     gradient=t(X)%*%(Y-pi)
+     omega=matrix(0,nrow(X),nrow(X));diag(omega)=(pi*(1-pi))
+     Hessian=-t(X)%*%omega%*%X
+     beta=cbind(beta,beta[,s]-solve(Hessian)%*%gradient)}
> beta
> -solve(Hessian)
> sqrt(-diag(solve(Hessian)))
```

Predicted Probability

Let $m(\boldsymbol{x}) = \mathbb{E}(Y|\boldsymbol{X} = \boldsymbol{x})$. With a logistic regression, we can get a prediction

$$\widehat{m}(\boldsymbol{x}) = \frac{\exp[\boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\beta}}]}{1+\exp[\boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\beta}}]}$$

```
> predict(logistic,type="response")[1:5]
        1         2         3         4         5
0.6013894 0.1693769 0.3289560 0.8817594 0.1424219
> predict(mlogistic,type="response")[1:5,]
      Death   Survival
1 0.3986106 0.6013894
2 0.8306231 0.1693769
3 0.6710440 0.3289560
4 0.1182406 0.8817594
5 0.8575781 0.1424219
```

Predicted Probability

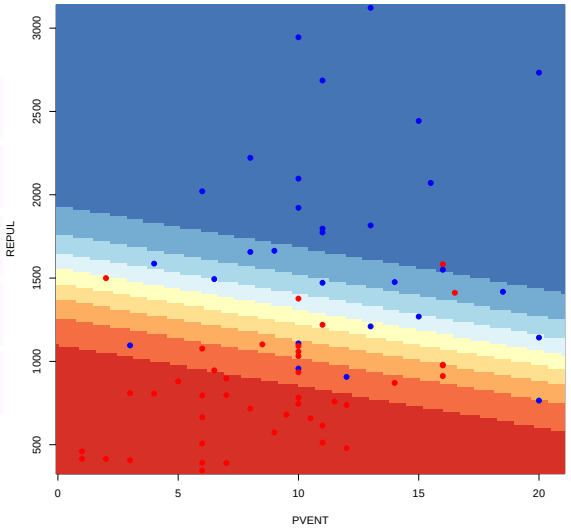
$$\widehat{m}(\boldsymbol{x}) = \frac{\exp[\boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\beta}}]}{1+\exp[\boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\beta}}]} = \frac{\exp[\widehat{\beta}_0+\widehat{\beta}_1 x_1+\cdots+\widehat{\beta}_k x_k]}{1+\exp[\widehat{\beta}_0+\widehat{\beta}_1 x_1+\cdots+\widehat{\beta}_k x_k]}$$

use

```
> predict(fit_glm, newdata = data, type="response")
```

e.g.

```
> GLM <- glm(PRONO ~ PVENT + REPUL, data =
    myocarde, family = binomial)
> pred_GLM = function(p,r){
+ return(predict(GLM, newdata =
+ data.frame(PVENT=p,REPUL=r), type="response")}
```

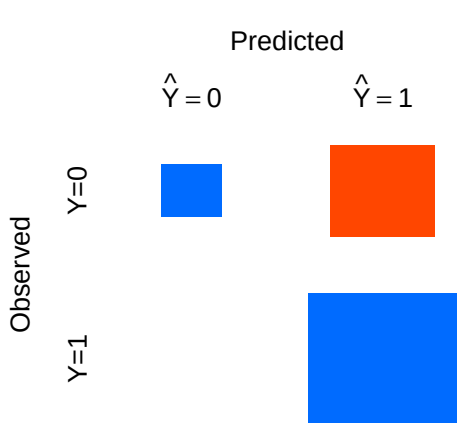


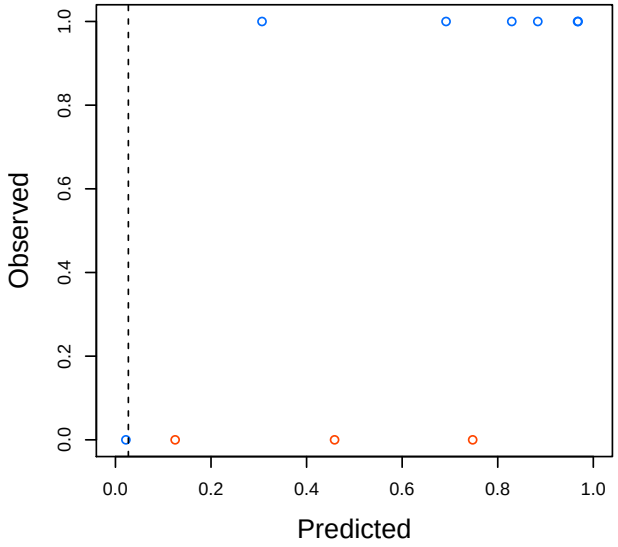
Predictive Classifier

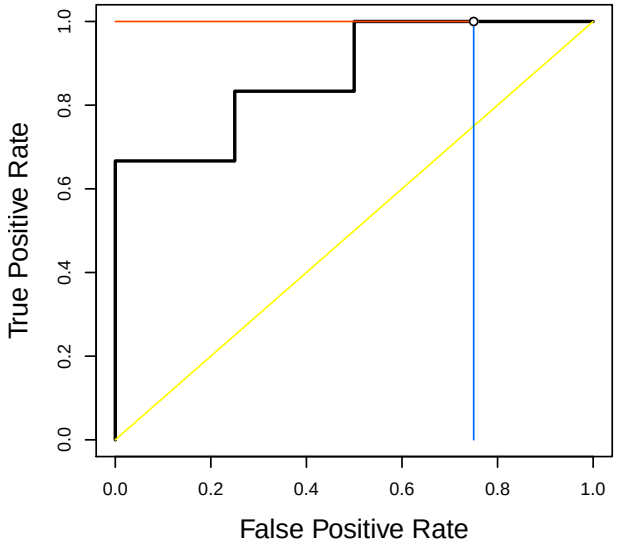
To go from a score to a class:

if $s(\boldsymbol{x}) > s$, then $\widehat{Y}(\boldsymbol{x}) = 1$ and $s(\boldsymbol{x}) \leq s$, then $\widehat{Y}(\boldsymbol{x}) = 0$

Plot $TP(s) = \mathbb{P}[\widehat{Y} = 1 | Y = 1]$ against $FP(s) = \mathbb{P}[\widehat{Y} = 1 | Y = 0]$







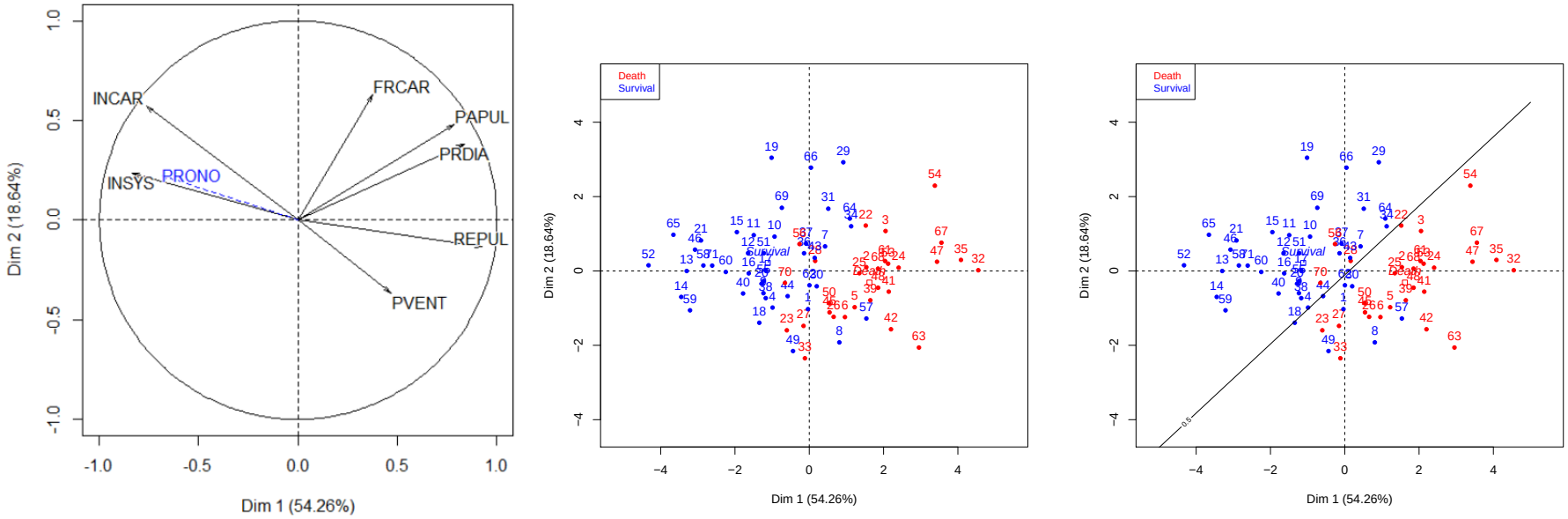
Predictive Classifier

With a threshold (e.g. $s = 50\%$) and the predicted probabilities, one can get a classifier and the confusion matrix

```
> probabilities <- predict(logistic, myocarde, type="response")
> predictions <- levels(myocarde$PRONO)[(probabilities>.5)+1]
> table(predictions, myocarde$PRONO)

predictions Death Survival
   Death       25         3
   Survival     4        39
```

Visualization of a Classifier in Higher Dimension...



Point $\boldsymbol{z} = (z_1, z_2, 0, \cdots, 0) \longrightarrow \boldsymbol{x} = (x_1, x_2, \cdots, x_k)$.

... but be carefull about interpretation !

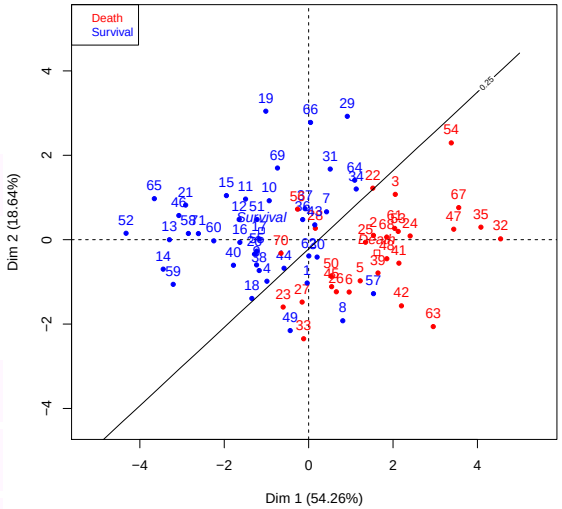
```
> prediction=predict(logistic,type="response")
```

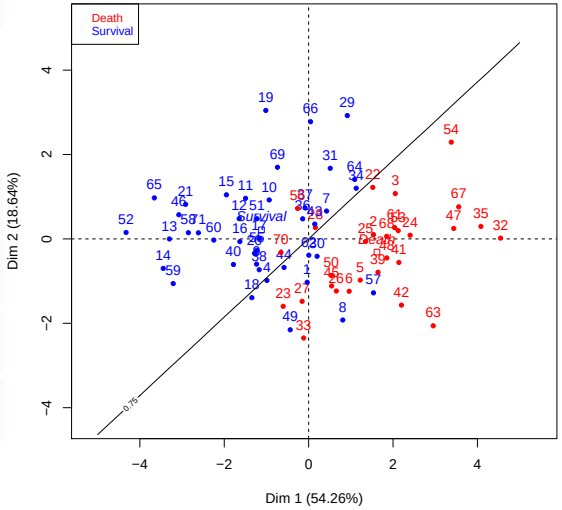
Use a 25% probability threshold

```
>  table(prediction>.25, myocarde$PRONO)
       Death Survival
  FALSE    19        2
  TRUE     10       40
```

or a 75% probability threshold

```
>  table(prediction>.75, myocarde$PRONO)
       Death Survival
  FALSE    27        9
  TRUE      2       33
```





Why a Logistic and not a Probit Regression?

Bliss (1934)) suggested a model such that

$$\mathbb{P}(Y=1|\boldsymbol{X}=\boldsymbol{x})=H(\boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}) \text{ where } H(\cdot)=\Phi(\cdot)$$

the c.d.f. of the $\mathcal{N}(0,1)$ distribution. This is the probit model. This yields a latent model, $y_i=\mathbf{1}(y_i^\star>0)$ where

$$y_i^\star=\boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}+\varepsilon_i \text{ is a nonobservable score.}$$

In the logistic regression, we model the odds ratio,

$$\frac{\mathbb{P}(Y=1|\boldsymbol{X}=\boldsymbol{x})}{\mathbb{P}(Y\neq 1|\boldsymbol{X}=\boldsymbol{x})}=\exp[\boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}]$$

$$\mathbb{P}(Y=1|\boldsymbol{X}=\boldsymbol{x})=H(\boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}) \text{ where } H(\cdot)=\frac{\exp[\cdot]}{1+\exp[\cdot]}$$

which is the c.d.f. of the logistic variable, see Verhulst (1845)

Table 3.2 Transformation of percentages to probits

%	0	1	2	3	4	5	6	7	8	9
0	—	2.67	2.95	3.12	3.25	3.36	3.45	3.52	3.59	3.66
10	3.72	3.77	3.82	3.87	3.92	3.96	4.01	4.05	4.08	4.12
20	4.16	4.19	4.23	4.26	4.29	4.33	4.36	4.39	4.42	4.45
30	4.48	4.50	4.53	4.56	4.59	4.61	4.64	4.67	4.69	4.72
40	4.75	4.77	4.80	4.82	4.85	4.87	4.90	4.92	4.95	4.97
50	5.00	5.03	5.05	5.08	5.10	5.13	5.15	5.18	5.20	5.23
60	5.25	5.28	5.31	5.33	5.36	5.39	5.41	5.44	5.47	5.50
70	5.52	5.55	5.58	5.61	5.64	5.67	5.71	5.74	5.77	5.81
80	5.84	5.88	5.92	5.95	5.99	6.04	6.08	6.13	6.18	6.23
90	6.28	6.34	6.41	6.48	6.55	6.64	6.75	6.88	7.05	7.33
—	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
99	7.33	7.37	7.41	7.46	7.51	7.58	7.65	7.75	7.88	8.09

Soit p la population : représentons par dp l'accroissement infiniment petit qu'elle reçoit pendant un temps infiniment court dt. Si la population croissait en progression géométrique, nous aurions l'équation $\frac{dp}{dt}=mp$. Mais comme la vitesse d'accroissement de la population est retardée par l'augmentation même du nombre des habitans, nous devrons retrancher de mp une fonction inconnue de p; de manière que la formule à intégrer deviendra

$$\frac{dp}{dt}=mp-\varphi(p).$$

L'hypothèse la plus simple que l'on puisse faire sur la forme de la fonction φ, est de supposer $\varphi(p)=np^2$. On trouve alors pour intégrale de l'équation ci-dessus

$$t=\frac{1}{m}[\log. p-\log.(m-np)]+\text{constante},$$

et il suffira de trois observations pour déterminer les deux coefficiens constans m et n et la constante arbitraire.

En résolvant la dernière équation par rapport à p, il vient

$$p=\frac{mp'e^{mt}}{np'e^{mt}+m-np'} \quad \ldots\ldots (1)$$

en désignant par p' la population qui répond à $t=o$, et par e la base des logarithmes népériens. Si l'on fait $t=\infty$, on voit que la valeur de p correspondante est $P=\frac{m}{n}$. Telle est donc *la limite supérieure de la population*.

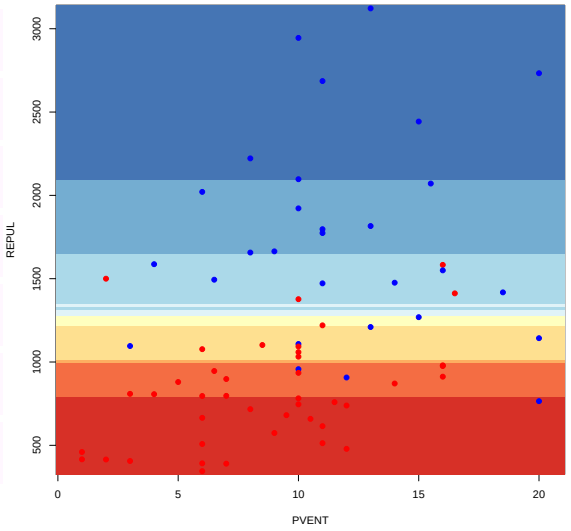
k-Nearest Neighbors (a.k.a. k-NN)

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. (Source: wikipedia).

$$\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] \sim \frac{1}{k} \sum_{d(\boldsymbol{x_i},\boldsymbol{x}) \text{ small}} y_i$$

For k-Nearest Neighbors, the class is usually the majority vote of the k closest neighbors of $\boldsymbol{x}$.

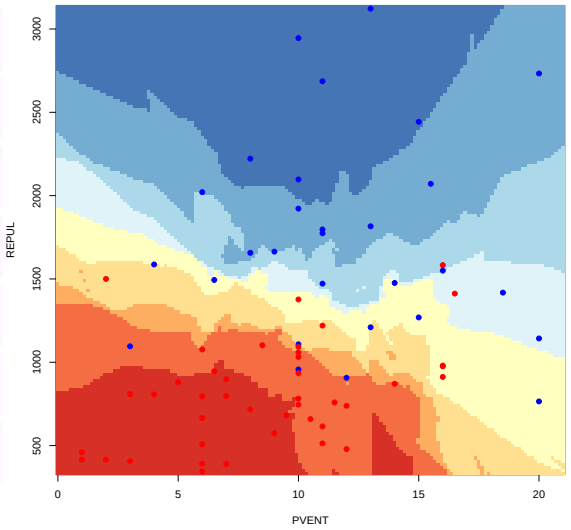
```
> library(caret)
> KNN <- knn3(PRONO ~ PVENT + REPUL, data =
    myocarde, k = 15)
>
> pred_KNN = function(p,r){
+ return(predict(KNN, newdata =
+ data.frame(PVENT=p,REPUL=r), type="prob")[,2]}
```



k-Nearest Neighbors

Distance $d(\cdot,\cdot)$ should not be sensitive to units: normalize by standard deviation

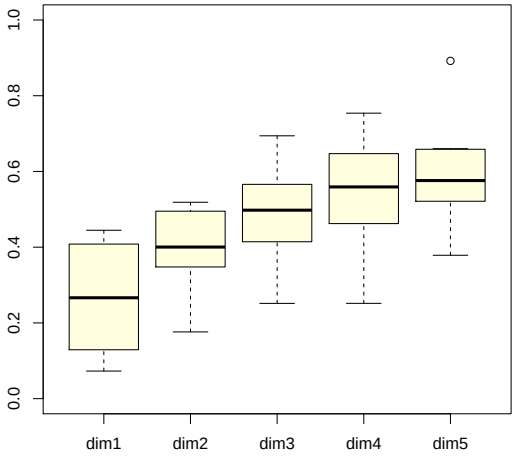
```
> sP <- sd(myocarde$PVENT); sR <- sd(myocarde$
    REPUL)
> KNN <- knn3(PRONO ~ I(PVENT/sP) + I(REPUL/sR),
    data = myocarde, k = 15)
> pred_KNN = function(p,r){
+ return(predict(KNN, newdata =
+ data.frame(PVENT=p, REPUL=r), type="prob")[,2]}
```



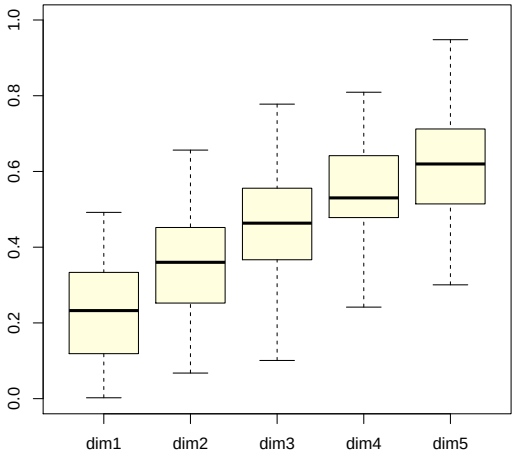
k-Nearest Neighbors and Curse of Dimensionality

The higher the dimension, the larger the distance to the closest neigbbor

$$\min_{i\in\{1,\cdots,n\}}\{d(\boldsymbol{a},\boldsymbol{x}_i)\}, \boldsymbol{x}_i\in\mathbb{R}^d.$$



$n = 10$



$n = 100$

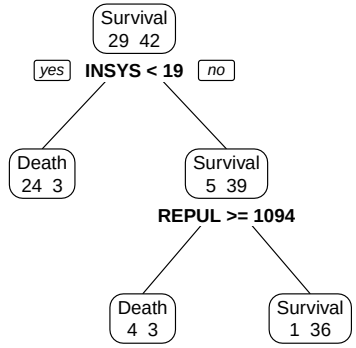
Classification (and Regression) Trees, CART

one of the predictive modelling approaches used in statistics, data mining and machine learning [...] In tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. (Source: wikipedia).

```
> library(rpart)
> cart<-rpart(PRONO~., data = myocarde)
> library(rpart.plot)
> library(rattle)
> prp(cart, type=2, extra=1)
```

or

```
> fancyRpartPlot(cart, sub="")
```



Classification (and Regression) Trees, CART

The impurity is a function φ of the probability to have 1 at node N, i.e. $\mathbb{P}[Y = 1|\text{ node } N]$, and

$$\mathcal{I}(N) = \varphi(\mathbb{P}[Y = 1|\text{ node } N])$$

φ is nonnegative ($\varphi \geq 0$), symmetric ($\varphi(p) = \varphi(1-p)$), with a minimum in 0 and 1 ($\varphi(0) = \varphi(1) < \varphi(p)$), e.g.

- Bayes error: $\varphi(p) = \min\{p, 1-p\}$
- cross-entropy: $\varphi(p) = -p\log(p) - (1-p)\log(1-p)$
- Gini index: $\varphi(p) = p(1-p)$

Those functions are concave, minimum at $p = 0$ and 1, maximum at $p = 1/2$.

Classification (and Regression) Trees, CART

To split N into two $\{N_L, N_R\}$, consider

$$\mathcal{I}(N_L, N_R) \sum_{x\in\{L,R\}} \frac{n_x}{n}\mathcal{I}(N_x)$$

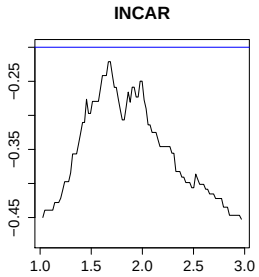
e.g. Gini index (used originally in CART, see Breiman *et al.* (1984))

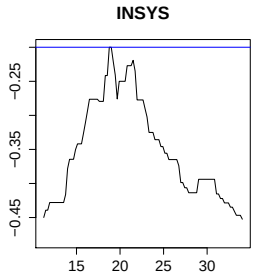
$$\text{gini}(N_L, N_R) = -\sum_{x\in\{L,R\}} \frac{n_x}{n} \sum_{y\in\{0,1\}} \frac{n_{x,y}}{n_x}\left(1-\frac{n_{x,y}}{n_x}\right)$$

and the cross-entropy (used in C4.5 and C5.0)

$$\text{entropy}(N_L, N_R) = -\sum_{x\in\{L,R\}} \frac{n_x}{n} \sum_{y\in\{0,1\}} \frac{n_{x,y}}{n_x}\log\left(\frac{n_{x,y}}{n_x}\right)$$

Classification (and Regression) Trees, CART

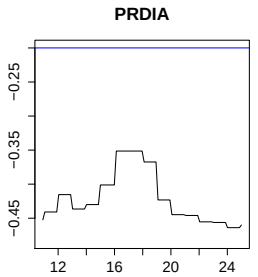


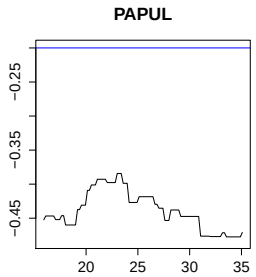


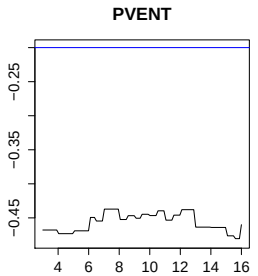
$$N_L\colon \{x_{i,j} \leq s\} \quad N_R\colon \{x_{i,j} > s\}$$

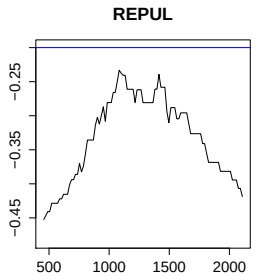
$$\text{solve} \max_{j\in\{1,\cdots,k\},s} \{\mathcal{I}(N_L, N_R)\}$$

$\longleftarrow$ first split

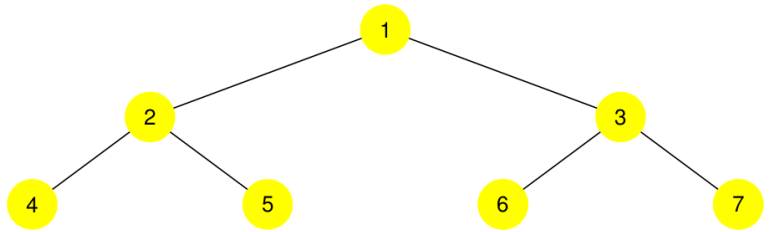


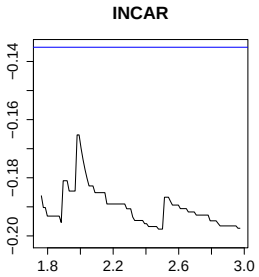


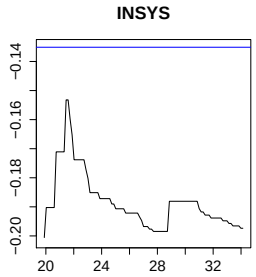


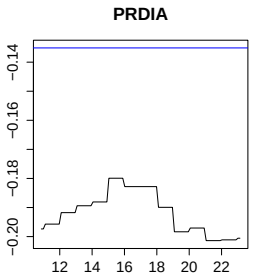


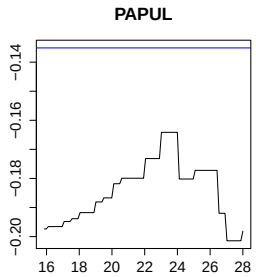
second split $\longrightarrow$

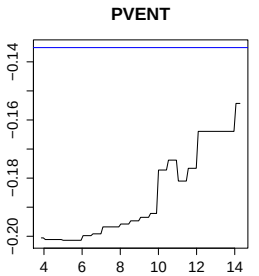


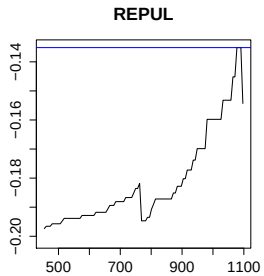












Pruning Trees

One can grow a big tree, until leaves have a (preset) small number of observations, and then possibly go back and prune branches (or leaves) that do not improve gains on good classification sufficiently.

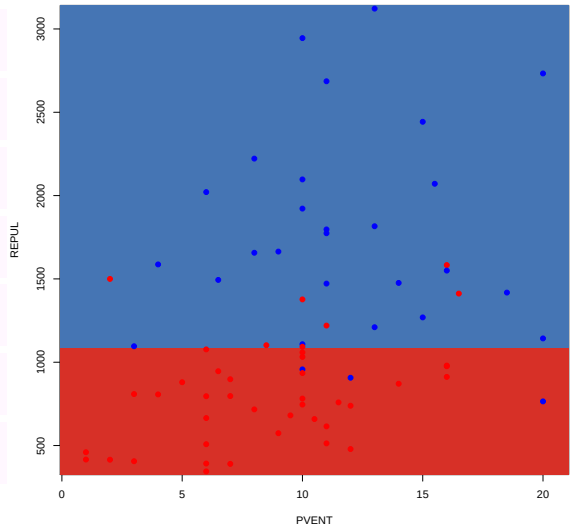
Or we can decide, at each node, whether we split, or not.

Pruning Trees

In trees, overfitting increases with the number of steps, and leaves. Drop in impurity at node N is defined as

$$\Delta\mathcal{I}(N_L, N_R) = \mathcal{I}(N) - \mathcal{I}(N_L, N_R) = \mathcal{I}(N) - \left(\frac{n_L}{n}\mathcal{I}(N_L) - \frac{n_R}{n}\mathcal{I}(N_R)\right)$$

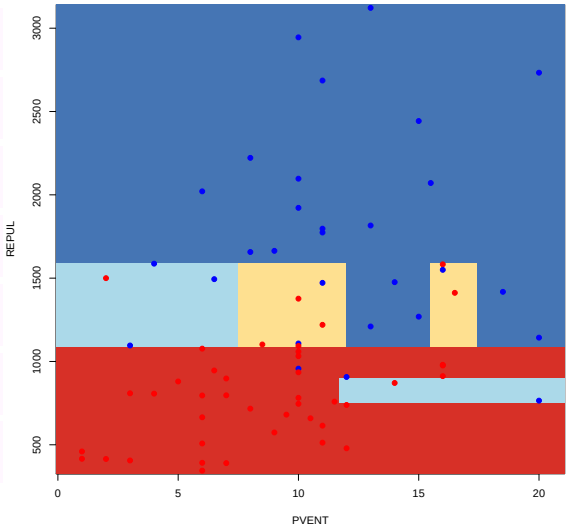
```
> library(rpart)
> CART <- rpart(PRONO ~ PVENT + REPUL, data =
    myocarde, minsplit = 20)
>
> pred_CART = function(p,r){
+ return(predict(CART, newdata =
+ data.frame(PVENT=p,REPUL=r)[,"Survival"])}
```



$\longrightarrow$ we cut if $\Delta\mathcal{I}(N_L, N_R)/\mathcal{I}(N)$ (relative gain) exceeds `cp` (complexity parameter, default 1%).

Pruning Trees

```
> library(rpart)
> CART <- rpart(PRONO ~ PVENT + REPUL, data =
    myocarde, minsplit = 5)
>
> pred_CART = function(p,r){
+ return(predict(CART, newdata =
+ data.frame(PVENT=p,REPUL=r)[,"Survival"])}
```



See also

```
> library(mvpart)
> ?prune
```

Define the missclassification rate of a tree $R(\text{tree})$

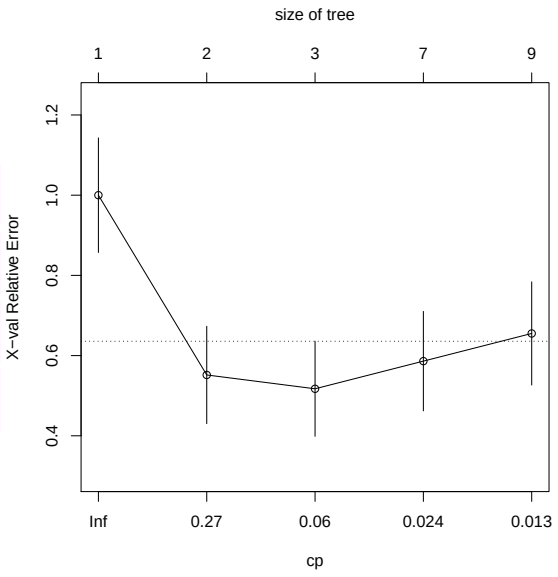
Pruning Trees

Given a cost-complexity parameter `cp` (see tunning parameter in Ridge-Lasso) define a penalized $R(\cdot)$

$$R_{\text{cp}}(\text{tree}) = \underbrace{R(\text{tree})}_{\text{loss}} + \underbrace{\text{cp}\|\text{tree}\|}_{\text{complexity}}$$

If `cp` is small the optimal tree is large, if `cp` is large the optimal tree has no leaf, see Breiman *et al.* (1984).

```
> cart <- rpart(PRONO ~ .,data = myocarde,minsplit
    =3)
> plotcp(cart)
> prune(cart, cp=0.06)
```



Bagging

Bootstrapped Aggregation (Bagging) , is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification (Source: wikipedia).

It is an ensemble method that creates multiple models of the same type from different sub-samples of the same dataset [boostrap]. The predictions from each separate model are combined together to provide a superior result [aggregation].

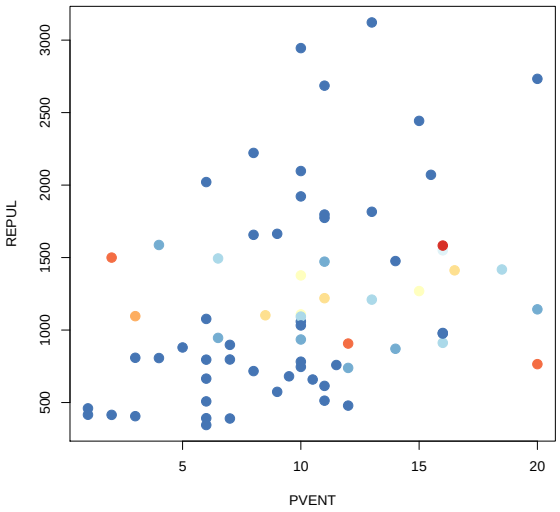
→ can be used on any kind of model, but interesting for trees, see Breiman (1996)

Boostrap can be used to define the concept of margin,

$$\text{margin}_i = \frac{1}{B}\sum_{b=1}^{B} \mathbf{1}(\widehat{y}_i = y_i) - \frac{1}{B}\sum_{b=1}^{B} \mathbf{1}(\widehat{y}_i \neq y_i)$$

Remark Probability that ith raw is not selection $(1-n^{-1})^n \to e^{-1} \sim 36.8\%$, cf training / validation samples (2/3-1/3)

Bagging Trees

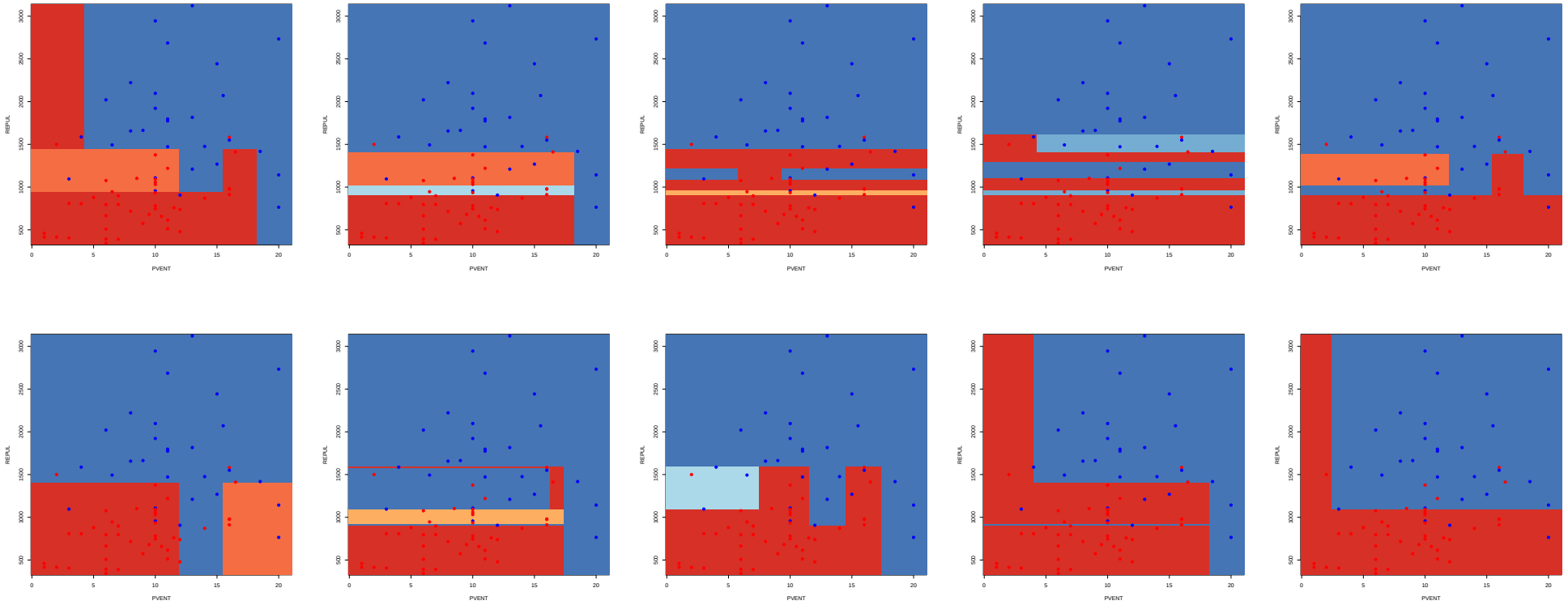



```
> margin <- matrix(NA,1e4,n)
> for(b in 1:1e4){
+ idx = sample(1:n,size=n,replace=TRUE)
> cart <- rpart(PRONO~PVENT+REPUL,
+ data=myocarde[idx,], minsplit = 5)
> margin[j,] <- (predict(cart2,newdata=
    myocarde,type="prob")[,"Survival"]>.5)!=
    (myocarde$PRONO=="Survival")
+ }
> apply(margin, 2, mean)
```

Bagging

Bagging Trees

Interesting because of instability in CARTs (in terms of tree structure, not necessarily prediction)



Bagging and Variance, Bagging and Bias

Assume that $y = m(\boldsymbol{x}) + \varepsilon$. The mean squared error over repeated random samples can be decomposed in three parts Hastie *et al.* (2001)

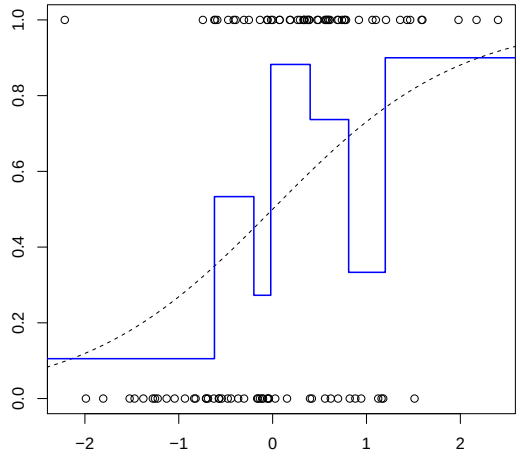
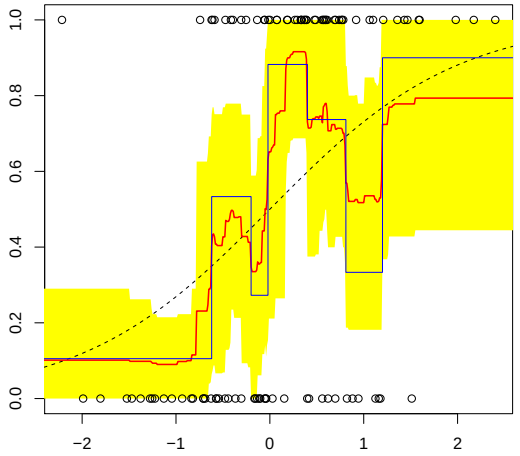
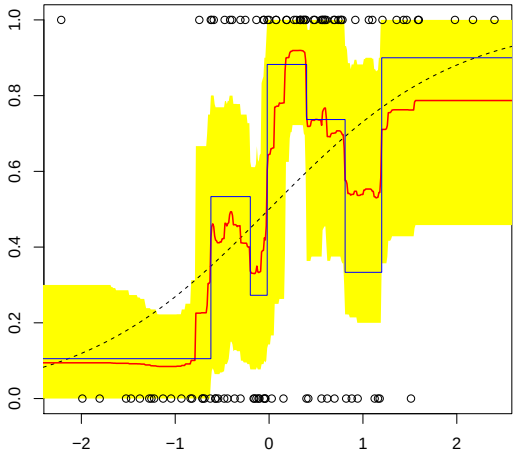
$$\mathbb{E}[(Y - \widehat{m}(\boldsymbol{x}))^2] = \underbrace{\sigma^2}_{1} + \underbrace{\big[\mathbb{E}[\widehat{m}(\boldsymbol{x})] - m(\boldsymbol{x})\big]^2}_{2} + \underbrace{\mathbb{E}\Big(\big[\widehat{m}(\boldsymbol{x}) - \mathbb{E}[(\widehat{m}(\boldsymbol{x})]\big]^2\Big)}_{3}$$

1 reflects the variance of Y around $m(\boldsymbol{x})$

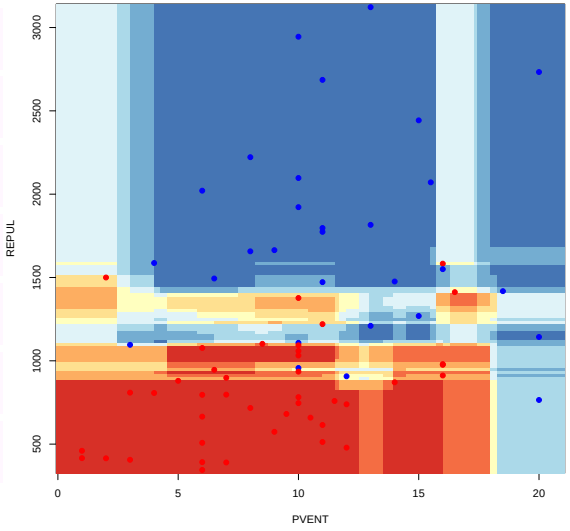
2 is the squared bias of $\widehat{m}(\boldsymbol{x})$

3 is the variance of $\widehat{m}(\boldsymbol{x})$

$\longrightarrow$ bias-variance tradeoff. Boostrap can be used to reduce the bias, and he variance (but be careful of outliers)

```
> library(ipred)
> BAG <- bagging(PRONO ~ PVENT + REPUL, data =
    myocarde)
>
> pred_BAG = function(p,r){
+ return(predict(BAG,newdata=
+ data.frame(PVENT=p,REPUL=r), type="prob")[,2])}
```

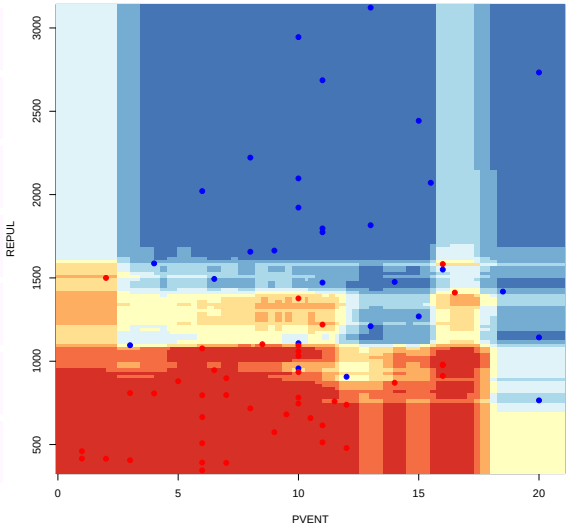


Random Forests

Strictly speaking, when boostrapping among observations, and aggregating, we use a bagging algorithm.

In the random forest algorithm, we combine Breiman's bagging idea and the random selection of features, introduced independently by Ho (1995)) and Amit & Geman (1997))

```
> library(randomForest)
> RF <- randomForest(PRONO ~ PVENT + REPUL, data =
    myocarde)
>
> pred_RF = function(p,r){
+ return(predict(RF,newdata=
+ data.frame(PVENT=p,REPUL=r), type="prob")[,2])}
```



Random Forest

At each node, select $\sqrt{k}$ covariates out of k (randomly).

Random Forest

can deal with small n large k-problems

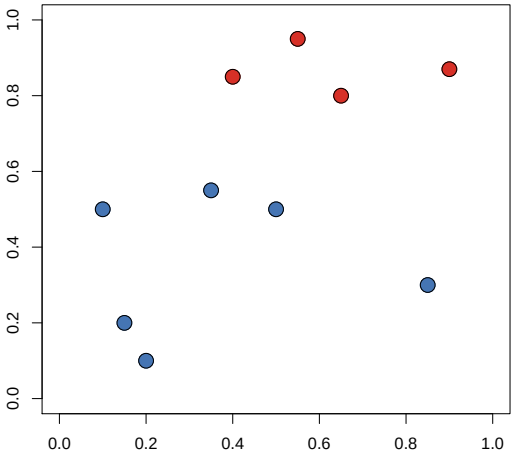
Random Forest are used not only for prediction, but also to assess variable importance (see last section).

Support Vector Machine

SVMs were developed in the 90's based on previous work, from Vapnik & Lerner (1963), see Vailant (1984)

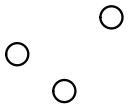
Assume that points are linearly separable, i.e. there is $\boldsymbol{\omega}$ and b such that

$$Y = \begin{cases} +1 \text{ if } \boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b > 0 \\ -1 \text{ if } \boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b < 0 \end{cases}$$

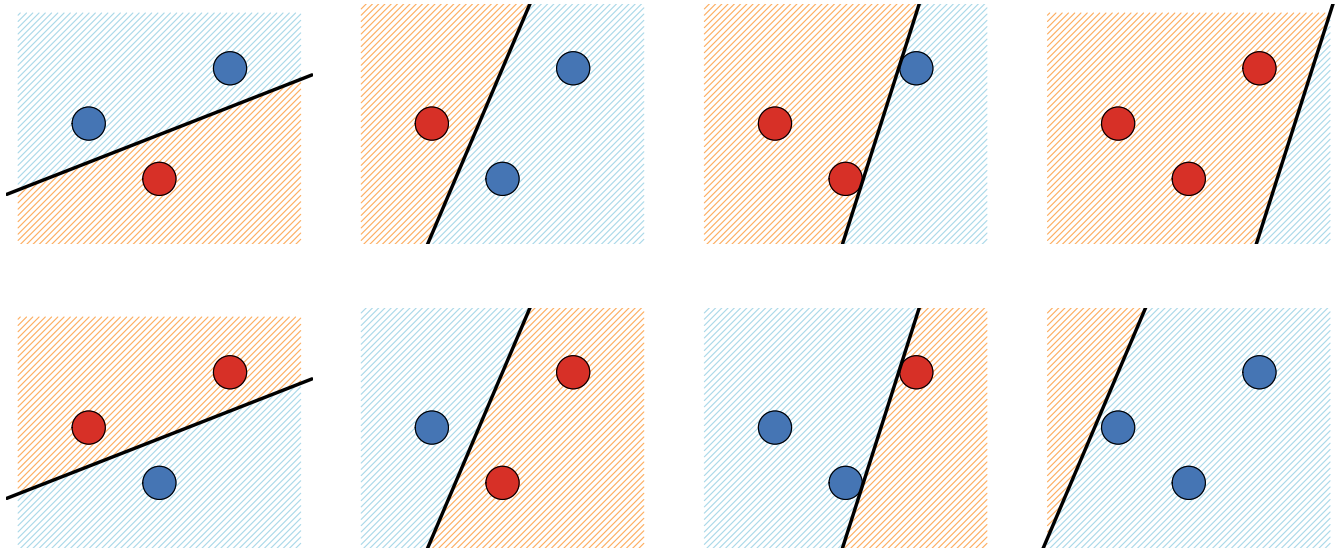


Problem: infinite number of solutions, need a good one, that separate the data, (somehow) far from the data.

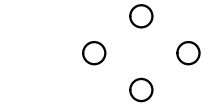
Concept : VC dimension. Let $\mathcal{H} : \{h : \mathbb{R}^d \mapsto \{-1, +1\}\}$. Then $\mathcal{H}$ is said to shatter a set of points $\boldsymbol{X}$ is all dichotomies can be achieved.
E.g. with those three points, all configurations can be achieved



Support Vector Machine



E.g. with those four points, several configurations cannot be achieved (with some linear separator, but they can with some quadratic one)



Support Vector Machine

Vapnik's (VC) dimension is the size of the largest shattered subset of $\boldsymbol{X}$.

This dimension is intersting to get an upper bound of the probability of miss-classification (with some complexity penalty, function of $\text{VC}(\mathcal{H})$).

Now, in practice, where is the optimal hyperplane ?

The distance from $\boldsymbol{x}_0$ to the hyperplane $\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}+b$ is

$$d(\boldsymbol{x}_0, H_{\boldsymbol{\omega},b}) = \frac{\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_0+b}{\|\boldsymbol{\omega}\|}$$

and the optimal hyperplane (in the separable case) is

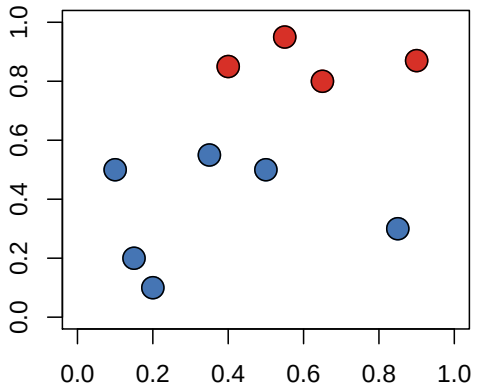
$$\text{argmin}\left\{\min_{i=1,\cdots,n} d(\boldsymbol{x}_i, H_{\boldsymbol{\omega},b})\right\}$$

Support Vector Machine

Define support vectors as observations such that

$$|\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b| = 1$$

The margin is the distance between hyperplanes defined by support vectors.



The distance from support vectors to $H_{\boldsymbol{\omega},b}$ is $\|\boldsymbol{\omega}\|^{-1}$, and the margin is then $2\|\boldsymbol{\omega}\|^{-1}$.

$\longrightarrow$ the algorithm is to minimize the inverse of the margins s.t. $H_{\boldsymbol{\omega},b}$ separates ± 1 points, i.e.

$$\min\left\{\frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{\omega}\right\} \text{ s.t. } Y_i(\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) \geq 1, \ \forall i.$$

Support Vector Machine

Problem difficult to solve: many inequality constraints (n)

$\longrightarrow$ solve the dual problem...

In the primal space, the solution was

$$\boldsymbol{\omega} = \sum \alpha_i Y_i \boldsymbol{x}_i \text{ with } \sum_{i=1} \alpha_i Y_i = 0.$$

In the dual space, the problem becomes (hint: consider the Lagrangian)

$$\max \left\{ \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i=1} \alpha_i \alpha_j Y_i Y_j \boldsymbol{x}_i^\mathsf{T} \boldsymbol{x}_j \right\} \text{ s.t. } \sum_{i=1} \alpha_i Y_i = 0.$$

which is usually written

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \boldsymbol{\alpha}^\mathsf{T} \boldsymbol{Q} \boldsymbol{\alpha} - \mathbf{1}^\mathsf{T} \boldsymbol{\alpha} \right\} \text{ s.t. } \begin{cases} 0 \leq \alpha_i \ \forall i \\ \boldsymbol{y}^\mathsf{T} \boldsymbol{\alpha} = 0 \end{cases}$$

where $\boldsymbol{Q} = [\boldsymbol{Q}_{i,j}]$ and $\boldsymbol{Q}_{i,j} = y_i y_j \boldsymbol{x}_i^\mathsf{T} \boldsymbol{x}_j$.

Support Vector Machine

Now, what about the non-separable case?

Here, we cannot have $y_i(\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) \geq 1 \ \forall i$.

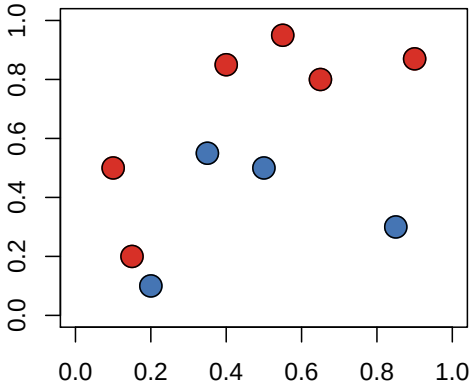
$\longrightarrow$ introduce slack variables,

$$\begin{cases} \boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b \geq +1 - \xi_i \text{ when } y_i = +1 \\ \boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b \leq -1 + \xi_i \text{ when } y_i = -1 \end{cases}$$

where $\xi_i \geq 0 \ \forall i$. There is a classification error when $\xi_i > 1$.

The idea is then to solve

$$\min\left\{\frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{\omega} + C\mathbf{1}^\mathsf{T}\mathbf{1}_{\boldsymbol{\xi}>1}\right\}, \text{ instead of } \min\left\{\frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{\omega}\right\}$$



Support Vector Machines, with a Linear Kernel

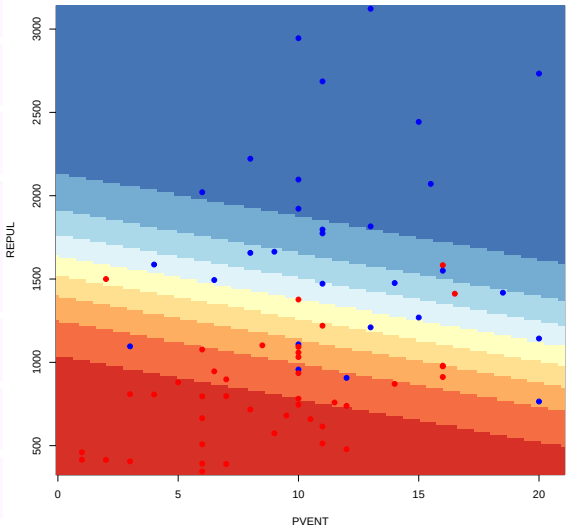
So far,

$$d(\boldsymbol{x}_0, H_{\boldsymbol{\omega},b}) = \min_{\boldsymbol{x}\in H_{\boldsymbol{\omega},b}} \{\|\boldsymbol{x}_0 - \boldsymbol{x}\|_{\ell_2}\}$$

where $\|\cdot\|_{\ell_2}$ is the Euclidean (ℓ_2) norm,

$$\|\boldsymbol{x}_0 - \boldsymbol{x}\|_{\ell_2} = \sqrt{(\boldsymbol{x}_0 - \boldsymbol{x})\cdot(\boldsymbol{x}_0 - \boldsymbol{x})} = \sqrt{\boldsymbol{x}_0\cdot\boldsymbol{x}_0 - 2\boldsymbol{x}_0\cdot\boldsymbol{x} + \boldsymbol{x}\cdot\boldsymbol{x}}$$

```
> library(kernlab)
> SVM2 <- ksvm(PRONO ~ PVENT + REPUL, data =
    myocarde,
+ prob.model = TRUE, kernel = "vanilladot")
> pred_SVM2 = function(p,r){
+ return(predict(SVM2,newdata=
+ data.frame(PVENT=p,REPUL=r), type="probabilities
    ")[,2])}
```



Support Vector Machines, with a Non Linear Kernel

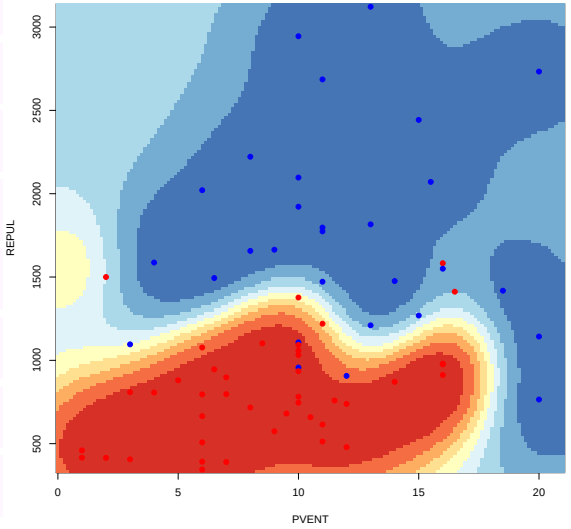
More generally,

$$d(\boldsymbol{x}_0, H_{\boldsymbol{\omega},b}) = \min_{\boldsymbol{x}\in H_{\boldsymbol{\omega},b}} \{\|\boldsymbol{x}_0 - \boldsymbol{x}\|_k\}$$

where $\|\cdot\|_k$ is some kernel-based norm,

$$\|\boldsymbol{x}_0 - \boldsymbol{x}\|_k = \sqrt{k(\boldsymbol{x}_0,\boldsymbol{x}_0) - 2k(\boldsymbol{x}_0,\boldsymbol{x}) + k(\boldsymbol{x}\cdot\boldsymbol{x})}$$

```
> library(kernlab)
> SVM2 <- ksvm(PRONO ~ PVENT + REPUL, data =
    myocarde,
+ prob.model = TRUE, kernel = "rbfdot")
> pred_SVM2 = function(p,r){
+ return(predict(SVM2,newdata=
+ data.frame(PVENT=p,REPUL=r), type="probabilities
    ")[,2])}
```

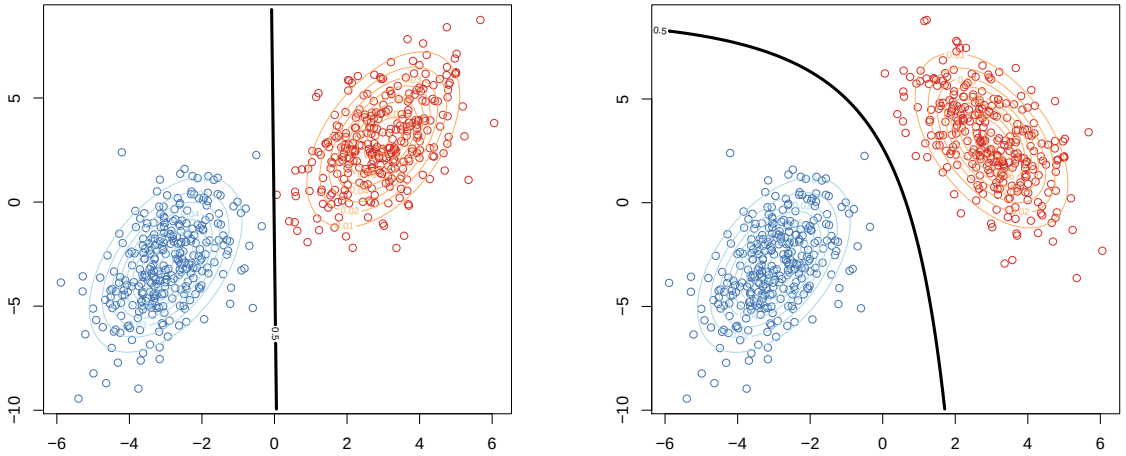


Still Hungry ?

There are still several (machine learning) techniques that can be used for classification

- Fisher's Linear or Quadratic Discrimination (closely related to logistic regression, and PCA), see Fisher (1936))

$$\boldsymbol{X}|Y=0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \quad \text{and} \quad \boldsymbol{X}|Y=1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$



Still Hungry ?

- Perceptron or more generally Neural Networks In machine learning, neural networks are a family of statistical learning models inspired by biological neural networks and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. wikipedia, see Rosenblatt (1957)
- Boosting (see next section)
- Naive Bayes In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. wikipedia, see Russell & Norvig (2003)

See also the (great) package

```
> library(caret)
```

Difference in Differences

In many applications (e.g. marketing), we do need two models to analyze the impact of a treatment. We need two groups, a control and a treatment group.

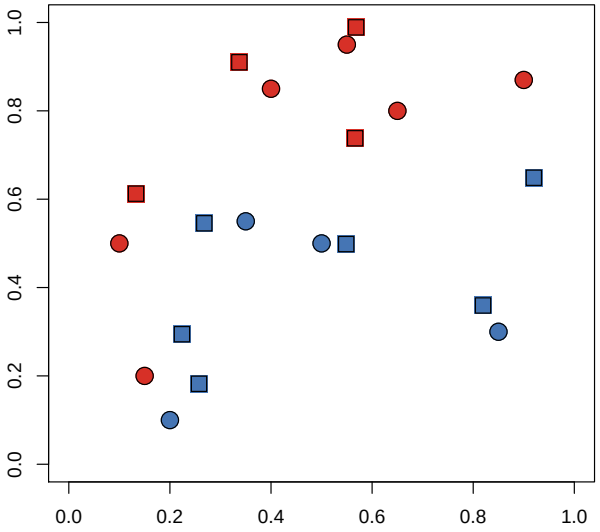
Data : $\{(\boldsymbol{x}_i, y_i)\}$ with $y_i \in \{\bullet, \bullet\}$

$\{(\boldsymbol{x}_j, y_j)\}$ with $y_i \in \{\blacksquare, \blacksquare\}$

See clinical trials, treatment vs. control group

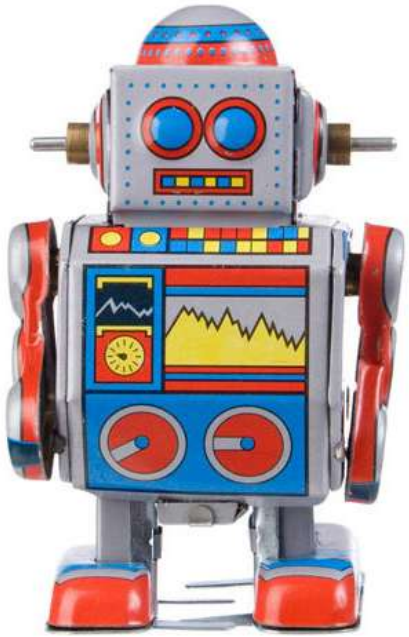
E.g. direct mail campaign in a bank

	Control ■	Promotion ●
No Purchase	85.17%	61.60%
Purchase	14.83%	38.40%



overall uplift effect +23.57%, see Guelman *et al.* (2014) for more details.

Part 3.
Regression



Regression?

In statistics, regression analysis is a statistical process for estimating the relationships among variables [...] In a narrower sense, regression may refer specifically to the estimation of continuous response variables, as opposed to the discrete response variables used in classification. (Source: wikipedia).

Here regression is opposed to classification (as in the CART algorithm). y is either a continuous variable $y \in \mathbb{R}$ or a counting variable $y \in \mathbb{N}$.

Regression? Parametrics, nonparametrics and machine learning

In many cases in econometric and actuarial literature we *simply* want a good fit for the conditional expectation, $\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}]$.

Regression analysis estimates the conditional expectation of the dependent variable given the independent variables (Source: wikipedia).

Example: A popular nonparametric technique, kernel based regression,

$$\widehat{m}(\boldsymbol{x}) = \frac{\sum_i Y_i \cdot K_h(\boldsymbol{X}_i - \boldsymbol{x})}{\sum_i K_h(\boldsymbol{X}_i - \boldsymbol{x})}$$

In econometric litterature, interest on asymptotic normality properties and plug-in techniques.

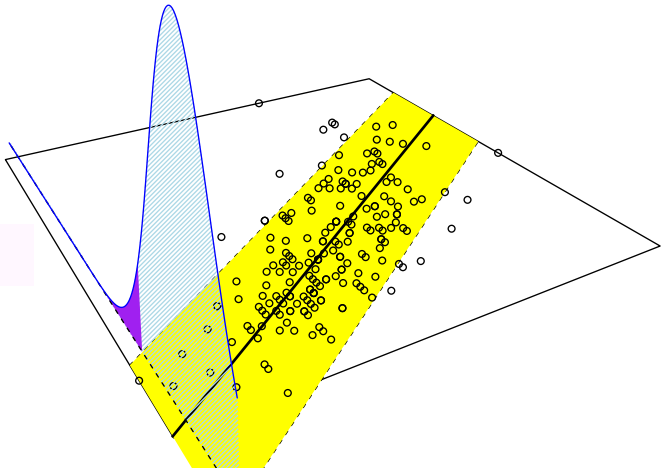
In machine learning, interest on out-of sample cross-validation algorithms.

Linear, Non-Linear and Generalized Linear

Linear Model:

- $(Y|\boldsymbol{X}=\boldsymbol{x}) \sim \mathcal{N}(\theta_{\boldsymbol{x}}, \sigma^2)$
- $\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] = \theta_{\boldsymbol{x}} = \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}$

```
> fit <- lm(y ~ x, data = df)
```

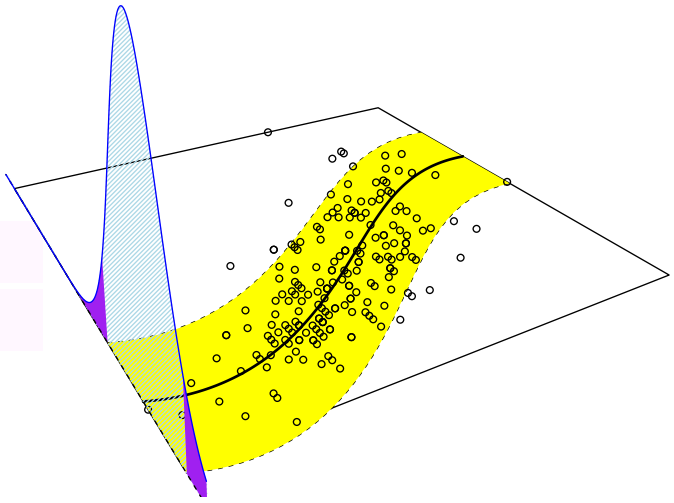


Linear, Non-Linear and Generalized Linear

NonLinear / NonParametric Model:

- $(Y|\boldsymbol{X}=\boldsymbol{x}) \sim \mathcal{N}(\theta_{\boldsymbol{x}}, \sigma^2)$
- $\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] = \theta_{\boldsymbol{x}} = m(\boldsymbol{x})$

```
> fit <- lm(y ~ poly(x, k), data = df)
> fit <- lm(y ~ bs(x), data = df)
```

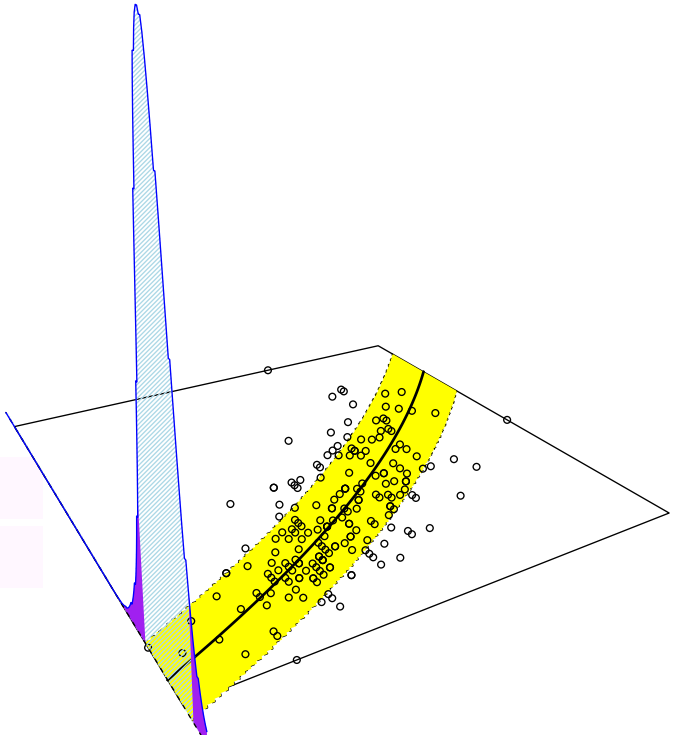


Linear, Non-Linear and Generalized Linear

Generalized Linear Model:

- $(Y|\boldsymbol{X}=\boldsymbol{x}) \sim \mathcal{L}(\theta_{\boldsymbol{x}}, \varphi)$
- $\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] = h^{-1}(\theta_{\boldsymbol{x}}) = h^{-1}(\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta})$

```
> fit <- glm(y ~ x, data = df,
+ family = poisson(link = "log")
```



Linear Model

Consider a linear regression model, $y_i = \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta} + \varepsilon_i$.

$\boldsymbol{\beta}$ is estimated using ordinary least squares, $\widehat{\boldsymbol{\beta}} = [\boldsymbol{X}^\mathsf{T}\boldsymbol{X}]^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{Y}$

$\longrightarrow$ best linear unbiased estimator

Unbiased estimators in important in statistics because they have nice mathematical properties (see Cramér-Rao lower bound).

Looking for biased estimators (bias-variance tradeoff) becomes important in high-dimension, see Burr & Fry (2005)

Linear Model and Loss Functions

Consider a linear model, with some general loss function ℓ, set $\ell(x,y)=R(x-y)$ and consider,

$$\widehat{\boldsymbol{\beta}} \in \text{argmin}\left\{\sum_{i=1}^{n} \ell(y_i, \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})\right\}$$

If R is differentiable, the first order condition would be

$$\sum_{i=1}^{n} R'\left(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}\right)\cdot \boldsymbol{x}_i^\mathsf{T} = 0.$$

i.e.

$$\sum_{i=1}^{n} \underbrace{\omega\left(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}\right)}_{\omega_i}\cdot \left(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}\right)\boldsymbol{x}_i^\mathsf{T} = 0 \text{ with } \omega(x)=\frac{R'(x)}{x},$$

It is the first order condition of a weighted ℓ_2 regression.

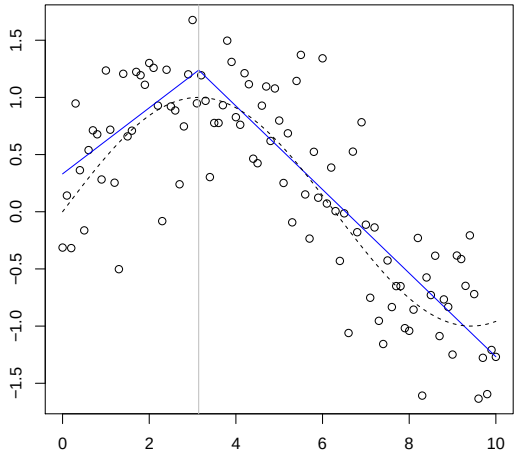
Linear Model and Loss Functions

But weights are unknown: use and iterative algorithm

```
> e <- residuals( lm(Y~X,data=db) )
> for( i in 1:100) {
+ W <- omega(e)
+ e <- residuals( lm(Y~X,data=db,weight(W)) )
+ }
```

Bagging Linear Models

```
> V=matrix(NA,100,251)
> for(i in 1:100){
+ ind <- sample(1:n,size=n,replace=TRUE)
+ V[i,] <- predict(lm(Y ~ X+ps(X),
+ data = db[ind,]),
+ newdata = data.frame(Y = u))}
```



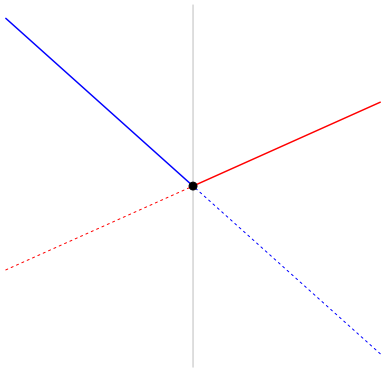
Regression Smoothers, *natura non facit saltus*

In statistical learning procedures, a key role is played by basis functions. We will see that it is common to assume that

$$m(\boldsymbol{x}) = \sum_{m=0}^{M} \beta_M h_m(\boldsymbol{x}),$$

where h_0 is usually a constant function and h_m defined basis functions.

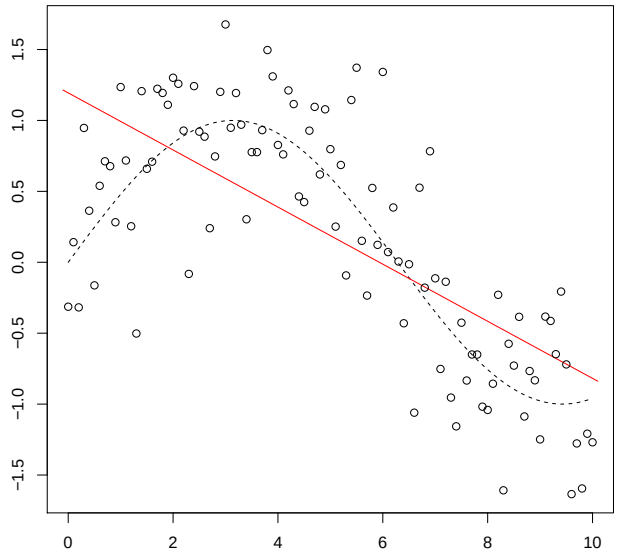
For instance, $h_m(x) = x^m$ for a polynomial expansion with a single predictor, or $h_m(x) = (x - s_m)_+$ for some knots s_m's (for linear splines, but one can consider quadratic or cubic ones).



Regression Smoothers: Polynomial Functions

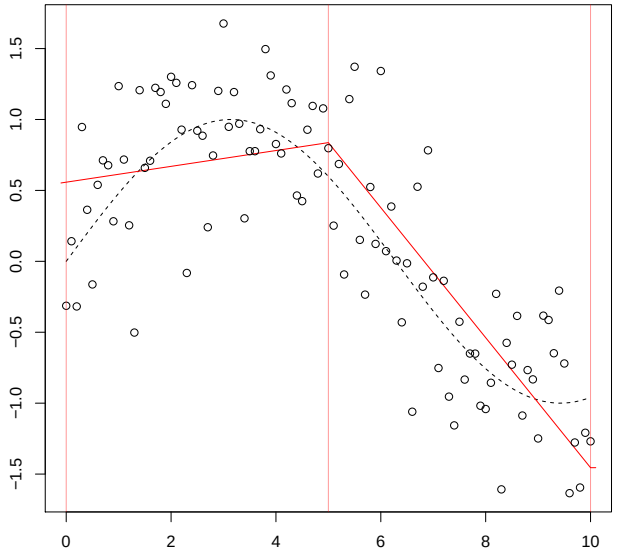
Stone-Weiestrass theorem every continuous function defined on a closed interval $[a, b]$ can be uniformly approximated as closely as desired by a polynomial function

```
> fit <- lm(Y ~ poly(X,k), data = db)
> predict(fit, newdata = data.frame(X=x))
```



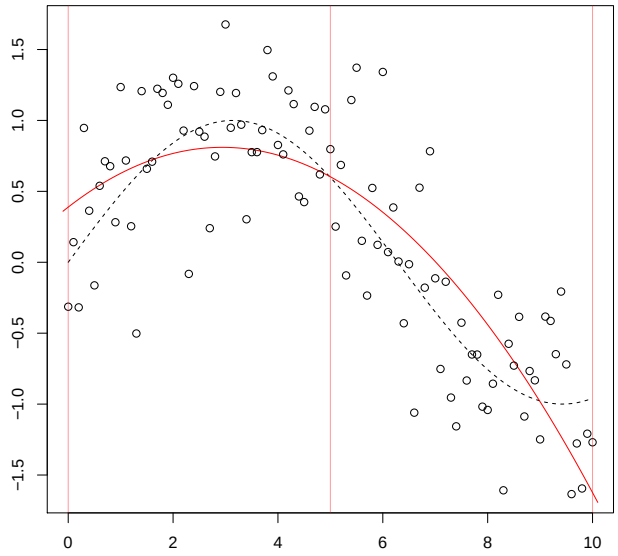
Regression Smoothers: Spline Functions

```
> fit <- lm(Y ~ bs(X,k,degree=1), data = db)
> predict(fit, newdata = data.frame(X=x))
```



Regression Smoothers: Spline Functions

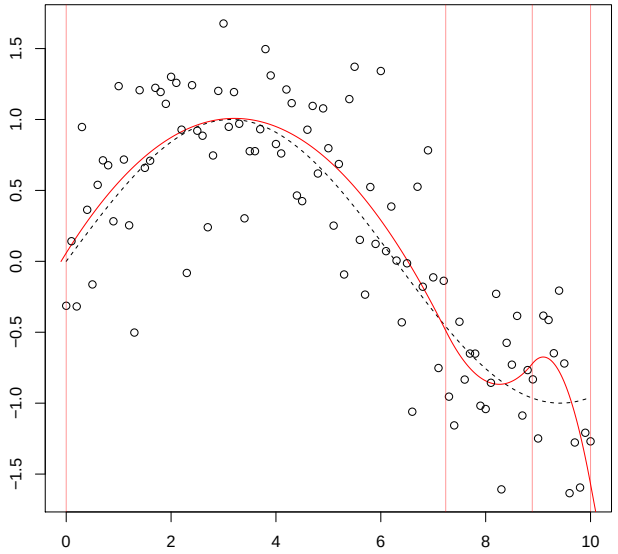
```
> fit <- lm(Y ~ bs(X,k,degree=2), data = db)
> predict(fit, newdata = data.frame(X=x))
```



see Generalized Additive Models.

Fixed Knots vs. Optimized Ones

```
> library(freeknotsplines)
> gen <- freelsgen(db$X, db$Y, degree=2,
    numknot=s)
> fit <- lm(Y ~ bs(X, gen$optknot, degree=2)
    , data = db)
> predict(fit, newdata = data.frame(X=x))
```



Penalized Smoothing

We have mentioned in the introduction that usually, we penalize a criteria (R^2 or log-likelihood) but it is also possible to penalize while fitting.

Heuristically, we have to minimuize the following objective function,

$$\text{objective}(\boldsymbol{\beta}) = \underbrace{\mathcal{L}(\boldsymbol{\beta})}_{\text{training loss}} + \underbrace{\mathcal{R}(\boldsymbol{\beta})}_{\text{regularization}}$$

The regression coefficient can be shrunk toward 0, making fitted values more homogeneous.

Consider a standard linear regression. The Ridge estimate is

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \left\{ \sum_{i=1}^{n} [y_i - \beta_0 - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}]^2 + \lambda \underbrace{\|\boldsymbol{\beta}\|_{\ell_2}}_{\mathbf{1}^\mathsf{T}\boldsymbol{\beta}^2} \right\}$$

for some tuning parameter λ.

Observe that $\widehat{\boldsymbol{\beta}} = [\boldsymbol{X}^\mathsf{T}\boldsymbol{X} + \lambda\mathbb{I}]^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}$.

We‘ inflate’ the $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ matrix by $\lambda\mathbb{I}$ so that it is positive definite whatever k, including $k > n$.

There is a Bayesian interpretation: if $\boldsymbol{\beta}$ has a $\mathcal{N}(\mathbf{0}, \tau^2\mathbb{I})$-prior and if resiuals are i.i.d. $\mathcal{N}(0, \sigma^2)$, then the posteriory mean (and median) $\widehat{\boldsymbol{\beta}}$ is the Ridge estimator, with $\lambda = \sigma^2/\tau^2$.

The Lasso estimate is

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} [y_i - \beta_0 - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}]^2 + \lambda \underbrace{\|\boldsymbol{\beta}\|_{\ell_1}}_{\mathbf{1}^\mathsf{T}|\boldsymbol{\beta}|} . \right\}$$

No explicit formulas, but simple nonlinear estimator (and quadratic programming routines are necessary).

The elastic net estimate is

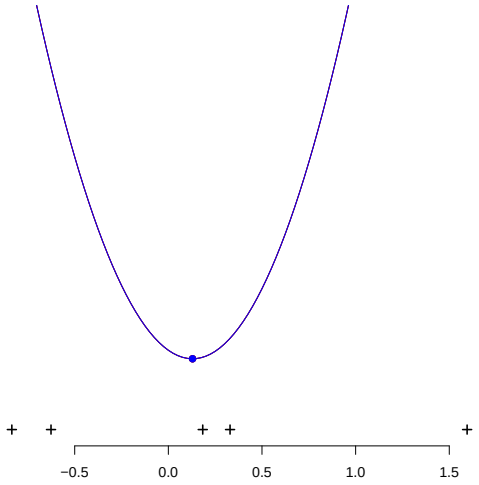
$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} [y_i - \beta_0 - \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta}]^2 + \lambda_1 \mathbf{1}^{\mathsf{T}}|\boldsymbol{\beta}| + \lambda_2 \mathbf{1}^{\mathsf{T}}\boldsymbol{\beta}^2. \right\}$$

See also LARS (Least Angle Regression) and Dantzig estimator.

Interpretation of Ridge and Lasso Estimators

Consider here the estimation of the mean,

- OLS, $\min\left\{\sum_{i=1}^{n}[y_i-m]^2\right\}$, $m^\star=\overline{y}=\dfrac{1}{n}\sum_{i=1}^{n}y_i$
- Ridge, $\min\left\{\sum_{i=1}^{n}[y_i-m]^2+\lambda m^2\right\}$,
- Lasso, $\min\left\{\sum_{i=1}^{n}[y_i-m]^2+\lambda|m|\right\}$,



Some thoughts about Tuning parameters

Regularization is a key issue in machine learning, to avoid overfitting.

In (traditional) econometrics are based on plug-in methods: see Silverman bandwith rule in Kernel density estimation,

$$h^\star = \left(\frac{4\widehat{\sigma}^5}{3n}\right) \sim 1.06\widehat{\sigma}n^{-1/5}.$$

In machine learning literature, use on out-of-sample cross-validation methods for choosing amount of regularization.

Optimal LASSO Penalty

Use cross validation, e.g. K-fold,

$$\widehat{\boldsymbol{\beta}}_{(-k)}(\lambda) = \text{argmin}\left(\{ \sum_{i \notin \mathcal{I}_k} [y_i - \boldsymbol{x}_i^\mathsf{T} \boldsymbol{\beta}]^2 + \lambda \sum_k |\beta_k| \right\}$$

then compute the sum or the squared errors,

$$Q_k(\lambda) = \sum_{i \notin \mathcal{I}_k} [y_i - \boldsymbol{x}_i^\mathsf{T} \widehat{\boldsymbol{\beta}}_{(-k)}(\lambda)]^2$$

and finally solve

$$\lambda^\star = \text{argmin}\left\{ \overline{Q}(\lambda) = \frac{1}{K} \sum_k Q_k(\lambda) \right\}$$

Note that this might overfit, so Hastie, Tibshiriani & Friedman (2009) suggest the largest λ such that

$$\overline{Q}(\lambda) \leq \overline{Q}(\lambda^\star) + \text{se}[\lambda^\star] \text{ with } \text{se}[\lambda]^2 = \frac{1}{K^2} \sum_{k=1}^{K} [Q_k(\lambda) - \overline{Q}(\lambda)]^2$$

Big Data, Oracle and Sparcity

Assume that k is large, and that $\boldsymbol{\beta} \in \mathbb{R}^k$ can be partitioned as $\boldsymbol{\beta} = (\boldsymbol{\beta}_{\text{imp}}, \boldsymbol{\beta}_{\text{non-imp}})$, as well as covariates $\boldsymbol{x} = (\boldsymbol{x}_{\text{imp}}, \boldsymbol{x}_{\text{non-imp}})$, with important and non-important variables, i.e. $\boldsymbol{\beta}_{\text{non-imp}} \sim \mathbf{0}$.

Goal : achieve variable selection and make inference of $\boldsymbol{\beta}_{\text{imp}}$

Oracle property of high dimensional model selection and estimation, see Fan and Li (2001). Only the oracle knows which variables are important...

If sample size is large enough $(n >> k_{\text{imp}} \left(1 + \log \frac{k}{k_{\text{imp}}}\right))$ we can do inference *as if* we knew which covariates were important: we can ignore the selection of covariates part, that is not relevant for the confidence intervals. This provides cover for ignoring the shrinkage and using regularstandard errors, see Athey & Imbens (2015).

Why Shrinkage Regression Estimates ?

Interesting for model selection (alternative to peanlized criterions) and to get a good balance between bias and variance.

In decision theory, an admissible decision rule is a rule for making a decision *such that there is not any other rule that is always better than it.*

When $k \geq 3$, ordinary least squares are not admissible, see the improvement by James–Stein estimator.

Regularization and Scalability

What if k is (extremely) large? never trust ols with more than five regressors (attributed to Zvi Griliches in Athey & Imbens (2015))

Use regularization techniques, see Ridge, Lasso, or subset selection

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} [y_i - \beta_0 - \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta}]^2 + \lambda \|\boldsymbol{\beta}\|_{\ell_0} \text{ where } \|\boldsymbol{\beta}\|_{\ell_0} = \sum_{k} \mathbf{1}(\beta_k \neq 0). \right\}$$

Penalization and Splines

In order to get a sufficiently smooth model, why not penalyse the sum of squares of errors,

$$\sum_{i=1}^{n}[y_i - m(\boldsymbol{x}_i)]^2 + \lambda \int [m''(\boldsymbol{t})]^2 d\boldsymbol{t}$$

for some tuning parameter λ. Consider some cubic spline basis, so that

$$m(\boldsymbol{x}) = \sum_{j=1}^{J} \theta_j N_j(\boldsymbol{x})$$

then the optimal expression for m is obtained using

$$\widehat{\boldsymbol{\theta}} = [\boldsymbol{N}^\mathsf{T}\boldsymbol{N} + \lambda\Omega]^{-1}\boldsymbol{N}^\mathsf{T}\boldsymbol{y}$$

where $\boldsymbol{N}_{i,j}$ is the matrix of $N_j(\boldsymbol{X}_i)$'s and $\Omega_{i,j} = \int N_i''(\boldsymbol{t})N_j''(\boldsymbol{t})d\boldsymbol{t}$

Smoothing with Multiple Regressors

Actually

$$\sum_{i=1}^{n}[y_i - m(\boldsymbol{x}_i)]^2 + \lambda \int [m''(\boldsymbol{t})]^2 d\boldsymbol{t}$$

is based on some multivariate penalty functional, e.g.

$$\int [m''(\boldsymbol{t})]^2 d\boldsymbol{t} = \int \left[\sum_i \left(\frac{\partial^2 m(\boldsymbol{t})}{\partial \boldsymbol{t}_i^2} \right)^2 + 2 \sum_{i,j} \left(\frac{\partial^2 m(\boldsymbol{t})}{\partial \boldsymbol{t}_i \partial \boldsymbol{t}_j} \right)^2 \right] d\boldsymbol{t}$$

Regression Trees

The partitioning is sequential, one covariate at a time (see adaptative neighbor estimation).

Start with $Q = \sum_{i=1}^{n} [y_i - \overline{y}]^2$

For covariate k and threshold t, split the data according to $\{x_{i,k} \leq t\}$ (L) or $\{x_{i,k} > t\}$ (R). Compute

$$\overline{y}_L = \frac{\sum_{i,x_{i,k} \leq t} y_i}{\sum_{i,x_{i,k} \leq t} 1} \text{ and } \overline{y}_R = \frac{\sum_{i,x_{i,k} > t} y_i}{\sum_{i,x_{i,k} > t} 1}$$

and let

$$m_i^{(k,t)} = \begin{cases} \overline{y}_L \text{ if } x_{i,k} \leq t \\ \overline{y}_R \text{ if } x_{i,k} > t \end{cases}$$

Regression Trees

Then compute $(k^\star, t^\star) = \text{argmin}\left\{\sum_{i=1}^{n}[y_i - m_i^{(k,t)}]^2\right\}$, and partition the space intro two subspace, whether $x_{k^\star} \leq t^\star$, or not.

Then repeat this procedure, and minimize

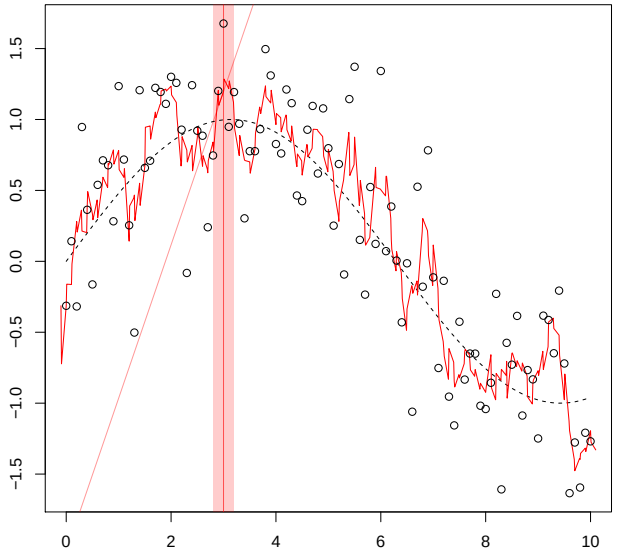
$$\sum_{i=1}^{n}[y_i - m_i]^2 + \lambda \cdot \#\{\text{leaves}\},$$

(cf LASSO).

One can also consider random forests with regression trees.

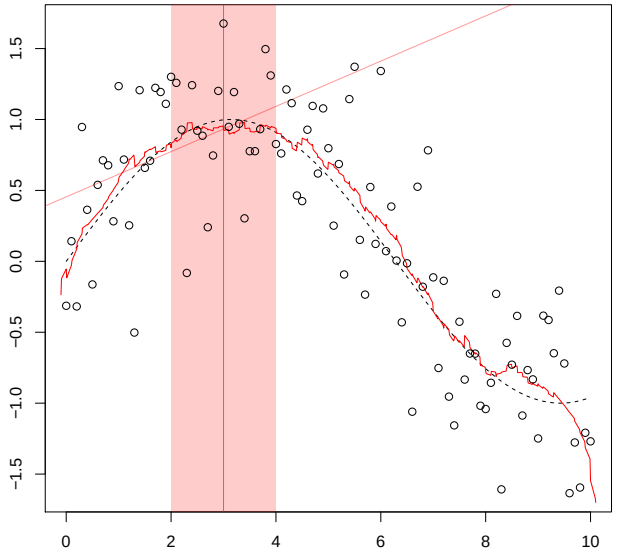
Local Regression

```
> W <- ( abs(db$X-x)<h )*1
> fit <- lm(Y ~ X, data = db, weights = W)
> predict(fit, newdata = data.frame(X=x))
```



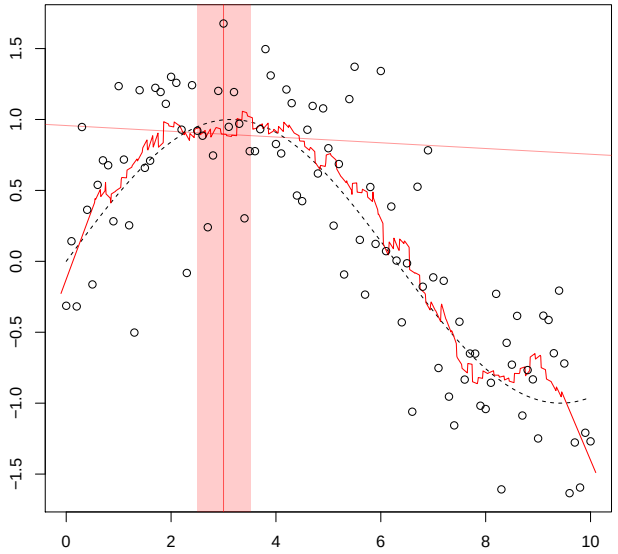
Local Regression

```
> W <- ( abs(db$X-x)<h )*1
> fit <- lm(Y ~ X, data = db, weights = W)
> predict(fit, newdata = data.frame(X=x))
```



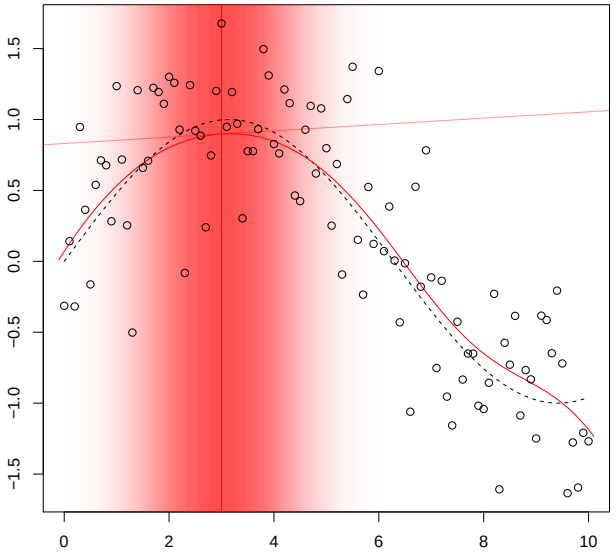
Local Regression : Nearest Neighbor

```
> W <- (rank( abs(db$X-x)<h ) <= k)*1
> fit <- lm(Y ~ X, data = db, weights = W)
> predict(fit, newdata = data.frame(X=x))
```



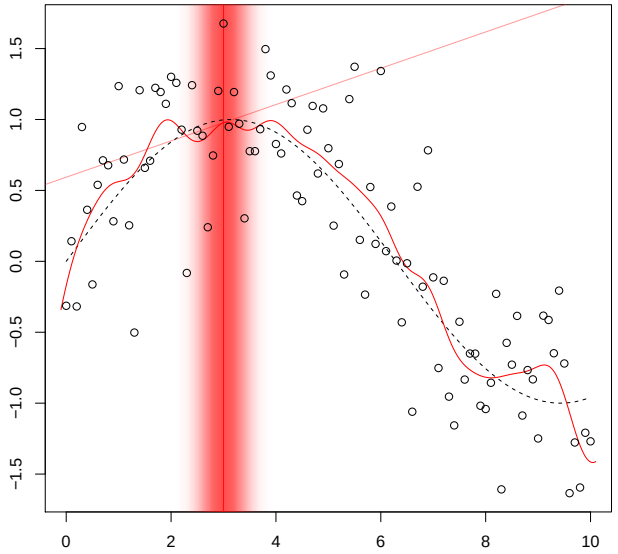
Local Regression : Kernel Based Smoothing

```
> library(KernSmooth)
> W <- dnorm( abs(db$X-x)<h )/h
> fit <- lm(Y ~ X, data = db, weights = W)
> predict(fit, newdata = data.frame(X=x))
> library(KernSmooth)
> library(sp)
```



Local Regression : Kernel Based Smoothing

```
> library(np)
> fit <- npreg(Y ~ X, data = db, bws = h,
+ ckertype = "gaussian")
> predict(fit, newdata = data.frame(X=x))
```



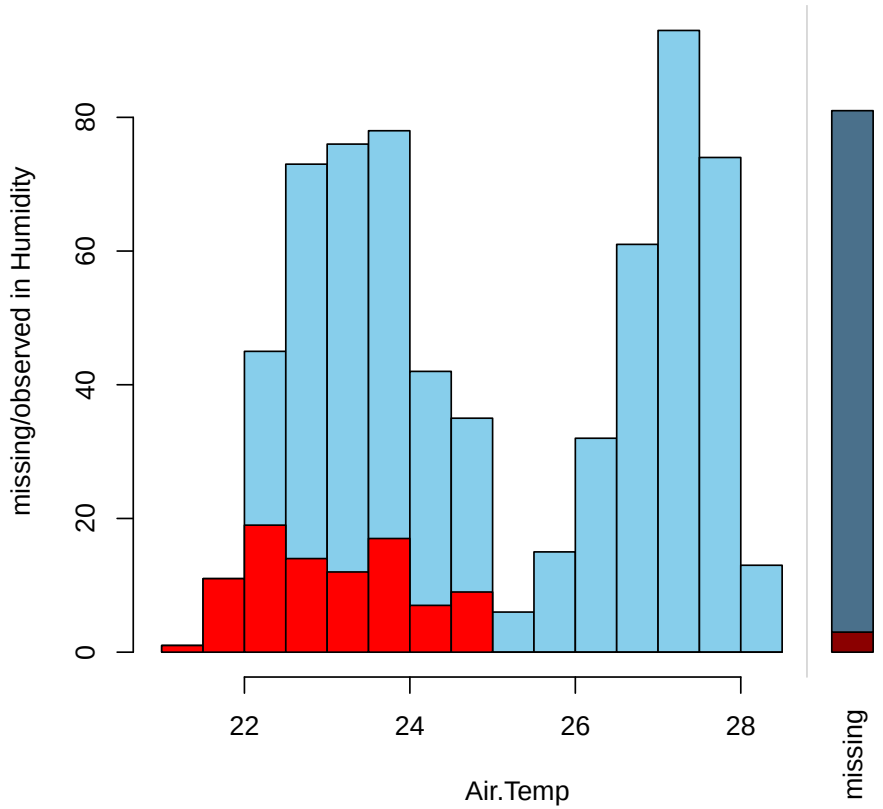
k-Nearest Neighbors and Imputation

Several packages deal with missing values, see e.g. VIM

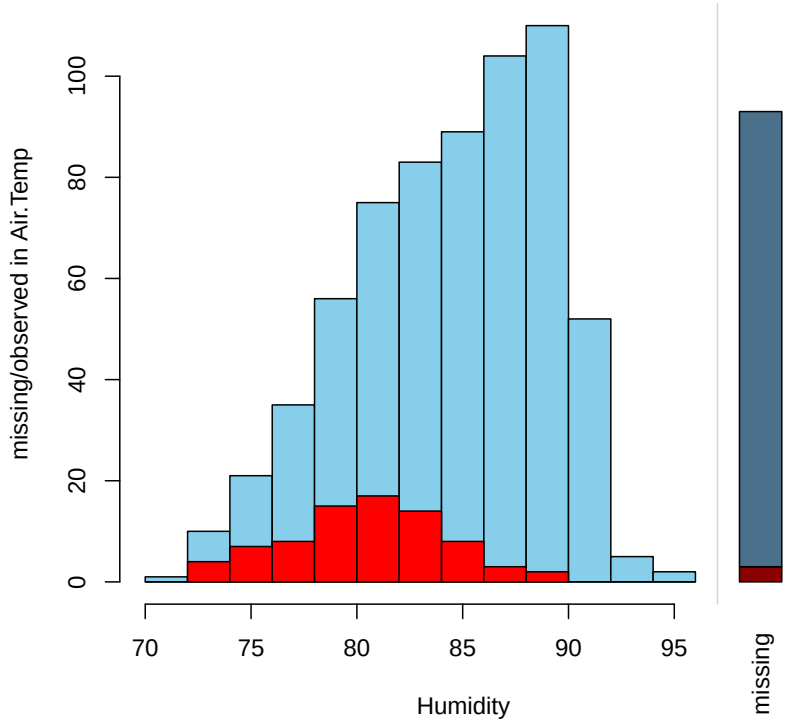
```
> library(VIM)
> data(tao)
> y <- tao[, c("Air.Temp", "Humidity")]
> summary(y)
    Air.Temp         Humidity
 Min.   :21.42   Min.   :71.60
 1st Qu.:23.26   1st Qu.:81.30
 Median :24.52   Median :85.20
 Mean   :25.03   Mean   :84.43
 3rd Qu.:27.08   3rd Qu.:88.10
 Max.   :28.50   Max.   :94.80
 NA's   :81      NA's   :93
```

Missing humidity giving the temperature

```
> y<-tao[,c("Air.Temp","Humidity")]
> histMiss(y)
```




```
> y<-tao[,c("Humidity","Air.Temp")]
> histMiss(y)
```



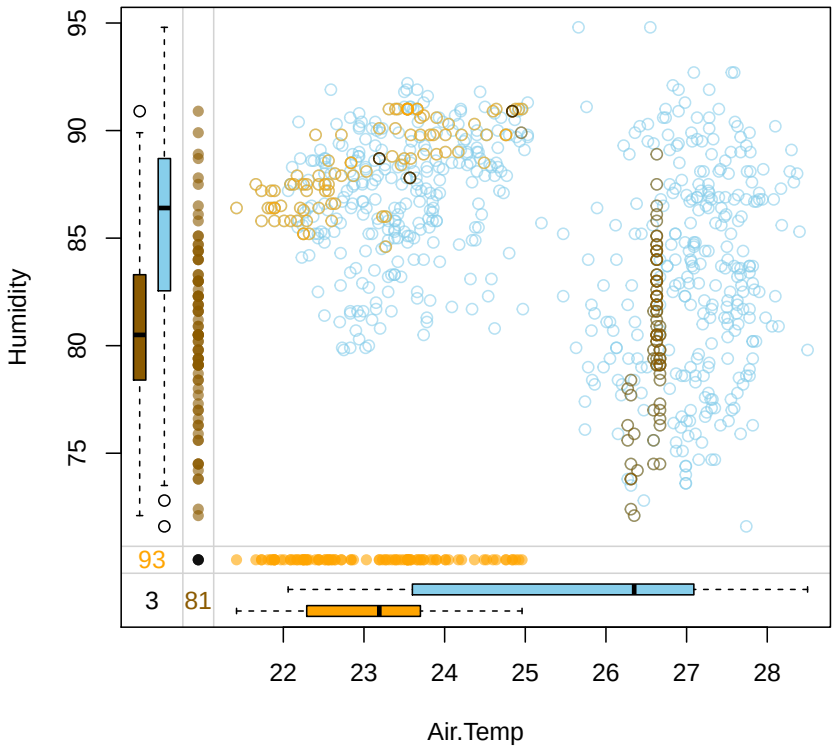
k-Nearest Neighbors and Imputation

This package countains a k-Neareast Neighbors algorithm for imputation

```
> tao_kNN <- kNN(tao, k = 5)
```

Imputation can be visualized using

```
vars <- c("Air.Temp","Humidity","
    Air.Temp_imp","Humidity_imp")
marginplot(tao_kNN[,vars],
    delimiter="imp", alpha=0.6)
```



From Linear to Generalized Linear Models

The (Gaussian) Linear Model and the logistic regression have been extended to the wide class of the exponential family,

$$f(y|\theta,\phi)=\exp\left(\frac{y\theta-b(\theta)}{a(\phi)}+c(y,\phi)\right),$$

where $a(\cdot)$, $b(\cdot)$ and $c(\cdot)$ are functions, θ is the natural - canonical - parameter and ϕ is a nuisance parameter.

The Gaussian distribution $\mathcal{N}(\mu,\sigma^2)$ belongs to this family

$$\underbrace{\theta=\mu}_{\theta\leftrightarrow\mathbb{E}(Y)},\quad \underbrace{\phi=\sigma^2}_{\phi\leftrightarrow\text{Var}(Y)},\quad a(\phi)=\phi,\quad b(\theta)=\theta^2/2$$

From Linear to Generalized Linear Models

The Bernoulli distribution $\mathcal{B}(p)$ belongs to this family

$$\underbrace{\theta = \log\frac{p}{1-p}}_{\theta = g_\star(\mathbb{E}(Y))}, \ a(\phi) = 1, \ b(\theta) = \log(1+\exp(\theta)), \text{ and } \phi = 1$$

where the $g_\star(\cdot)$ is some link function (here the logistic transformation): the canonical link.

Canonical links are

```
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

From Linear to Generalized Linear Models

Observe that

$$\mu = \mathbb{E}(Y) = b'(\theta) \text{ and } \text{Var}(Y) = b''(\theta) \cdot \phi = \underbrace{b''([b']^{-1}(\mu)) \cdot \phi}_{\text{variance function } V(\mu)}$$

$\longrightarrow$ distributions are characterized by this variance function, e.g. $V(\mu) = 1$ for the Gaussian family (homoscedastic models), $V(\mu) = \mu$ for the Poisson and $V(\mu) = \mu^2$ for the Gamma distribution, $V(\mu) = \mu^3$ for the inverse-Gaussian family.

Note that $g_\star(\cdot) = [b']^{-1}(\cdot)$ is the canonical link.

Tweedie (1984) suggested a power-type variance function $V(\mu) = \mu^\gamma \cdot \phi$. When $\gamma \in [1, 2]$, then Y has a compound Poisson distribution with Gamma jumps.

```
> library(tweedie)
```

From the Exponential Family to GLM's

So far, there no regression model. Assume that

$$f(y_i|\theta_i,\phi)=\exp\left(\frac{y_i\theta_i-b(\theta_i)}{a(\phi)}+c(y_i,\phi)\right) \text{ where } \theta_i=g_\star^{-1}(g(\boldsymbol{x}_i^\top\boldsymbol{\beta}))$$

so that the log-likelihood is

$$\mathcal{L}(\boldsymbol{\theta},\phi|\boldsymbol{y})=\prod_{i=1}^n f(y_i|\theta_i,\phi)=\exp\left(\frac{\sum_{i=1}^n y_i\theta_i-\sum_{i=1}^n b(\theta_i)}{a(\phi)}+\sum_{i=1}^n c(y_i,\phi)\right).$$

To derive the first order condition, observe that we can write

$$\frac{\partial\log\mathcal{L}(\boldsymbol{\theta},\phi|\boldsymbol{y}_i)}{\partial\boldsymbol{\beta}_j}=\boldsymbol{\omega}_{i,j}\boldsymbol{x}_{i,j}[y_i-\mu_i]$$

for some $\boldsymbol{\omega}_{i,j}$ (see e.g. Müller (2004)) which are simple when $g_\star=g$.

From the Exponential Family to GLM's

The first order conditions can be writen

$$\boldsymbol{X}^\top \boldsymbol{W}^{-1}[\boldsymbol{y}-\boldsymbol{\mu}]=\boldsymbol{0}$$

which are first order conditions for a weighted linear regression model.

As for the logistic regression, $\boldsymbol{W}$ depends on unkown $\boldsymbol{\beta}$'s : use an iterative algorithm

1. Set $\widehat{\boldsymbol{\mu}}_0=\boldsymbol{y}$, $\boldsymbol{\theta}_0=g(\widehat{\boldsymbol{\mu}}_0)$ and

$$\boldsymbol{z}_0=\boldsymbol{\theta}_0+(\boldsymbol{y}-\widehat{\boldsymbol{\mu}}_0)g'(\widehat{\boldsymbol{\mu}}_0).$$

Define $\boldsymbol{W}_0=\text{diag}[g'(\widehat{\boldsymbol{\mu}}_0)^2\text{Var}(\widehat{\boldsymbol{y}})]$ and fit a (weighted) lineare regression of $\boldsymbol{Z}_0$ on $\boldsymbol{X}$, i.e.

$$\widehat{\boldsymbol{\beta}}_1=[\boldsymbol{X}^\top\boldsymbol{W}_0^{-1}\boldsymbol{X}]^{-1}\boldsymbol{X}^\top\boldsymbol{W}_0^{-1}\boldsymbol{z}_0$$

2. Set $\widehat{\boldsymbol{\mu}}_k=\boldsymbol{X}\widehat{\boldsymbol{\beta}}_k$, $\boldsymbol{\theta}_k=g(\widehat{\boldsymbol{\mu}}_k)$ and

$$\boldsymbol{z}_k=\boldsymbol{\theta}_k+(\boldsymbol{y}-\widehat{\boldsymbol{\mu}}_k)g'(\widehat{\boldsymbol{\mu}}_k).$$

From the Exponential Family to GLM's

Define $\boldsymbol{W}_k = \text{diag}[g'(\widehat{\boldsymbol{\mu}}_k)^2 \text{Var}(\widehat{\boldsymbol{y}})]$ and fit a (weighted) lineare regression of $\boldsymbol{Z}_k$ on $\boldsymbol{X}$, i.e.

$$\widehat{\boldsymbol{\beta}}_{k+1} = [\boldsymbol{X}^\top \boldsymbol{W}_k^{-1} \boldsymbol{X}]^{-1} \boldsymbol{X}^\top \boldsymbol{W}_k^{-1} \boldsymbol{Z}_k$$

and loop... until changes in $\widehat{\boldsymbol{\beta}}_{k+1}$ are (sufficiently) small.

Under some technical conditions, we can prove that $\widehat{\boldsymbol{\beta}} \xrightarrow{\mathbb{P}} \boldsymbol{\beta}$ and

$$\sqrt{n}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \xrightarrow{\mathcal{L}} \mathcal{N}(\boldsymbol{0}, I(\boldsymbol{\beta})^{-1}).$$

where numerically $I(\boldsymbol{\beta}) = \phi \cdot [\boldsymbol{X}^\top \boldsymbol{W}_\infty^{-1} \boldsymbol{X}])$.

From the Exponential Family to GLM's

We estimate (see linear regression estimation) ϕ by

$$\widehat{\phi} = \frac{1}{n - \dim(\boldsymbol{X})} \sum_{i=1}^{n} \boldsymbol{\omega}_{i,i} \frac{[\boldsymbol{y}_i - \widehat{\boldsymbol{\mu}}_i]}{\text{Var}(\widehat{\boldsymbol{\mu}}_i)}$$

This asymptotic expression can be used to derive confidence intervals, or tests. But is might be a poor approximation when n is small. See use of boostrap in claims reserving.

Those are theorerical results: in practice, the algorithm may fail to converge

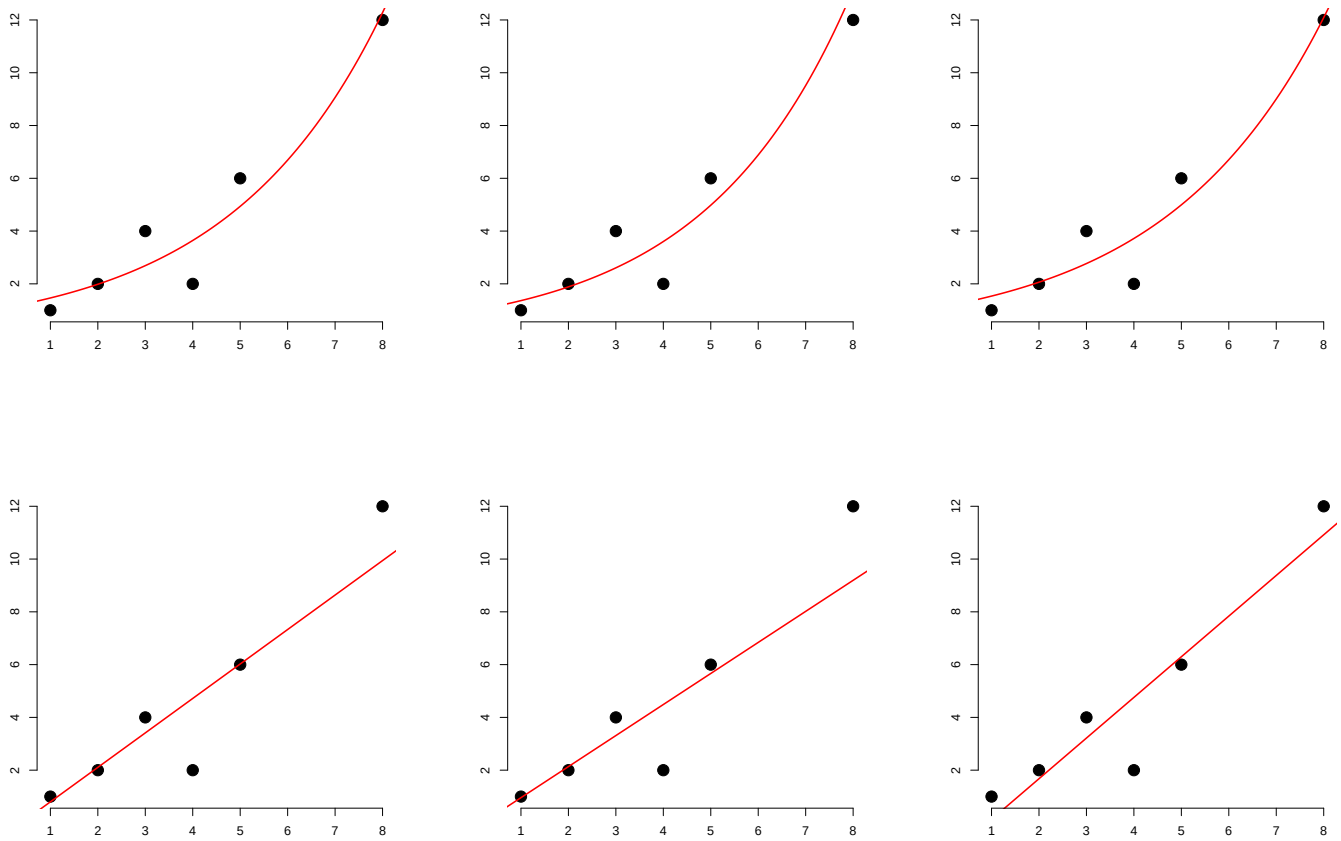
GLM's outside the Exponential Family?

Actually, it is possible to consider more general distributions, see Yee (2014))

```
> library(VGAM)
> vglm(y ~ x, family = Makeham)
> vglm(y ~ x, family = Gompertz)
> vglm(y ~ x, family = Erlang)
> vglm(y ~ x, family = Frechet)
> vglm(y ~ x, family = pareto1(location=100))
```

Those functions can also be used for a multivariate response $\boldsymbol{y}$

GLM: Link and Distribution



GLM: Distribution?

From a computational point of view, the Poisson regression is not (really) related to the Poisson distribution.

Here we solve the first order conditions (or normal equations)

$$\sum_i [Y_i - \exp(\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta})]X_{i,j} = 0 \ \forall j$$

with unconstraint $\boldsymbol{\beta}$, using Fisher's scoring technique $\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \boldsymbol{H}_k^{-1}\nabla_k$

$$\text{where } \boldsymbol{H}_k = -\sum_i \exp(\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta}_k)\boldsymbol{X}_i\boldsymbol{X}_i^\mathsf{T} \text{ and } \nabla_k = \sum_i \boldsymbol{X}_i^\mathsf{T}[Y_i - \exp(\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta}_k)]$$

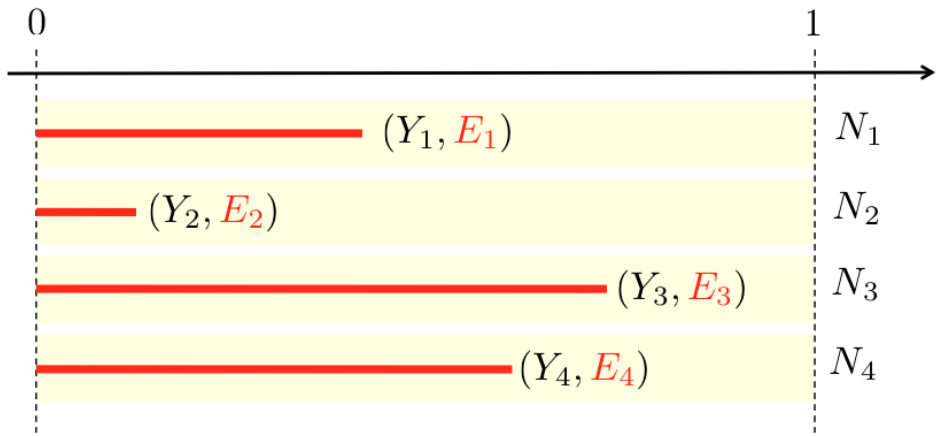
$\longrightarrow$ There is no assumption here about $Y \in \mathbb{N}$: it is possible to run a Poisson regression on non-integers.

The Exposure and (Annual) Claim Frequency

In General Insurance, we should predict blueyearly claims frequency. Let N_i denote the number of claims over one year for contrat i.

We did observe only the contract for a period of time E_i

Let Y_i denote the observed number of claims, over period $[0, E_i]$.



The Exposure and (Annual) Claim Frequency

Assuming that claims occurence is driven by a Poisson process of intensity λ, if $N_1 \sim \mathcal{P}(\lambda)$, then $Y_i \sim \mathcal{P}(\lambda \cdot E_i)$.

$$\mathcal{L}(\lambda, \boldsymbol{Y}, \boldsymbol{E}) = \prod_{i=1}^{n} \frac{e^{-\lambda E_i} [\lambda E_i]_i^Y}{Y_i!}$$

the first order condition is

$$\frac{\partial}{\partial \lambda} \log \mathcal{L}(\lambda, \boldsymbol{Y}, \boldsymbol{E}) = -\sum_{i=1}^{n} E_i + \frac{1}{\lambda} \sum_{i=1}^{n} Y_i = 0$$

for

$$\widehat{\lambda} = \frac{\sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} E_i} = \sum_{i=1}^{n} \omega_i \frac{Y_i}{E_i} \text{ where } \omega_i = \frac{E_i}{\sum_{i=1}^{n} E_i}$$

The Exposure and (Annual) Claim Frequency

Assume that

$$Y_i \sim \mathcal{P}(\lambda_i \cdot E_i) \text{ where } \lambda_i = \exp[\boldsymbol{X}_i'\boldsymbol{\beta}].$$

Here $\mathbb{E}(Y_i|\boldsymbol{X}_i) = \text{Var}(Y_i|\boldsymbol{X}_i) = \lambda_i = \exp[\boldsymbol{X}_i'\boldsymbol{\beta} + \log E_i]$.

$$\log \mathcal{L}(\boldsymbol{\beta}; \boldsymbol{Y}) = \sum_{i=1}^{n} Y_i \cdot [\boldsymbol{X}_i'\boldsymbol{\beta} + \log E_i] - (\exp[\boldsymbol{X}_i'\boldsymbol{\beta}] + \log E_i) - \log(Y_i!)$$

```
> model <- glm(y~x,offset=log(E),family=poisson)
> model <- glm(y~x + offset(log(E)),family=poisson)
```

Taking into account the exposure in other models is difficult...

Boosting

Boosting is a machine learning ensemble meta-algorithm for reducing bias primarily and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. (source: Wikipedia)

The heuristics is simple: we consider an iterative process where we keep modeling the errors.

Fit model for $\boldsymbol{y}$, $m_1(\cdot)$ from $\boldsymbol{y}$ and $\boldsymbol{X}$, and compute the error, $\boldsymbol{\varepsilon}_1 = \boldsymbol{y} - m_1(\boldsymbol{X})$.

Fit model for $\boldsymbol{\varepsilon}_1$, $m_2(\cdot)$ from $\boldsymbol{\varepsilon}_1$ and $\boldsymbol{X}$, and compute the error, $\boldsymbol{\varepsilon}_2 = \boldsymbol{\varepsilon}_1 - m_2(\boldsymbol{X})$, etc. Then set

$$m(\cdot) = \underbrace{m_1(\cdot)}_{\sim y} + \underbrace{m_2(\cdot)}_{\sim \varepsilon_1} + \underbrace{m_3(\cdot)}_{\sim \varepsilon_2} + \cdots + \underbrace{m_k(\cdot)}_{\sim \varepsilon_{k-1}}$$

Boosting

With (very) general notations, we want to solve

$$m^\star = \text{argmin}\{\mathbb{E}[\ell(Y, m(\boldsymbol{X}))]\}$$

for some loss function ℓ.

It is an iterative procedure: assume that at some step k we have an estimator $m_k(\boldsymbol{X})$. Why not constructing a new model that might improve our model,

$$m_{k+1}(\boldsymbol{X}) = m_k(\boldsymbol{X}) + h(\boldsymbol{X}).$$

What $h(\cdot)$ could be?

Boosting

In a perfect world, $h(\boldsymbol{X}) = \boldsymbol{y} - m_k(\boldsymbol{X})$, which can be interpreted as a residual.

Note that this residual is the gradient of $\frac{1}{2}[y - m(\boldsymbol{x})]^2$

A gradient descent is based on Taylor expansion

$$\underbrace{f(\boldsymbol{x}_k)}_{\langle f, \boldsymbol{x}_k\rangle} \sim \underbrace{f(\boldsymbol{x}_{k-1})}_{\langle f, \boldsymbol{x}_{k-1}\rangle} + \underbrace{(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})}_{\alpha} \underbrace{\nabla f(\boldsymbol{x}_{k-1})}_{\langle \nabla f, \boldsymbol{x}_{k-1}\rangle}$$

But here, it is different. We claim we can write

$$\underbrace{f_k(\boldsymbol{x})}_{\langle f_k, \boldsymbol{x}\rangle} \sim \underbrace{f_{k-1}(\boldsymbol{x})}_{\langle f_{k-1}, \boldsymbol{x}\rangle} + \underbrace{(f_k - f_{k-1})}_{\beta} \underbrace{?}_{\langle f_{k-1}, \nabla \boldsymbol{x}\rangle}$$

where ? is interpreted as a 'gradient'.

Boosting

Here, f_k is a $\mathbb{R}^d \to \mathbb{R}$ function, so the gradient should be in such a (big) functional space $\to$ want to approximate that function.

$$m_k(\boldsymbol{x}) = m_{k-1}(\boldsymbol{x}) + \underset{f \in \mathcal{F}}{\text{argmin}} \left\{ \sum_{i=1}^{n} \ell(Y_i, m_{k-1}(\boldsymbol{x}) + f(\boldsymbol{x})) \right\}$$

where $f \in \mathcal{F}$ means that we seek in a class of weak learner functions.

If learner are two strong, the first loop leads to some fixed point, and there is no learning procedure, see linear regression $y = \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta} + \varepsilon$. Since $\varepsilon \perp \boldsymbol{x}$ we cannot learn from the residuals.

Boosting with some Shrinkage

Consider here some quadratic loss function.

In order to make sure that we learn weakly, we can use some shrinkage parameter ν (or collection of parameters ν_j) so that

$$\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] = m(\boldsymbol{x}) \sim m_M(\boldsymbol{x}) = \sum_{j=1}^{M} \nu_j h_j(\boldsymbol{x})$$

The problem is always the same. At stage j, we should solve

$$\min_{h(\cdot)} \left\{ \sum_{i=1}^{n} [\underbrace{y_i - m_{j-1}(\boldsymbol{x}_i)}_{\varepsilon_{i,j-1}} - h(\boldsymbol{x}_i)]^2 \right\}$$

Boosting with some Shrinkage

The algorithm is then

- start with some (simple) model $\boldsymbol{y} = h_1(\boldsymbol{x})$
- compute the residuals (including ν), $\boldsymbol{\varepsilon}_1 = \boldsymbol{y} - \nu h_1(\boldsymbol{x})$

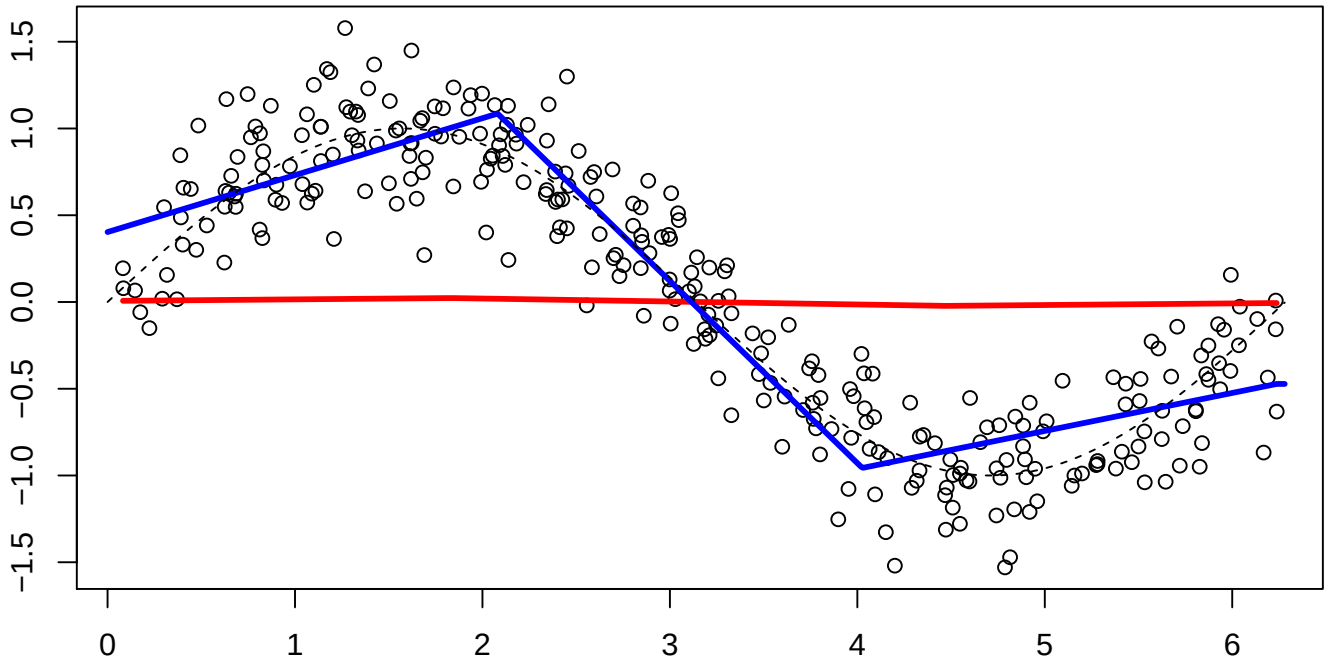
and at step j,

- consider some (simple) model $\boldsymbol{\varepsilon}_j = h_j(\boldsymbol{x})$
- compute the residuals (including ν), $\boldsymbol{\varepsilon}_{j+1} = \boldsymbol{\varepsilon}_j - \nu h_j(\boldsymbol{x})$

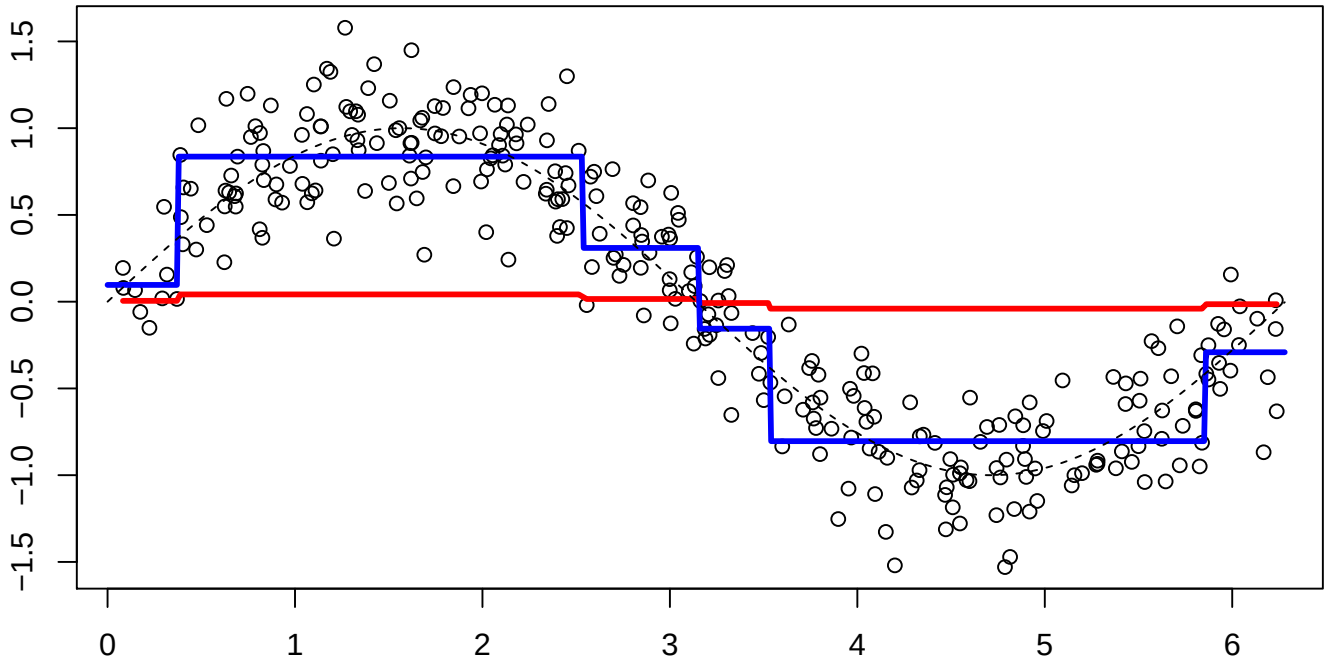
and loop. And set finally

$$\widehat{\boldsymbol{y}} = \sum_{j=1}^{M} \nu h_j(\boldsymbol{x})$$

Boosting with Piecewise Linear Spline Functions



Boosting with Trees (Stump Functions)



Boosting for Classification

Still seek $m^\star(\cdot) = \text{argmin}\{\mathbb{E}[\ell(Y, m(\boldsymbol{X}))]\}$

Here $y \in \{-1, +1\}$, and use $\ell(y, m(\boldsymbol{x})) = e^{-y \cdot m(\boldsymbol{x})}$: AdaBoot algorithm.

Note that

$$\mathbb{P}[Y = +1 | \boldsymbol{X} = \boldsymbol{x}] = \frac{1}{1 + e^{2m^\star \boldsymbol{x}}}$$

cf probit transform... Can be seen as iteration on weights. At step k solve

$$\underset{h(\cdot)}{\text{argmin}} \left\{ \sum_{i=1}^{n} \underbrace{e^{y_i \cdot m_k(\boldsymbol{x}_i)}}_{\omega_{i,k}} \cdot e^{y_i \cdot h(\boldsymbol{x}_i)} \right\}$$

Exponential distribution, deviance, loss function, residuals, etc

- Gaussian distribution $\longleftrightarrow$ ℓ_2 loss function

Deviance is $\sum_{i=1}^{n}(y_i - m(\boldsymbol{x}_i))^2$, with gradient $\widehat{\varepsilon}_i = y_i - m(\boldsymbol{x}_i)$

- Laplace distribution $\longleftrightarrow$ ℓ_1 loss function

Deviance is $\sum_{i=1}^{n}|y_i - m(\boldsymbol{x}_i))|$, with gradient $\widehat{\varepsilon}_i = \text{sign}(y_i - m(\boldsymbol{x}_i))$

Exponential distribution, deviance, loss function, residuals, etc

- Bernoullli $\{-1,+1\}$ distribution $\longleftrightarrow$ ℓ_{adaboost} loss function

Deviance is $\sum_{i=1}^{n} e^{-y_i m(\boldsymbol{x}_i)}$, with gradient $\widehat{\varepsilon}_i = -y_i e^{-[y_i] m(\boldsymbol{x}_i)}$

- Bernoullli $\{0,1\}$ distribution

Deviance $2\sum_{i=1}^{n}[y_i \cdot \log\left(\frac{y_i}{m(\boldsymbol{x}_i)}\right)(1-y_i)\log\left(\frac{1-y_i}{1-m(\boldsymbol{x}_i)}\right)$ with gradient

$$\widehat{\varepsilon}_i = y_i - \frac{\exp[m(\boldsymbol{x}_i)]}{1+\exp[m(\boldsymbol{x}_i)]}$$

- Poisson distribution

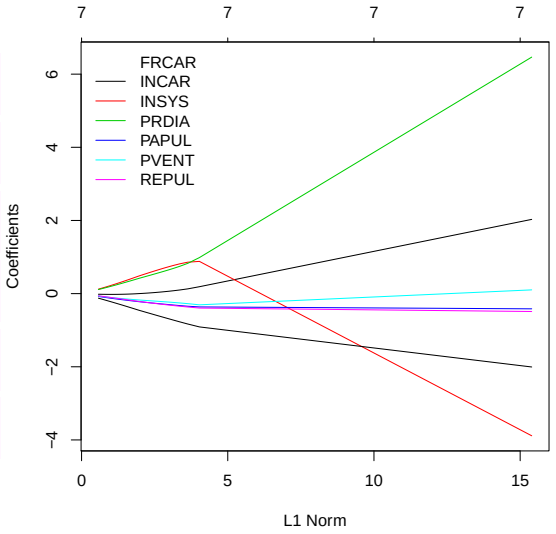
Deviance $2\sum_{i=1}^{n}\left(y_i \cdot \log\left(\frac{y_i}{m(\boldsymbol{x}_i)}\right) - [y_i - m(\boldsymbol{x}_i)]\right)$ with gradient $\widehat{\varepsilon}_i = \frac{y_i - m(\boldsymbol{x}_i)}{\sqrt{m(\boldsymbol{x}_i)}}$

Regularized GLM

In Regularized GLMs, we introduced a penalty in the loss function (the deviance), see e.g. ℓ_1 regularized logistic regression

$$\max\left\{\sum_{i=1}^{n}\Big(y_i[\beta_0+\boldsymbol{x}_i^\top\boldsymbol{\beta}-\log[1+e^{\beta_0+\boldsymbol{x}_i^\top\boldsymbol{\beta}}]]\Big)-\lambda\sum_{j=1}^{k}|\beta_j|\right\}$$

```
> library(glmnet)
> y <- myocarde$PRONO
> x <- as.matrix(myocarde[,1:7])
> glm_ridge <- glmnet(x, y, alpha=0, lambda=seq
    (0,2,by=.01), family="binomial")
> plot(lm_ridge)
```



Collective vs. Individual Model

Consider a Tweedie distribution, with variance function power $p \in (0,1)$, mean μ and scale parameter ϕ, then it is a compound Poisson model,

- $N \sim \mathcal{P}(\lambda)$ with $\lambda = \dfrac{\phi\mu^{2-p}}{2-p}$
- $Y_i \sim \mathcal{G}(\alpha,\beta)$ with $\alpha = -\dfrac{p-2}{p-1}$ and $\beta = \dfrac{\phi\mu^{1-p}}{p-1}$

Consversely, consider a compound Poisson model $N \sim \mathcal{P}(\lambda)$ and $Y_i \sim \mathcal{G}(\alpha,\beta)$,

- variance function power is $p = \dfrac{\alpha+2}{\alpha+1}$
- mean is $\mu = \dfrac{\lambda\alpha}{\beta}$
- scale parameter is $\phi = \dfrac{[\lambda\alpha]^{\frac{\alpha+2}{\alpha+1}-1}\beta^{2-\frac{\alpha+2}{\alpha+1}}}{\alpha+1}$

seems to be equivalent... but it's not.

Collective vs. Individual Model

In the context of regression

$$N_i \sim \mathcal{P}(\lambda_i) \text{ with } \lambda_i = \exp[\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta}_\lambda]$$

$$Y_{j,i} \sim \mathcal{G}(\mu_i, \phi) \text{ with } \mu_i = \exp[\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta}_\mu]$$

Then $S_i = Y_{1,i} + \cdots + Y_{N,i}$ has a Tweedie distribution

- variance function power is $p = \dfrac{\phi+2}{\phi+1}$
- mean is $\lambda_i\mu_i$
- scale parameter is $\dfrac{\lambda_i^{\frac{1}{\phi+1}-1}}{\mu_i^{\frac{\phi}{\phi+1}}}\left(\dfrac{\phi}{1+\phi}\right)$

There are $1+2\dim(\boldsymbol{X})$ degrees of freedom.

Collective vs. Individual Model

Note that the scale parameter should not depend on i. A Tweedie regression is

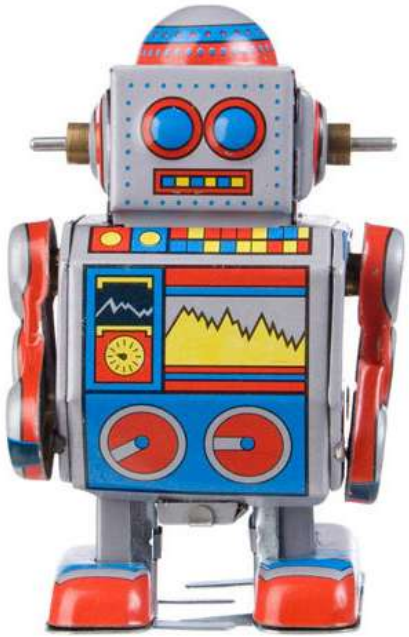
- variance function power is $p = \in (0, 1)$
- mean is $\mu_i = \exp[\boldsymbol{X}_i^\mathsf{T}\boldsymbol{\beta}_{\text{Tweedie}}]$
- scale parameter is ϕ

There are $2 + \dim(\boldsymbol{X})$ degrees of freedom.

Note that oone can easily boost a Tweedie model

```
> library(TDboost)
```

Part 4.
Model Choice, Feature Selection, etc.



AIC, BIC

AIC and BIC are both maximum likelihood estimate driven and penalize useless parameters(to avoid overfitting)

$$AIC = -2\log[\text{likelihood}] + 2k \text{ and } BIC = -2\log[\text{likelihood}] + \log(n)k$$

AIC focus on overfit, while BIC depends on n so it might also avoid underfit

BIC penalize complexity more than AIC does.

Minimizing AIC $\Leftrightarrow$ minimizing cross-validation value, Stone (1977).

Minimizing BIC $\Leftrightarrow$ k-fold leave-out cross-validation, Shao (1997), with $k = n[1 - (\log n - 1)]$

$\longrightarrow$ used in econometric stepwise procedures

Cross-Validation

Formally, the leave-one-out cross validation is based on

$$CV = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \widehat{m}_{-i}(\boldsymbol{x}_i))$$

where $\widehat{m}_{-i}$ is obtained by fitting the model on the sample where observation i has been dropped.

The Generalized cross-validation, for a quadratic loss function, is defined as

$$GCV = \frac{1}{n}\sum_{i=1}^{n} \left[\frac{y_i - \widehat{m}_{-i}(\boldsymbol{x}_i)}{1 - \text{trace}(\boldsymbol{S})/n}\right]^2$$

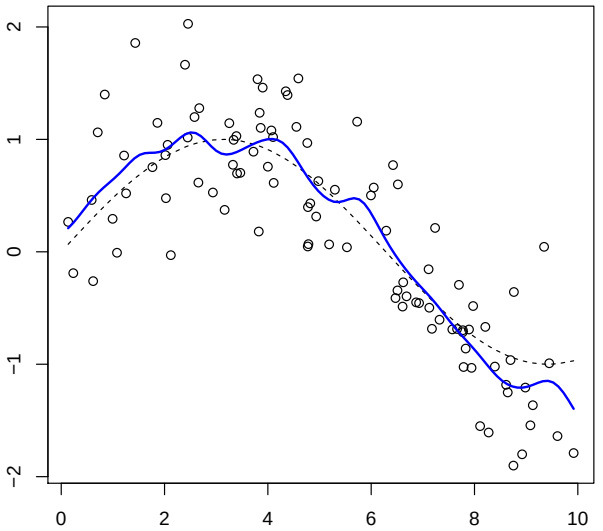
Cross-Validation for kernel based local regression

Econometric approach

Define $\widehat{m}(x) = \widehat{\beta}_0^{[x]} + \widehat{\beta}_1^{[x]} x$ with

$$(\widehat{\beta}_0^{[x]}, \widehat{\beta}_1^{[x]}) = \underset{(\beta_0,\beta_1)}{\text{argmin}} \left\{ \sum_{i=1}^{n} \omega_{h^\star}^{[x]} [y_i - (\beta_0 + \beta_1 x_i)]^2 \right\}$$

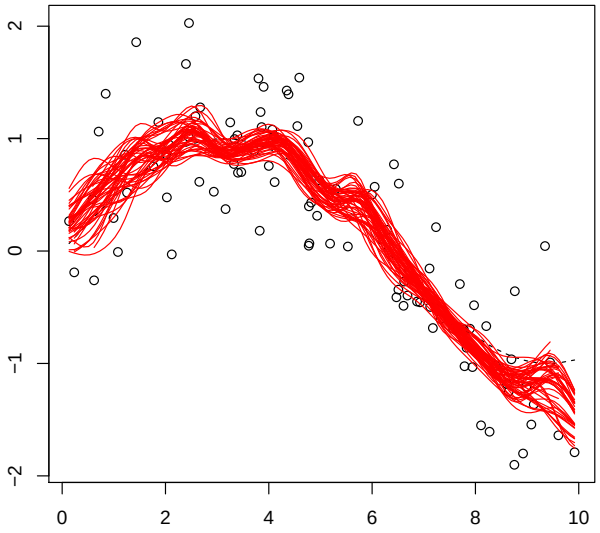
where $h^\star$ is given by some rule of thumb (see previous discussion).

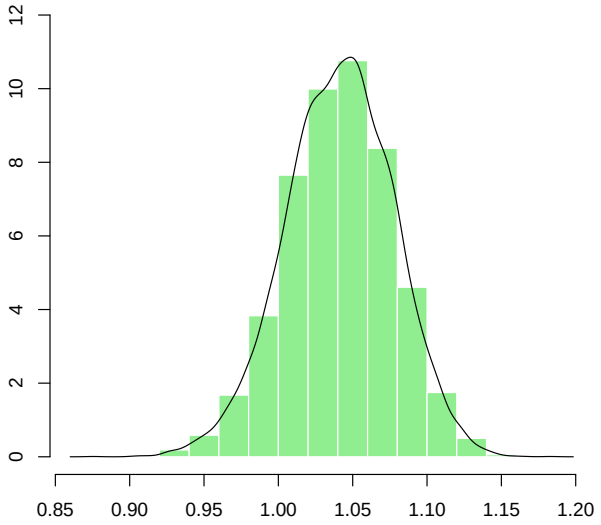


Cross-Validation for kernel based local regression

Bootstrap based approach

Use bootstrap samples, compute $h_b^\star$, and get $\widehat{m}_b(x)$'s.





Cross-Validation for kernel based local regression

Statistical learning approach (Cross Validation (leave-one-out))

Given $j \in \{1, \cdots, n\}$, given h, solve

$$(\widehat{\beta}_0^{[(i),h]}, \widehat{\beta}_1^{[(i),h]}) = \underset{(\beta_0,\beta_1)}{\text{argmin}} \left\{ \sum_{j \neq i} \omega_h^{(i)} [Y_j - (\beta_0 + \beta_1 x_j)]^2 \right\}$$

and compute $\widehat{m}_{(i)}^{[h]}(x_i) = \widehat{\beta}_0^{[(i),h]} + \widehat{\beta}_1^{[(i),h]} x_i$. Define

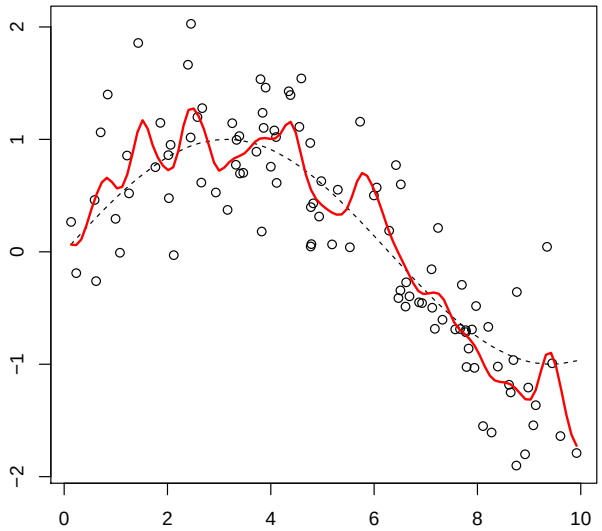
$$\text{mse}(h) = \sum_{i=1}^{n} [y_i - \widehat{m}_{(i)}^{[h]}(x_i)]^2$$

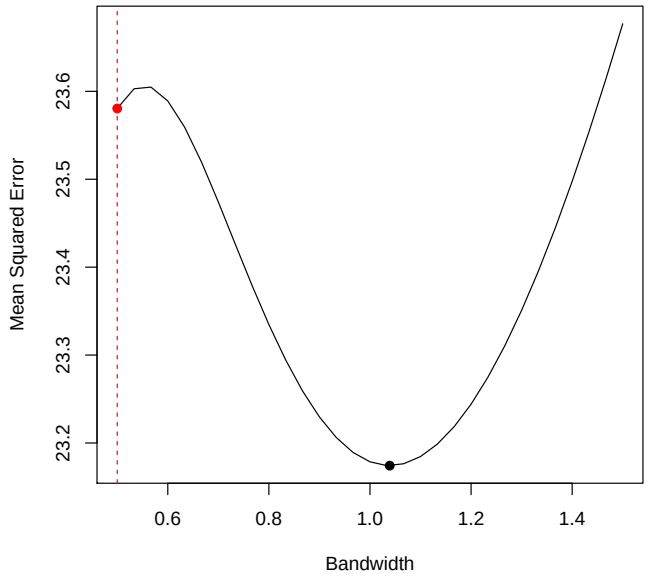
and set $h^\star = \text{argmin}\{\text{mse}(h)\}$.

Then compute $\widehat{m}(x) = \widehat{\beta}_0^{[x]} + \widehat{\beta}_1^{[x]} x$ with

$$(\widehat{\beta}_0^{[x]}, \widehat{\beta}_1^{[x]}) = \underset{(\beta_0,\beta_1)}{\text{argmin}} \left\{ \sum_{i=1}^{n} \omega_{h^\star}^{[x]} [y_i - (\beta_0 + \beta_1 x_i)]^2 \right\}$$

Cross-Validation for kernel based local regression





Cross-Validation for kernel based local regression

Statistical learning approach (Cross Validation (k-fold))

Given $\mathcal{I} \in \{1, \cdots, n\}$, given h, solve

$$(\widehat{\beta}_0^{[(\mathcal{I}),h]}, \widehat{\beta}_1^{[x_i,h]}) = \underset{(\beta_0,\beta_1)}{\text{argmin}} \left\{ \sum_{j \notin \mathcal{I}} \omega_h^{(\mathcal{I})} [y_j - (\beta_0 + \beta_1 x_j)]^2 \right\}$$

and compute $\widehat{m}_{(\mathcal{I})}^{[h]}(x_i) = \widehat{\beta}_0^{[(i),h]} + \widehat{\beta}_1^{[(i),h]} x_i$, $\forall i \in \mathcal{I}$. Define

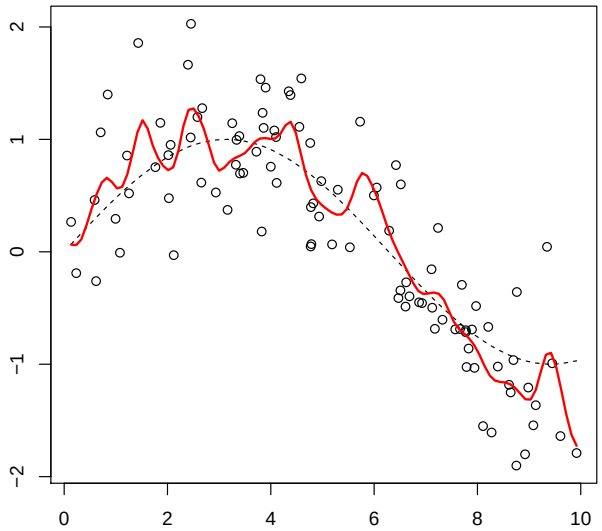
$$\text{mse}(h) = \sum_{\mathcal{I}} \sum_{i \in \mathcal{I}} [y_i - \widehat{m}_{(\mathcal{I})}^{[h]}(x_i)]^2$$

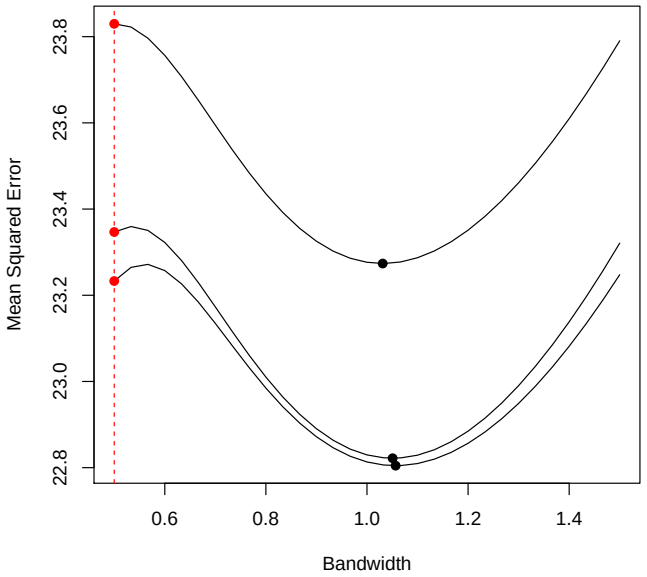
and set $h^\star = \text{argmin}\{\text{mse}(h)\}$.

Then compute $\widehat{m}(x) = \widehat{\beta}_0^{[x]} + \widehat{\beta}_1^{[x]} x$ with

$$(\widehat{\beta}_0^{[x]}, \widehat{\beta}_1^{[x]}) = \underset{(\beta_0,\beta_1)}{\text{argmin}} \left\{ \sum_{i=1}^{n} \omega_{h^\star}^{[x]} [y_i - (\beta_0 + \beta_1 x_i)]^2 \right\}$$

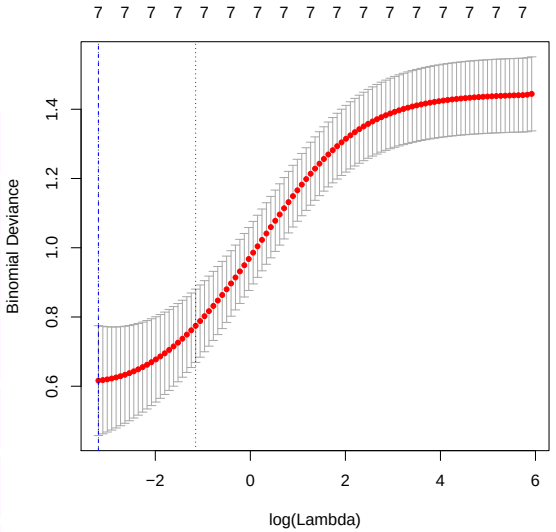
Cross-Validation for kernel based local regression

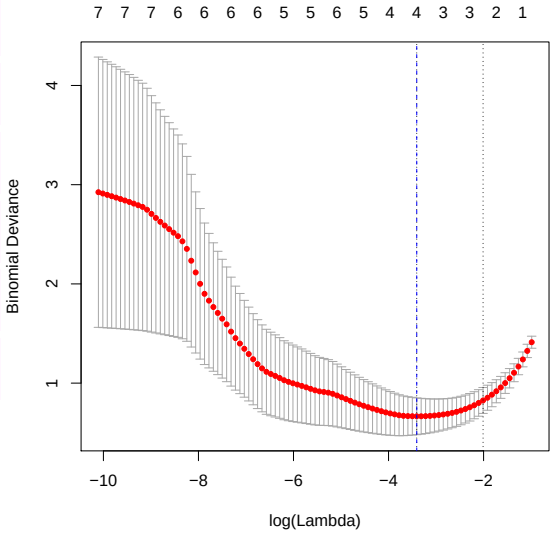




Cross-Validation for Ridge & Lasso

```
> library(glmnet)
> y <- myocarde$PRONO
> x <- as.matrix(myocarde[,1:7])
> cvfit <- cv.glmnet(x, y, alpha=0, family =
+ "binomial", type = "auc", nlambda = 100)
> cvfit$lambda.min
[1] 0.0408752
> plot(cvfit)
> cvfit <- cv.glmnet(x, y, alpha=1, family =
+ "binomial", type = "auc", nlambda = 100)
> cvfit$lambda.min
[1] 0.03315514
> plot(cvfit)
```



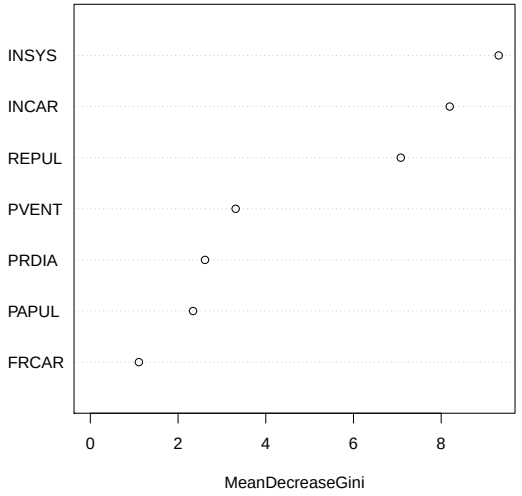


Variable Importance for Trees

Given some random forest with M trees, set $I(X_k) = \frac{1}{M}\sum_m \sum_t \frac{N_t}{N}\Delta i(t)$

where the first sum is over all trees, and the second one is over all nodes where the split is done based on variable X_k.

```
> RF=randomForest(PRONO ~ .,data = myocarde)
> varImpPlot(RF,main="")
> importance(RF)
      MeanDecreaseGini
FRCAR         1.107222
INCAR         8.194572
INSYS         9.311138
PRDIA         2.614261
PAPUL         2.341335
PVENT         3.313113
REPUL         7.078838
```

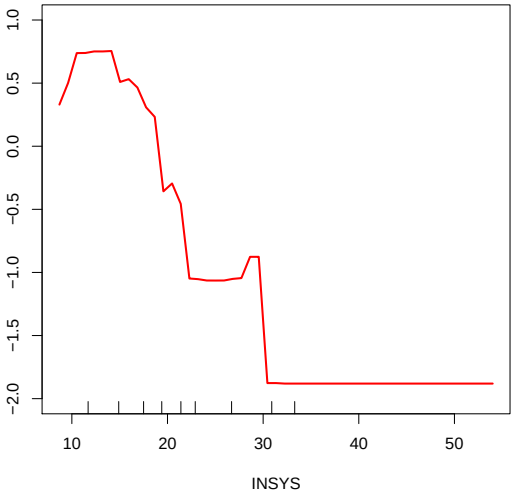


Partial Response Plots

One can also compute Partial Response Plots,

$$x \mapsto \frac{1}{n}\sum_{i=1}^{n} \widehat{\mathbb{E}}[Y|X_k = x, \boldsymbol{X}_{i,(k)} = \boldsymbol{x}_{i,(k)}]$$

```
> importanceOrder <- order(-RF$importance)
> names <- rownames(RF$importance)[importanceOrder
    ]
> for (name in names)
+ partialPlot(RF, myocarde, eval(name), col="red",
     main="", xlab=name)
```

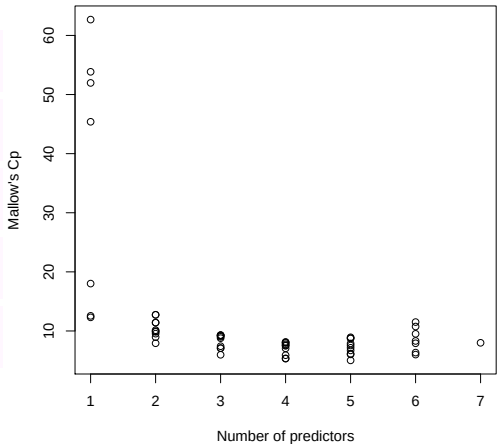


Feature Selection

Use Mallow's C_p, from Mallow (1974) on all subset of predictors, in a regression

$$C_p = \frac{1}{S^2}\sum_{i=1}^{n}[Y_i - \widehat{Y}_i]^2 - n + 2p,$$

```
> library(leaps)
> y <- as.numeric(train_myocarde$PRONO)
> x <- data.frame(train_myocarde[,-8])
> selec = leaps(x, y, method="Cp")
> plot(selec$size-1, selec$Cp)
```

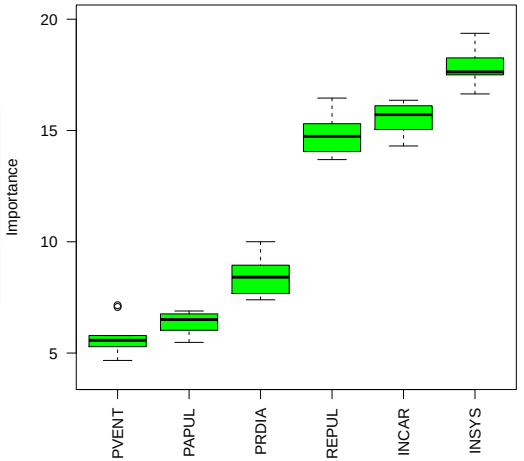


Feature Selection

Use random forest algorithm, removing some features at each iterations (the less relevent ones).

The algorithm uses shadow attributes (obtained from existing features by shuffling the values).

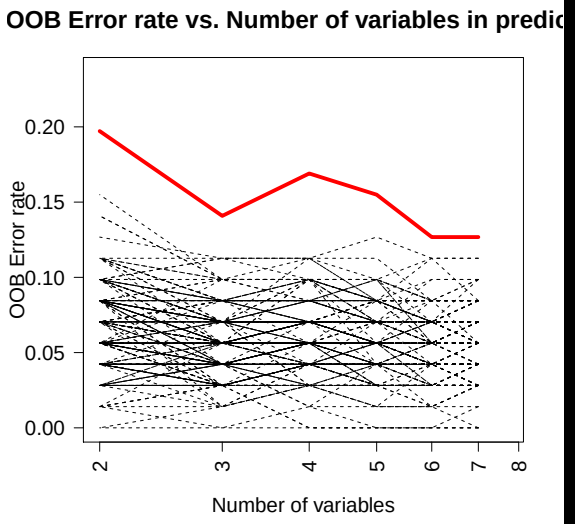
```
> library(Boreta)
> B <- Boruta(PRONO ~ ., data=myocarde, ntree=500)
> plot(B)
```



Feature Selection

Use random forests, and variable importance plots

```
> library(varSelRFBoot)
> X <- as.matrix(myocarde[,1:7])
> Y <- as.factor(myocarde$PRONO)
> library(randomForest)
> rf <- randomForest(X, Y, ntree = 200, importance
    = TRUE)
> V <- randomVarImpsRF(X, Y, rf, usingCluster =
  FALSE)
> VB <- varSelRFBoot(X, Y,usingCluster = FALSE)
> plot(VB)
```

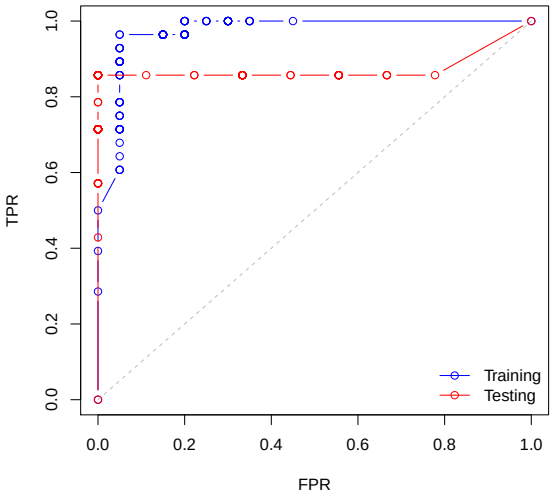


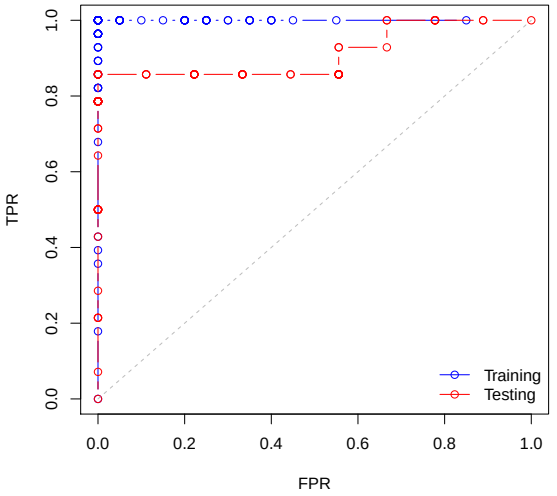
ROC (and beyond)

	$Y=0$	$Y=1$		prvalence
$\widehat{Y}=0$	true negative N_{00}	false negative (type II) N_{01}	negative predictive value NPV$=\frac{N_{00}}{N_{0\cdot}}$	false omission rate FOR$=\frac{N_{01}}{N_{0\cdot}}$
$\widehat{Y}=1$	false positive (type I) N_{10}	true positive N_{11}	false discovery rate FDR$=\frac{N_{10}}{N_{1\cdot}}$	positive predictive value PPV$=\frac{N_{11}}{N_{1\cdot}}$ (precision)
negative likelihood ratio LR-=FNR/TNR	true negative rate TNR$=\frac{N_{00}}{N_{\cdot 0}}$ (specificity)	false negative rate FNR$=\frac{N_{01}}{N_{\cdot 1}}$		
positive likelihood ratio LR+=TPR/FPR	false positive rate FPR$=\frac{N_{10}}{N_{\cdot 0}}$ (fall out)	true positive rate TPR$=\frac{N_{11}}{N_{\cdot 1}}$ (sensitivity)		
diagnostic odds ratio = LR+/LR-				

Comparing Classifiers: ROC Curves

```
> library(randomForest)
> fit=randomForest(PRONO~.,data=train_myocarde)
> train_Y=(train_myocarde$PRONO=="Survival")
> test_Y =(test_myocarde$PRONO=="Survival")
> train_S=predict(fit,type="prob",newdata=train
    _myocarde)[,2]
> test_S=predict(fit,type="prob",newdata=test_
    myocarde)[,2]
> vp=seq(0,1,length=101)
> roc_train=t(Vectorize(function(u) roc.curve(
    train_Y,train_S,s=u))(vp))
> roc_test=t(Vectorize(function(u) roc.curve(
    test_Y,test_S,s=u))(vp))
> plot(roc_train,type="b",col="blue",xlim=0:1,
    ylim=0:1)
```





Comparing Classifiers: ROC Curves

The Area Under the Curve, AUC, can be interpreted as the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one, see Swets, Dawes & Monahan (2000)

Many other quantities can be computed, see

```
> library(hmeasures)
> HMeasure(Y,S)$metrics[,1:5]
Class labels have been switched from (DECES,SURVIE) to (0,1)
                H       Gini        AUC       AUCH         KS
scores 0.7323154 0.8834154 0.9417077 0.9568966 0.8144499
```

with the H-measure (see hmeasure), Gini and AUC, as well as the area under the convex hull (AUCH).

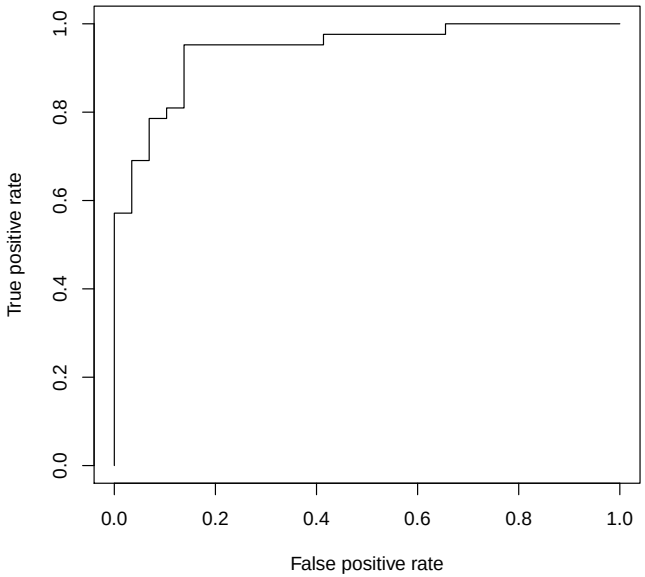
Comparing Classifiers: ROC Curves

Consider our previous logistic regression (on heart attacks)

```
> logistic <- glm(PRONO~. , data=myocarde,
    family=binomial)
> Y <- myocarde$PRONO
> S <- predict(logistic, type="response")
```

For a standard ROC curve

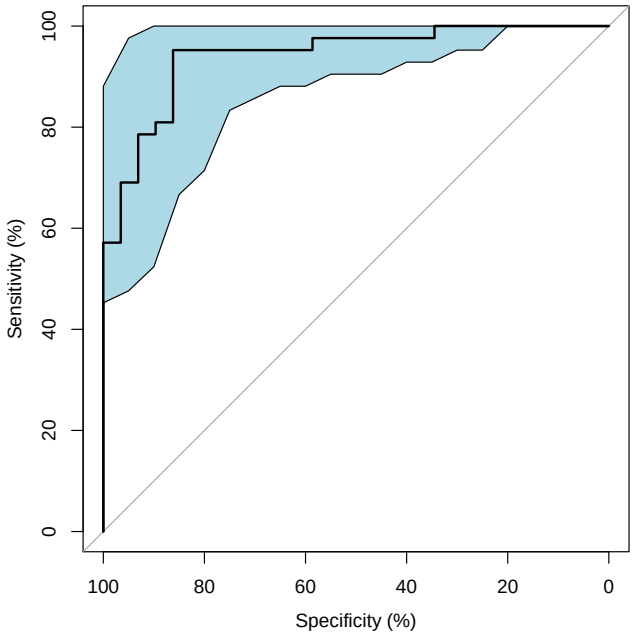
```
> library(ROCR)
> pred <- prediction(S,Y)
> perf <- performance(pred, "tpr", "fpr")
> plot(perf)
```



Comparing Classifiers: ROC Curves

On can get econfidence bands (obtained using bootstrap procedures)

```
> library(pROC)
> roc <- plot.roc(Y, S, main="", percent=TRUE,
    ci=TRUE)
> roc.se <- ci.se(roc,specificities=seq(0, 100,
     5))
> plot(roc.se, type="shape", col="light blue")
```



see also for Gains and Lift curves

```
> library(gains)
```

Comparing Classifiers: Accuracy and Kappa

Kappa statistic κ compares an Observed Accuracy with an Expected Accuracy (random chance), see Landis & Koch (1977).

	$Y=0$	$Y=1$	
$\widehat{Y}=0$	TN	FN	TN+FN
$\widehat{Y}=1$	FP	TP	FP+TP
	TN+FP	FN+TP	n

See also Obsersed and Random Confusion Tables

	$Y=0$	$Y=1$	
$\widehat{Y}=0$	25	3	28
$\widehat{Y}=1$	4	39	43
	29	42	71

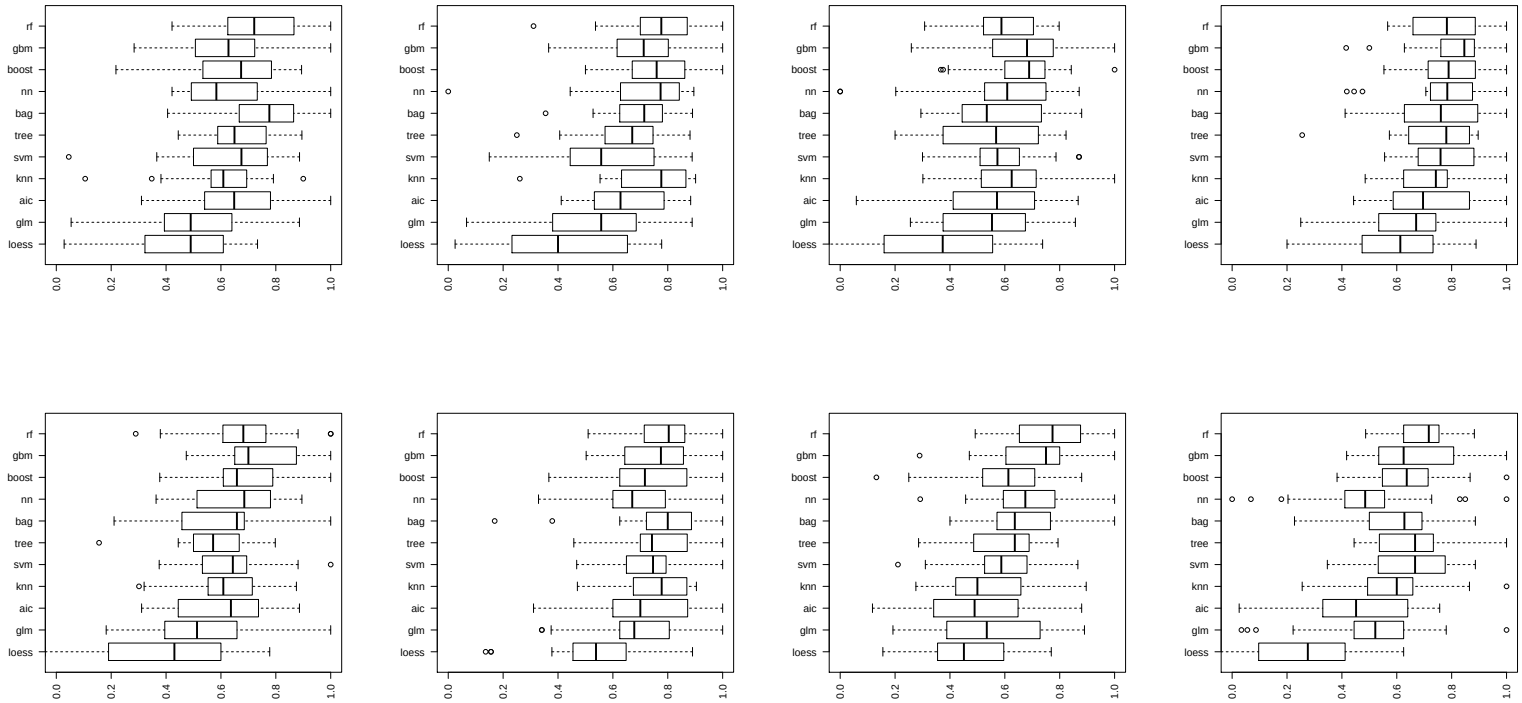
	$Y=0$	$Y=1$	
$\widehat{Y}=0$	11.44	16.56	28
$\widehat{Y}=1$	17.56	25.44	43
	29	42	71

$$\text{total accuracy} = \frac{TP+TN}{n} \sim 90.14\%$$

$$\text{random accuracy} = \frac{[TN+FP]\cdot[TP+FN]+[TP+FP]\cdot[TN+FN]}{n^2} \sim 51.93\%$$

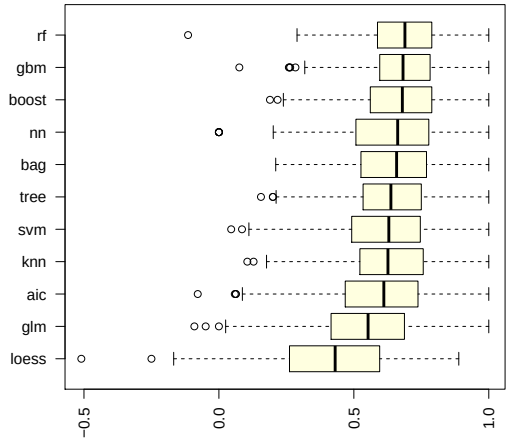
$$\kappa = \frac{\text{total accuracy} - \text{random accuracy}}{1 - \text{random accuracy}} \sim 79.48\%$$

Comparing Models on the myocarde Dataset



Comparing Models on the myocarde Dataset

If we average over all training samples

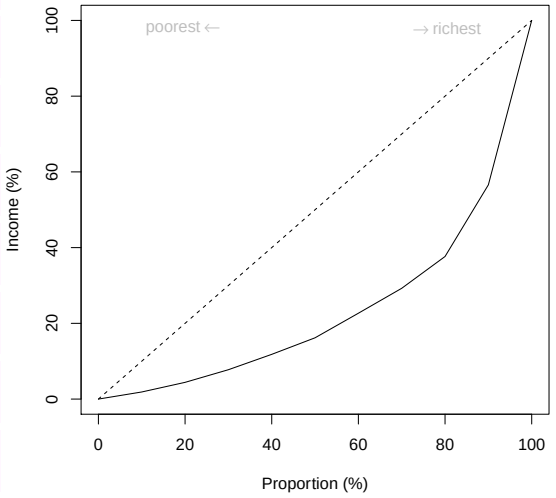


Gini and Lorenz Type Curves

Consider an ordered sample $\{y_1, \cdots, y_n\}$ of incomes, with $y_1 \leq y_2 \leq \cdots \leq y_n$, then Lorenz curve is

$$\{F_i, L_i\} \text{ with } F_i = \frac{i}{n} \text{ and } L_i = \frac{\sum_{j=1}^{i} y_j}{\sum_{j=1}^{n} y_j}$$

```
> L <- function(u,vary="income"){
+   base=base[order(base[,vary],decreasing=FALSE),]
+   base$cum=(1:nrow(base))/nrow(base)
+   return(sum(base[base$cum<=u,vary])/
+            sum(base[,vary]))}
> vu <- seq(0,1,length=nrow(base)+1)
> vv <- Vectorize(function(u) L(u))(vu)
> plot(vu, vv, type = "l")
```



Gini and Lorenz Type Curves

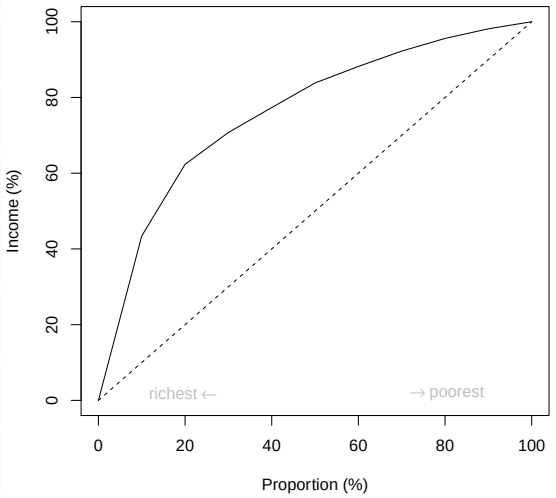
The theoretical curve, given a distribution F, is

$$u \mapsto L(u) = \frac{\int_{-\infty}^{F^{-1}(u)} t dF(t)}{\int_{-\infty}^{+\infty} t dF(t)}$$

see Gastwirth (1972).

One can also sort them from high to low incomes, $y_1 \geq y_2 \geq \cdots \geq y_n$

```
> L <- function(u,vary="income"){
+   base=base[order(base[,vary],decreasing=TRUE),]
+   base$cum=(1:nrow(base))/nrow(base)
+   return(sum(base[base$cum<=u,vary])/
+            sum(base[,vary]))}
> vu <- seq(0,1,length=nrow(base)+1)
> vv <- Vectorize(function(u) L(u))(vu)
> plot(vu, vv, type = "l")
```



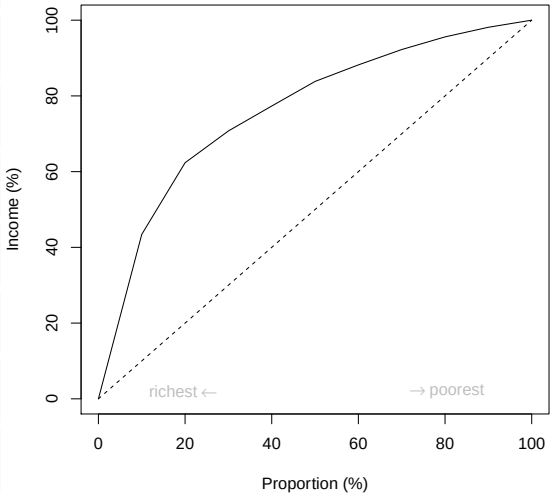
Gini and Lorenz Type Curves

We want to compare two regression models, $\widehat{m}_1(\cdot)$ and $\widehat{m}_2(\cdot)$, in the context of insurance pricing, see Frees, Meyers & Cummins (2014). We have observed losses y_i and premiums $\widehat{m}(\boldsymbol{x}_i)$. Consider an ordered sample by the model,

$$\widehat{m}(\boldsymbol{x}_1) \geq \widehat{m}(\boldsymbol{x}_2) \geq \cdots \geq \widehat{m}(\boldsymbol{x}_n)$$

then plot $\{F_i, L_i\}$ with $F_i = \dfrac{i}{n}$ and $L_i = \dfrac{\sum_{j=1}^{i} y_j}{\sum_{j=1}^{n} y_j}$

```
> L <- function(u,varx="premium",vary="losses"){
+   base=base[order(base[,varx],decreasing=TRUE),]
+   base$cum=(1:nrow(base))/nrow(base)
+   return(sum(base[base$cum<=u,vary])/
+            sum(base[,vary]))}
> vu <- seq(0,1,length=nrow(base)+1)
> vv <- Vectorize(function(u) L(u))(vu)
> plot(vu, vv, type = "l")
```



Model Selection, or Aggregation?

We have k models, $\widehat{m}_1(\boldsymbol{x}), \cdots, \widehat{m}_k(\boldsymbol{x})$ for the same y-variable, that can be trees, vsm, regression, etc.

Instead of selecting the best model, why not consider

$$\widehat{m}^\star(\boldsymbol{x}) = \sum_{\kappa=1}^{k} \omega_\kappa \widehat{m}_\kappa(\boldsymbol{x})$$

for some weights ω_κ.

New problem: solve $\min_{\omega_1,\cdots,\omega_k} \left\{ \sum_{i=1}^{n} \ell\left(y_i - \sum_{\kappa=1}^{k} \omega_\kappa \widehat{m}_\kappa(\boldsymbol{x}_i) \right) \right\}$

Note that it might be interesting to regularize, by adding a Lasso-type penalty term, based on $\lambda \sum_{\kappa=1}^{k} |\omega_\kappa|$

Brief Summary

k-nearest neighbors

+ very intuitive
— sensitive to irrelevant features, does not work in high dimension

trees and forests

+ single tree easy to interpret, tolerent to irrelevant features, works with big data
— cannot handle linear combinations of features

support vector machine

+ good predictive power
— looks like a black box

boosting

+ intuitive and efficient
— looks like a black box

References (... to go further)

Hastie *et al.* (2009) The Elements of Statistical Learning. Springer.

Flach (2012) Machine Learning. Cambridge University Press.

Vapnik (1998) Statistical Learning Theory. Wiley.

Shapire & Freund (2012) Boosting. MIT Press.

Berk (2008) Statistical Learning from a Regression Perspective. Springer.

Kuhn & Johnson (2013) Applied Predictive Modeling. Springer.

Mohri, Rostamizadeh & Talwalker (2012) Foundations of Machine Learning. MIT Press.

Duda, Hart & Stork (2012) Pattern Classification. Springer.

Angrist & Pischke (2015) Mastering Metrics. Princeton University Press.

Mitchell (1997) Machine Learning. McGraw-Hill.

Murphy (2012) Machine Learning: a Probabilistic Perspective. MIT Press.

Charpentier. (2014) Computational Actuarial Pricing, with R. CRC.