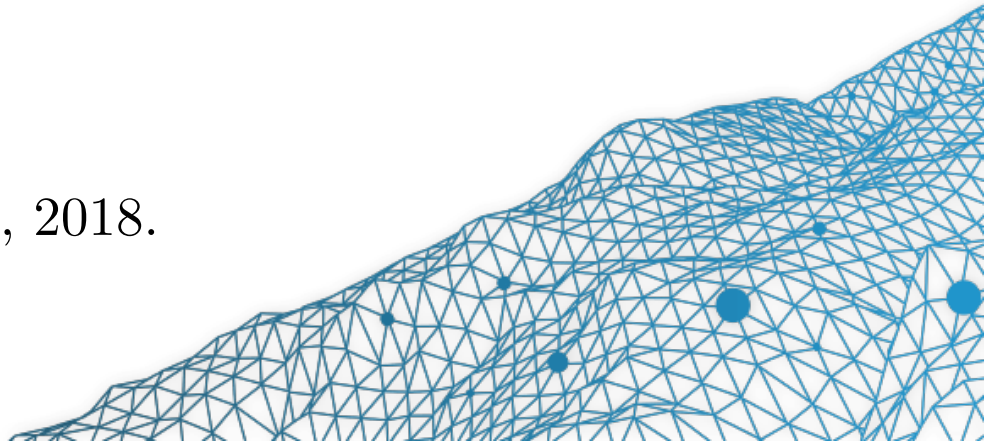
Big Data for Economics # 7

A. Charpentier (Université de Rennes 1)

https://github.com/freakonometrics/ub

UB School of Economics Summer School, 2018.



#7 New Tools for Classification Problems

Classification : Logistic Regression

The logistic regression is based on the assumption that

$$Y|\boldsymbol{X}=\boldsymbol{x}\sim\mathcal{B}(p_{\boldsymbol{x}}),\text{ where }p_{\boldsymbol{x}}=\frac{\exp[\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta}]}{1+\exp[\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta}]}=[1+\exp[-\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta}]]^{-1}$$

The goal is to estimate parameter $\boldsymbol{\beta}$.

Recall that the heuristics for the use of that function for the probability is that

$$\log[\text{odds}(Y=1)]=\log\frac{\mathbb{P}[Y=1]}{\mathbb{P}[Y=0]}=\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta}$$

- Maximum of the (log)-likelihood function

The log-likelihood is here

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

where $p_i=(1+\exp[-\boldsymbol{x}_i{}^{\mathsf{T}}\boldsymbol{\beta}])^{-1}$.

Classification : Logistic Regression

Using `optim()` might not be a great idea.

BFGS algorithm (Broyden, Fletcher, Goldfarb & Shanno) based on Newton's algorithm. To find a zero of function g, $g(x_0) = 0$, use Taylor's expansion,

$$0 = g(x_0) = g(x) + g'(x)(x_0 - x)$$

i.e.

$$x_{\text{new}} = x_{\text{old}} - \frac{g(x_{\text{old}})}{g'(x_{\text{old}})} \text{ from some starting point}$$

Here h is the gradient of the log-likelihood,

$$\boldsymbol{\beta}_{\text{new}} = \boldsymbol{\beta}_{\text{old}} - \left(\frac{\partial^2 \log \mathcal{L}(\boldsymbol{\beta}_{\text{old}})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} \right)^{-1} \cdot \frac{\partial \log \mathcal{L}(\boldsymbol{\beta}_{\text{old}})}{\partial \boldsymbol{\beta}}$$

Classification : Logistic Regression

Numerically, since only g is given, consider some small $h > 0$ and set

$$x_{\text{new}} = x_{\text{old}} - \frac{2h \cdot g(x_{\text{old}})}{g(x_{\text{old}} + h) + g(x_{\text{old}} - h)} \text{ from some starting point}$$

```
y = myocarde$PRONO
X = cbind(1,as.matrix(myocarde[,1:7]))
negLogLik = function(beta){
 -sum(-y*log(1 + exp(-(X%*%beta))) - (1-y)*log(1 + exp(X%*%beta)))
 }
beta_init = lm(PRONO~.,data=myocarde)$coefficients
logistic_opt = optim(par = beta_init, negLogLik, hessian=TRUE, method
     = "BFGS", control=list(abstol=1e-9))
```

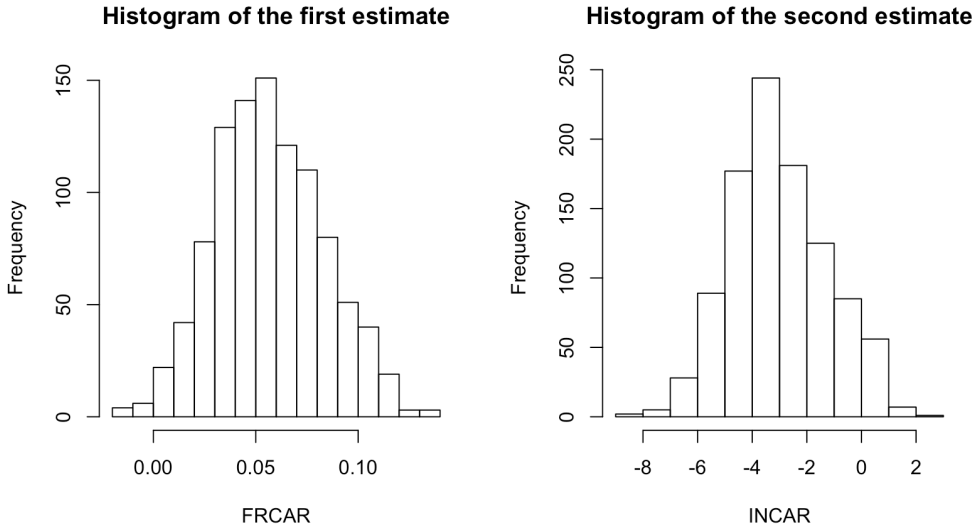
Classification : Logistic Regression

Here, we obtain

```
logistic_opt$par
 (Intercept)         FRCAR         INCAR         INSYS
 1.656926397  0.045234029 -2.119441743  0.204023835
       PRDIA         PAPUL         PVENT         REPUL
-0.102420095  0.165823647 -0.081047525 -0.005992238
simu = function(i){
logistic_opt_i = optim(par = rnorm(8,0,3)*beta_init,
negLogLik, hessian=TRUE, method = "BFGS",
control=list(abstol=1e-9))
logistic_opt_i$par[2:3]
}
v_beta = t(Vectorize(simu)(1:1000))
```

Classification : Logistic Regression

```
plot(v_beta)
par(mfrow=c(1,2))
hist(v_beta[,1],xlab=names(myocarde)[1])
hist(v_beta[,2],xlab=names(myocarde)[2])
```



- Fisher Algorithm

We want to use Newton's algorithm

$$\beta_{\text{new}} = \beta_{\text{old}} - \left(\frac{\partial^2 \log \mathcal{L}(\beta_{\text{old}})}{\partial \beta \partial \beta^T} \right)^{-1} \cdot \frac{\partial \log \mathcal{L}(\beta_{\text{old}})}{\partial \beta}$$

but here, the gradient and the hessian matrix have explicit expressions

$$\frac{\partial \log \mathcal{L}(\beta_{old})}{\partial \beta} = \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p}_{old})$$

$$\frac{\partial^2 \log \mathcal{L}(\beta_{old})}{\partial \beta \partial \beta^T} = -\boldsymbol{X}^{\mathsf{T}} \boldsymbol{\Delta}_{old} \boldsymbol{X}$$

Note that it is fast and robust (to the starting point)

```
Y=myocarde$PRONO
X=cbind(1,as.matrix(myocarde[,1:7]))
colnames(X)=c("Inter",names(myocarde[,1:7]))
 beta=as.matrix(lm(Y~0+X)$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[,8:10]
                  [,1]           [,2]           [,3]
XInter -10.187641685 -10.187641696 -10.187641696
XFRCAR   0.138178119   0.138178119   0.138178119
XINCAR  -5.862429035  -5.862429037  -5.862429037
XINSYS   0.717084018   0.717084018   0.717084018
XPRDIA  -0.073668171  -0.073668171  -0.073668171
XPAPUL   0.016756506   0.016756506   0.016756506
XPVENT  -0.106776012  -0.106776012  -0.106776012
XREPUL  -0.003154187  -0.003154187  -0.003154187
```

- iteratively reweighted least squares

We want to compute something like

$$\boldsymbol{\beta}_{new} = (\boldsymbol{X}^\top \boldsymbol{\Delta}_{old} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{\Delta}_{old} \boldsymbol{z}$$

(if we do substitute matrices in the analytical expressions) where

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{\beta}_{old} + \boldsymbol{\Delta}_{old}^{-1}[\boldsymbol{y} - \boldsymbol{p}_{old}]$$

But actually, that's simply a standard least-square problem

$$\boldsymbol{\beta}_{new} = \text{argmin}\left\{(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^\top \boldsymbol{\Delta}_{old}^{-1}(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})\right\}$$

The only problem here is that weights $\boldsymbol{\Delta}_{old}$ are functions of unknown $\boldsymbol{\beta}_{old}$

But actually, if we keep iterating, we should be able to solve it : given the $\boldsymbol{\beta}$ we got the weights, and with the weights, we can use weighted OLS to get an updated $\boldsymbol{\beta}$. That's the idea of iteratively reweighted least squares.

```
df = myocarde
beta_init = lm(PRONO~.,data=df)$coefficients
```

```
X = cbind(1,as.matrix(myocarde[,1:7]))
beta = beta_init
for(s in 1:1000){
p = exp(X %*% beta) / (1+exp(X %*% beta))
omega = diag(nrow(df))
diag(omega) = (p*(1-p))
df$Z = X %*% beta + solve(omega) %*% (df$PRONO - p)
beta = lm(Z~.,data=df[,-8], weights=diag(omega))$coefficients
}
 beta
  (Intercept)         FRCAR         INCAR         INSYS         PRDIA
-10.187641696  0.138178119  -5.862429037  0.717084018  -0.073668171
        PAPUL         PVENT         REPUL
  0.016756506  -0.106776012  -0.003154187
```

Classification : Logistic Regression

```
summary( lm(Z~.,data=df[,-8], weights=diag(omega)))

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -10.187642  10.668138  -0.955    0.343
FRCAR         0.138178   0.102340   1.350    0.182
INCAR        -5.862429   6.052560  -0.969    0.336
INSYS         0.717084   0.503527   1.424    0.159
PRDIA        -0.073668   0.261549  -0.282    0.779
PAPUL         0.016757   0.306666   0.055    0.957
PVENT        -0.106776   0.099145  -1.077    0.286
REPUL        -0.003154   0.004386  -0.719    0.475
```

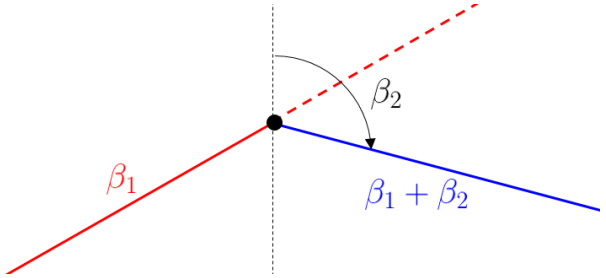
Classification : Logistic Regression

```
summary(glm(PRONO~.,data=myocarde,family=binomial(link = "logit")))

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
```

Classification : Logistic Regression with Splines

To get home-made linear splines, use

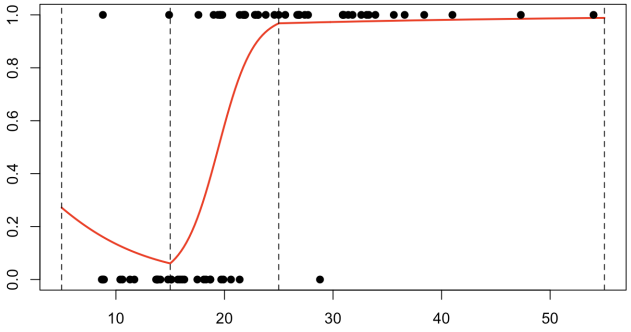
$$\beta_1 x + \beta_2 (x-s)_+$$




```
pos = function(x,s) (x-s)*(x>=s)
```

```
reg = glm(PRONO~INSYS+pos(INSYS,15)+pos(INSYS,25),data=myocarde,
    family=binomial)
```

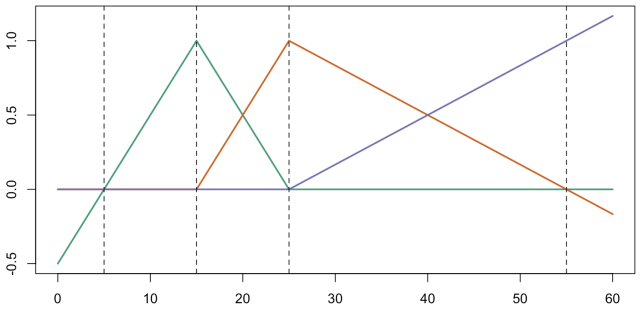
```
summary(reg)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.1109    3.278   -0.034    0.973
INSYS        -0.1751    0.252   -0.693    0.488
pos(INS,15)   0.7900    0.374    2.109    0.034 *
pos(INS,25)  -0.5797    0.290   -1.997    0.045 *
```



Classification : Logistic Regression with Splines

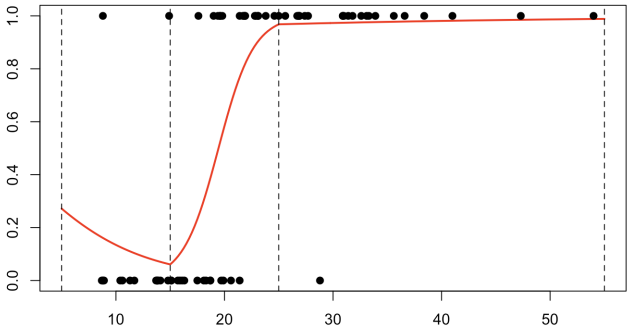
We can define spline functions with support $(5, 55)$ and with knots $\{15, 25\}$

```
library(splines)
clr6 = c("#1b9e77","#d95f02","#7570b3","#
    e7298a","#66a61e","#e6ab02")
x = seq(0,60,by=.25)
B = bs(x,knots=c(15,25),Boundary.knots=c(5,55)
    ,degre=1)
matplot(x,B,type="l",lty=1,lwd=2,col=clr6)
```



Classification : Logistic Regression with Splines

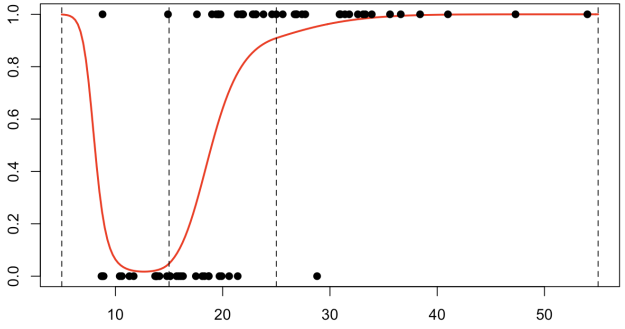
```
reg = glm(PRONO~bs(INSYS,knots=c(15,25),
Boundary.knots=c(5,55),degre=1),
data=myocarde,family=binomial)
summary(reg)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.9863     2.055  -0.480    0.631
bs(INSYS)1   -1.7507     2.526  -0.693    0.488
bs(INSYS)2    4.3989     2.061   2.133    0.032 *
bs(INSYS)3    5.4572     5.414   1.008    0.313
```



Classification : Logistic Regression with Splines

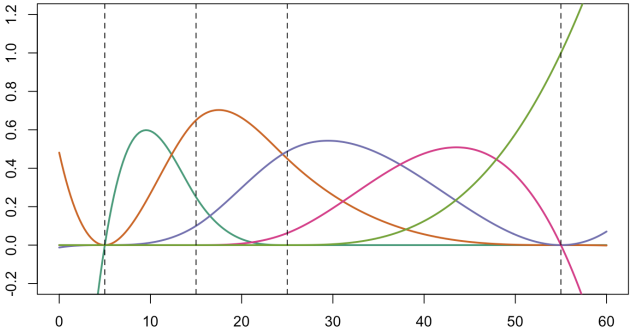
For quadratic splines, consider a decomposition on x,x^2,$(x-s_1)^2_+$

```
pos2 = function(x,s) (x-s)^2*(x>=s)
reg = glm(PRONO~poly(INSYS,2)+pos2(INSYS,15)+
    pos2(INSYS,25),
data=myocarde,family=binomial)
summary(reg)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 29.9842    15.236    1.968   0.049 *
poly(2)1   408.7851   202.419    2.019   0.043 *
poly(2)2   199.1628   101.589    1.960   0.049 *
pos2(15)     -0.2281     0.126   -1.805   0.071 .
pos2(25)      0.0439     0.080    0.545   0.585
```

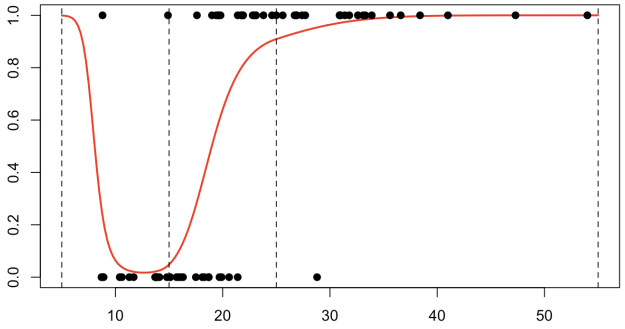


Classification : Logistic Regression with Splines

```
x = seq(0,60,by=.25)
B=bs(x,knots=c(15,25),Boundary.knots=c(5,55),
    degre=2)
matplot(x,B,type="l",xlab="INSYS",col=clr6)
```

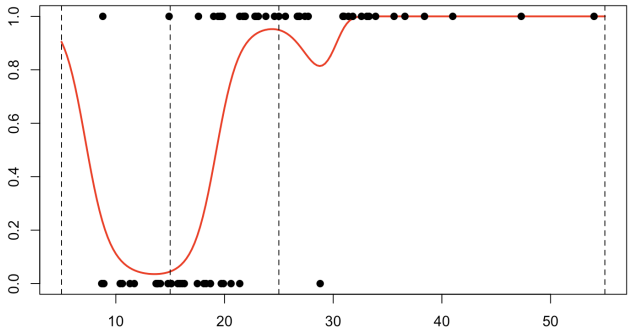


```
reg = glm(PRONO~bs(INSYS,knots=c(15,25),
Boundary.knots=c(5,55),degre=2),data=myocarde,
family=binomial)
summary(reg)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)     7.186      5.261   1.366    0.172
bs(INSYS)1    -14.656      7.923  -1.850    0.064 .
bs(INSYS)2     -5.692      4.638  -1.227    0.219
bs(INSYS)3     -2.454      8.780  -0.279    0.779
bs(INSYS)4      6.429     41.675   0.154    0.877
```

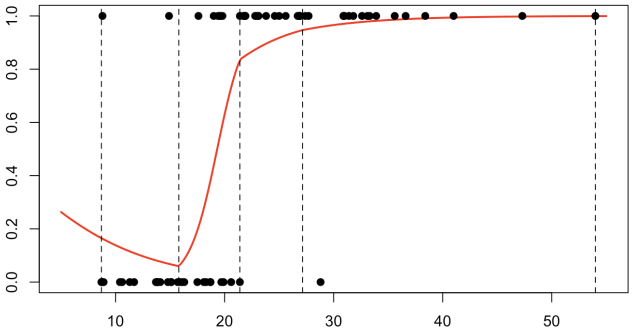


Finally, for cubic splines, use x, x^2, x^3, $(x-s_1)^3_+$, $(x-s_2)^3_+$, etc.

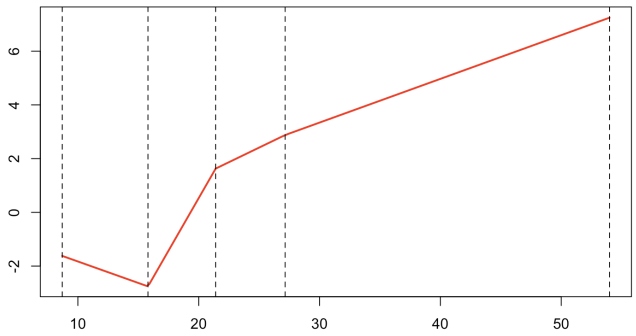
```
B=bs(x,knots=c(15,25),Boundary.knots=c(5,55),
    degre=3)
matplot(x,B,type="l",lwd=2,col=clr6,lty=1,ylim
    =c(-.2,1.2))
abline(v=c(5,15,25,55),lty=2)
```




```
reg = glm(PRONO~1+bs(INSYS,degree=1,df=4),data
    =myocarde,family=binomial)
attr(reg$terms, "predvars")[[3]]
bs(INSYS, degree = 1L, knots = c(15.8, 21.4,
    27.15),
Boundary.knots = c(8.7, 54), intercept = FALSE
    )
quantile(myocarde$INSYS,(0:4)/4)
   0%   25%   50%   75%  100%
 8.70 15.80 21.40 27.15 54.00
```

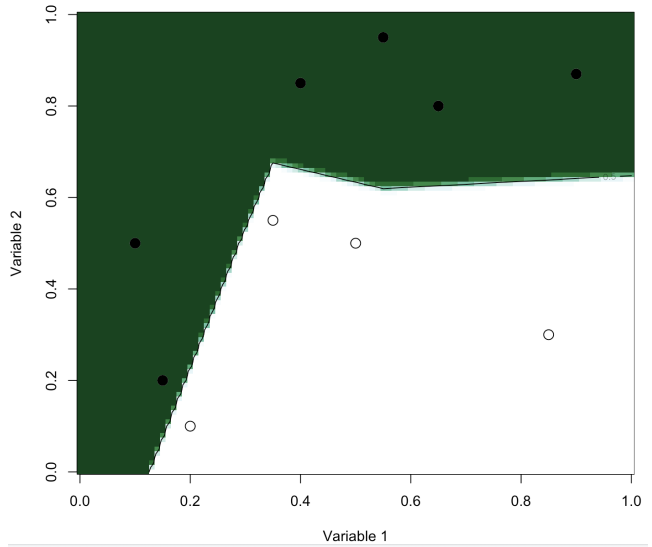


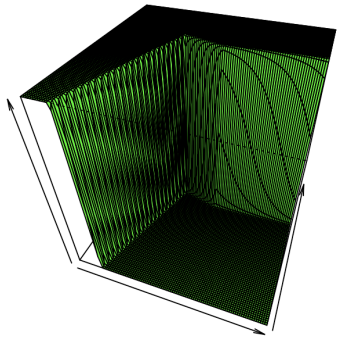
```r
B = bs(x,degree=1,df=4)
B = cbind(1,B)
y = B%*%coefficients(reg)
plot(x,y,type="l",col="red",lwd=2)
abline(v=quantile(myocarde$INSYS,(0:4)/4),lty
    =2)
```



Classification : Logistic Regression with Splines

```
reg = glm(y~bs(x1,degree=1,df=3)+bs(x2,degree
    =1,df=3),data=df,family=binomial(link = "
    logit"))
u = seq(0,1,length=101)
p = function(x,y) predict.glm(reg,newdata=data
    .frame(x1=x,x2=y),type="response")
v = outer(u,u,p)
image(u,u,v,xlab="Variable 1",ylab="Variable 2
    ",col=clr10,breaks=(0:10)/10)
points(df$x1,df$x2,pch=19,cex=1.5,col="white")
points(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],
    cex=1.5)
contour(u,u,v,levels = .5,add=TRUE)
```





Classification : Regression with Kernels and k-NN

We do not use the probit framework, it is purely a non-parametric one.

The moving regressogram is

$$\hat{m}(x) = \frac{\sum_{i=1}^{n} y_i \cdot \mathbf{1}(x_i \in [x \pm h])}{\sum_{i=1}^{n} \mathbf{1}(x_i \in [x \pm h])}$$

Consider

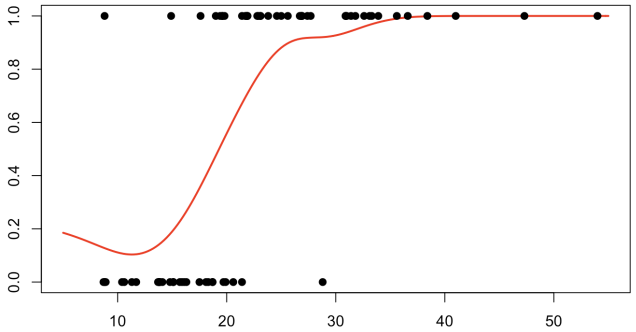
$$\tilde{m}(x) = \frac{\sum_{i=1}^{n} y_i \cdot k_h(x - x_i)}{\sum_{i=1}^{n} k_h(x - x_i)}$$

where (classically), for some kernel k,

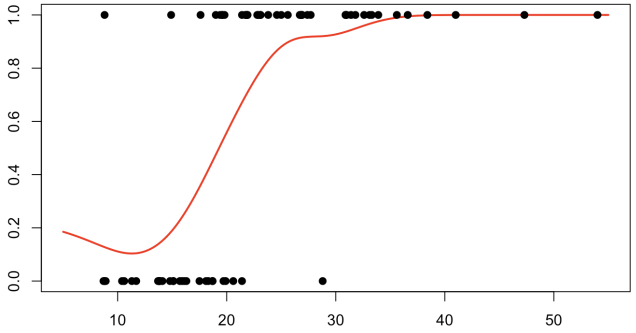
$$k_h(x) = k\left(\frac{x}{h}\right) k$$

Classification : Regression with Kernels and k-NN

```
mean_x = function(x,bw){
  w = dnorm((myocarde$INSYS-x)/bw, mean=0,sd
    =1)
  weighted.mean(myocarde$PRONO,w)}
u = seq(5,55,length=201)
v = Vectorize(function(x) mean_x(x,3))(u)
plot(u,v,ylim=0:1,type="l",col="red")
points(myocarde$INSYS,myocarde$PRONO,pch=19)
```

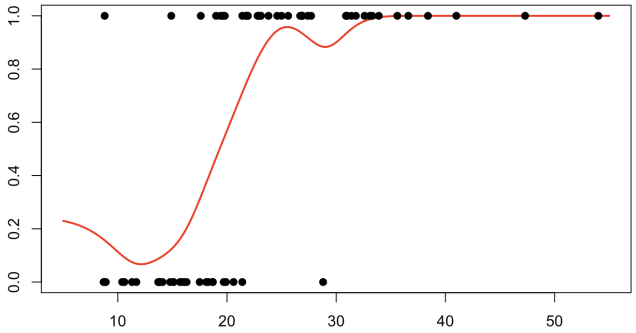



```
v = Vectorize(function(x) mean_x(x,2))(u)
plot(u,v,ylim=0:1,type="l",col="red")
points(myocarde$INSYS,myocarde$PRONO,pch=19)
```



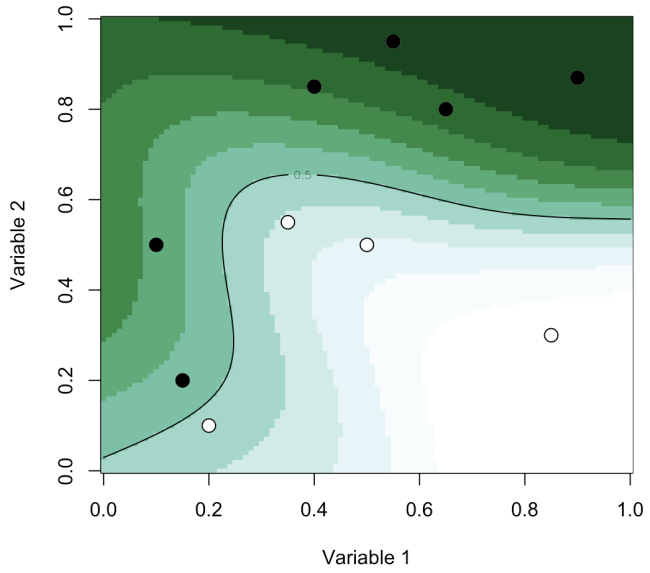
Classification : Regression with Kernels and k-NN

```
reg = ksmooth(myocarde$INSYS,myocarde$PRONO,"
    normal",bandwidth = 2*exp(1))
plot(reg$x,reg$y,ylim=0:1,type="l",col="red",
    lwd=2,xlab="INSYS",ylab="")
points(myocarde$INSYS,myocarde$PRONO,pch=19)
```



Classification : Regression with Kernels and k-NN

```
u = seq(0,1,length=101)
p = function(x,y){
  bw1 = .2; bw2 = .2
  w = dnorm((df$x1-x)/bw1, mean=0,sd=1)*
      dnorm((df$x2-y)/bw2, mean=0,sd=1)
  weighted.mean(df$y=="1",w)
}
v = outer(u,u,Vectorize(p))
image(u,u,v,col=clr10,breaks=(0:10)/10)
points(df$x1,df$x2,pch=19,cex=1.5,col="white")
points(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],
    cex=1.5)
contour(u,u,v,levels = .5,add=TRUE)
```



Classification : Regression with Kernels and k-NN

One can also use k-nearest neigthbors

$$\tilde{m}_k(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}\omega_{i,k}(\boldsymbol{x})y_i$$

where $\omega_{i,k}(\boldsymbol{x}) = n/k$ if $i \in \mathcal{I}_{\boldsymbol{x}}^k$ with

$$\mathcal{I}_{\boldsymbol{x}}^k = \{i : \boldsymbol{x}_i \text{ one of the } k \text{ nearest observations to } \boldsymbol{x}\}I$$

Classically, use Mahalanobis distance

```
Sigma = var(myocarde[,1:7])
Sigma_Inv = solve(Sigma)
d2_mahalanobis = function(x,y,Sinv){as.numeric(x-y)%*%Sinv%*%t(x-y)}
k_closest = function(i,k){
  vect_dist = function(j) d2_mahalanobis(myocarde[i,1:7],myocarde[j
    ,1:7],Sigma_Inv)
vect = Vectorize(vect_dist)((1:nrow(myocarde)))
which((rank(vect)))}
```

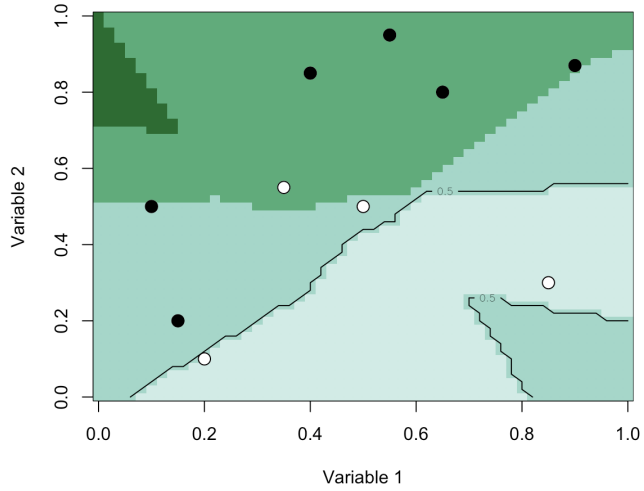
```r
k_majority = function(k){
  Y=rep(NA,nrow(myocarde))
  for(i in 1:length(Y)) Y[i] = sort(myocarde$PRONO[k_closest(i,k)])[(
    k+1)/2]
  return(Y)}
```

```r
k_mean = function(k){
  Y=rep(NA,nrow(myocarde))
  for(i in 1:length(Y)) Y[i] = mean(myocarde$PRONO[k_closest(i,k)])
  return(Y)}
```

```r
Sigma_Inv = solve(var(df[,c("x1","x2")]))
u = seq(0,1,length=51)
p = function(x,y){
  k = 6
  vect_dist = function(j)  d2_mahalanobis(c(x,y),df[j,c("x1","x2")],
    Sigma_Inv)
  vect = Vectorize(vect_dist)(1:nrow(df))
  idx  = which(rank(vect)<=k)
```

```
  return(mean((df$y==1)[idx]))}
```

```
v = outer(u,u,Vectorize(p))
image(u,u,v,col=clr10,breaks=(0:10)/10)
points(df$x1,df$x2,pch=19,cex=1.5,col="white")
points(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],
    cex=1.5)
contour(u,u,v,levels = .5,add=TRUE)
```



Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

We want to solve something like

$$\widehat{\boldsymbol{\beta}}_\lambda = \text{argmin}\{-\log \mathcal{L}(\boldsymbol{\beta}|\boldsymbol{x}, \boldsymbol{y}) + \lambda \|\boldsymbol{\beta}\|_{\ell_2}^2\}$$

In the case where we consider the log-likelihood of some Gaussian variable, we get the sum of the square of the residuals, and we can obtain an explicit solution. But not in the context of a logistic regression.

The heuristics about Ridge regression is the following graph. In the background, we can visualize the (two-dimensional) log-likelihood of the logistic regression, and the blue circle is the constraint we have, if we rewrite the optimization problem as a constrained optimization problem :

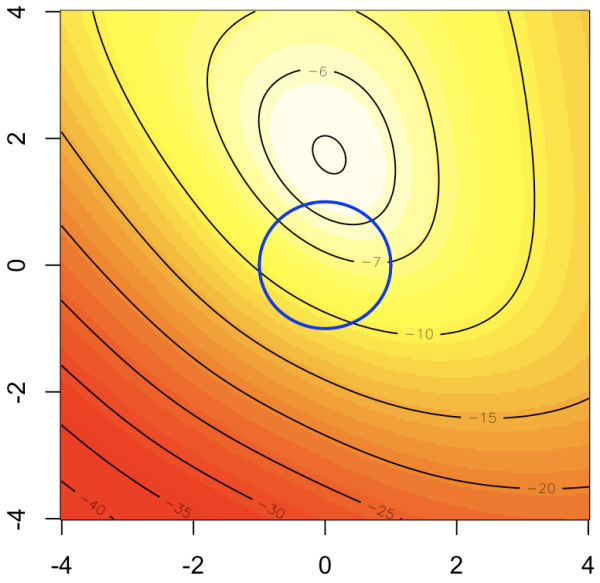
$$\min_{\boldsymbol{\beta}:\|\boldsymbol{\beta}\|_{\ell_2}^2 \leq s} \left\{ \sum_{i=1}^{n} -\log \mathcal{L}(y_i, \beta_0 + \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}) \right\}$$

Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

It can be written equivalently (it is a strictly convex problem)

$$\min_{\boldsymbol{\beta},\lambda}\left\{-\sum_{i=1}^{n}\log\mathcal{L}(y_i,\beta_0+\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta})+\lambda\|\boldsymbol{\beta}\|_{\ell_2}^2\right\}$$

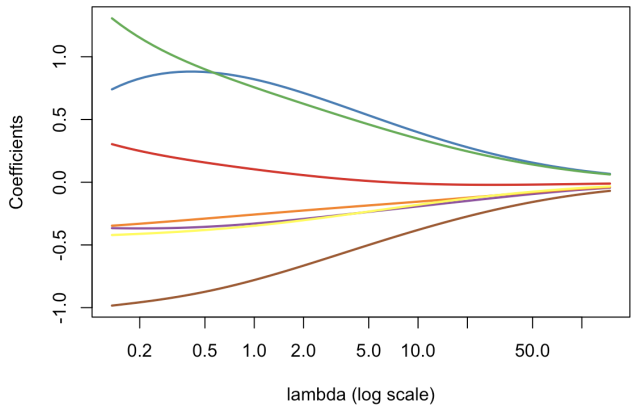
Thus, the constrained maximum should lie in the blue disk.




```
> PennegLogLik = function(bbeta,lambda=0){
  b0   = bbeta[1]
  beta = bbeta[-1]
 -sum(-y*log(1 + exp(-(b0+X%*%beta))) - (1-y)*
  log(1 + exp(b0+X%*%beta)))+lambda*sum(beta^2)}
```

Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

```
> opt_ridge = function(lambda){
beta_init = lm(PRONO~.,data=myocarde)$
    coefficients
logistic_opt = optim(par = beta_init*0,
    function(x) PennegLogLik(x,lambda), method
     = "BFGS", control=list(abstol=1e-9))
logistic_opt$par[-1]}
> v_lambda = c(exp(seq(-2,5,length=61)))
> est_ridge = Vectorize(opt_ridge)(v_lambda)
> plot(v_lambda,est_ridge[1,])
```



Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

- Using Fisher's (penalized) score

On the penalized problem, we can easily prove that

$$\frac{\partial \log \mathcal{L}_p(\boldsymbol{\beta}_{\lambda,old})}{\partial \boldsymbol{\beta}} = \frac{\partial \log \mathcal{L}(\boldsymbol{\beta}_{\lambda,old})}{\partial \boldsymbol{\beta}} - 2\lambda \boldsymbol{\beta}_{old}$$

while

$$\frac{\partial^2 \log \mathcal{L}_p(\boldsymbol{\beta}_{\lambda,old})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = \frac{\partial^2 \log \mathcal{L}(\boldsymbol{\beta}_{\lambda,old})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} - 2\lambda \mathbb{I}$$

Hence

$$\boldsymbol{\beta}_{\lambda,new} = (\boldsymbol{X}^\mathsf{T} \boldsymbol{\Delta}_{old} \boldsymbol{X} + 2\lambda \mathbb{I})^{-1} \boldsymbol{X}^\mathsf{T} \boldsymbol{\Delta}_{old} \boldsymbol{z}$$

And interestingly, with that algorithm, we can also derive the variance of the estimator

$$\text{Var}[\widehat{\boldsymbol{\beta}}_\lambda] = [\boldsymbol{X}^\mathsf{T} \boldsymbol{\Delta} \boldsymbol{X} + 2\lambda \mathbb{I}]^{-1} \boldsymbol{X}^\mathsf{T} \boldsymbol{\Delta} \text{Var}[\boldsymbol{z}] \boldsymbol{\Delta} \boldsymbol{X} [\boldsymbol{X}^\mathsf{T} \boldsymbol{\Delta} \boldsymbol{X} + 2\lambda \mathbb{I}]^{-1}$$

where $\text{Var}[\boldsymbol{z}] = \boldsymbol{\Delta}^{-1}$

Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

The code to compute $\widehat{\beta}_\lambda$ is then

```
Y = myocarde$PRONO
X = myocarde[,1:7]
for(j in 1:7) X[,j] = (X[,j]-mean(X[,j]))/sd(X[,j])
X = as.matrix(X)
X = cbind(1,X)
colnames(X) = c("Inter",names(myocarde[,1:7]))
 beta = as.matrix(lm(Y~0+X)$coefficients,ncol=1)
 for(s in 1:9){
   pi = exp(X%*%beta[,s])/(1+exp(X%*%beta[,s]))
   Delta = matrix(0,nrow(X),nrow(X));diag(Delta)=(pi*(1-pi))
   z = X%*%beta[,s] + solve(Delta)%*%(Y-pi)
   B = solve(t(X)%*%Delta%*%X+2*lambda*diag(ncol(X))) %*% (t(X)%*%
    Delta%*%z)
   beta = cbind(beta,B)}
beta[,8:10]
                [,1]         [,2]         [,3]
```

```
XInter   0.59619654   0.59619654   0.59619654
XFRCAR   0.09217848   0.09217848   0.09217848
XINCAR   0.77165707   0.77165707   0.77165707
XINSYS   0.69678521   0.69678521   0.69678521
XPRDIA  -0.29575642  -0.29575642  -0.29575642
XPAPUL  -0.23921101  -0.23921101  -0.23921101
XPVENT  -0.33120792  -0.33120792  -0.33120792
XREPUL  -0.84308972  -0.84308972  -0.84308972
```

Use

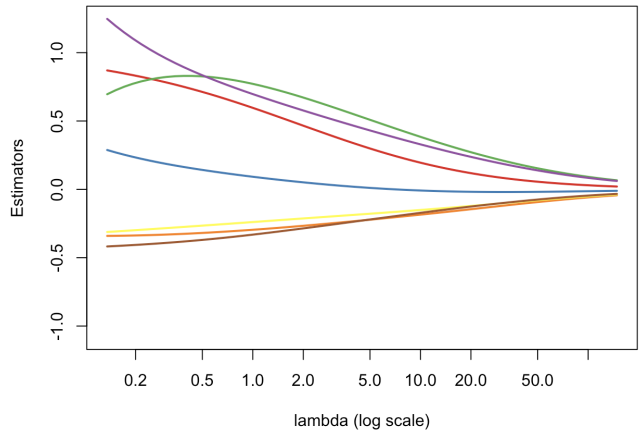
$$\text{Var}[\widehat{\beta}_\lambda] = [\boldsymbol{X}^\mathsf{T}\boldsymbol{\Delta}\boldsymbol{X} + 2\lambda\mathbb{I}]^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{\Delta}\text{Var}[\boldsymbol{z}]\boldsymbol{\Delta}\boldsymbol{X}[\boldsymbol{X}^\mathsf{T}\boldsymbol{\Delta}\boldsymbol{X} + 2\lambda\mathbb{I}]^{-1}$$

where $\text{Var}[\boldsymbol{z}] = \boldsymbol{\Delta}^{-1}$.

```
newton_ridge = function(lambda=1){
 beta = as.matrix(lm(Y~0+X)$coefficients,ncol=1)*runif(8)
 for(s in 1:20){
   pi = exp(X%*%beta[,s])/(1+exp(X%*%beta[,s]))
   Delta = matrix(0,nrow(X),nrow(X));diag(Delta)=(pi*(1-pi))
   z = X%*%beta[,s] + solve(Delta)%*%(Y-pi)
```

```
    B = solve(t(X)%*%Delta%*%X+2*lambda*diag(ncol(X))) %*% (t(X)%*%
     Delta%*%z)
    beta = cbind(beta,B)}
Varz = solve(Delta)
Varb = solve(t(X)%*%Delta%*%X+2*lambda*diag(ncol(X))) %*% t(X)%*%
     Delta %*% Varz %*%
   Delta %*% X %*% solve(t(X)%*%Delta%*%X+2*lambda*diag(ncol(X)))
return(list(beta=beta[,ncol(beta)],sd=sqrt(diag(Varb))))}
```

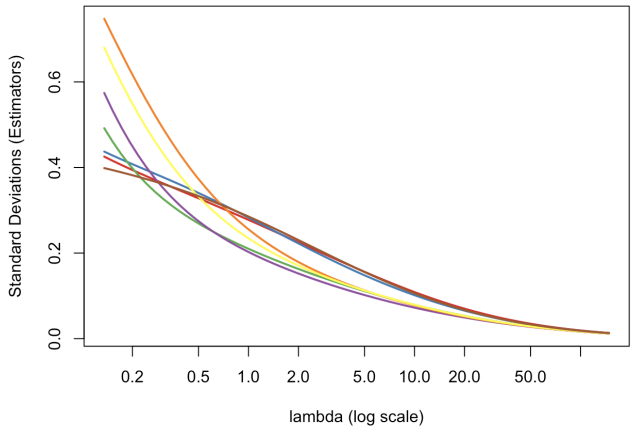
```
v_lambda=c(exp(seq(-2,5,length=61)))
est_ridge=Vectorize(function(x) newton_ridge(x
    )$beta)(v_lambda)
library("RColorBrewer")
colrs=brewer.pal(7,"Set1")
plot(v_lambda,est_ridge[1,],col=colrs[1],type=
    "l")
for(i in 2:7) lines(v_lambda,est_ridge[i,],col
    =colrs[i])
```



Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

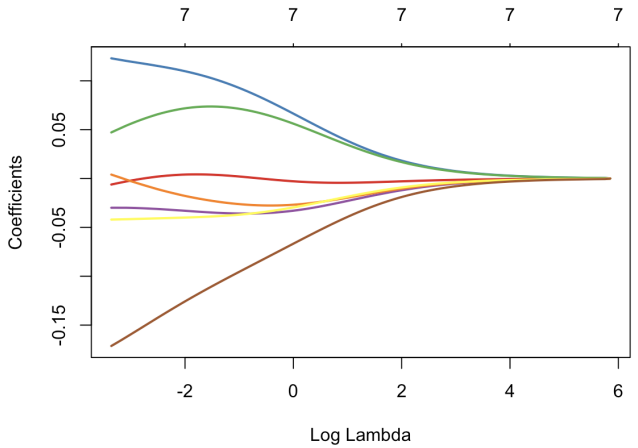
And for the variance

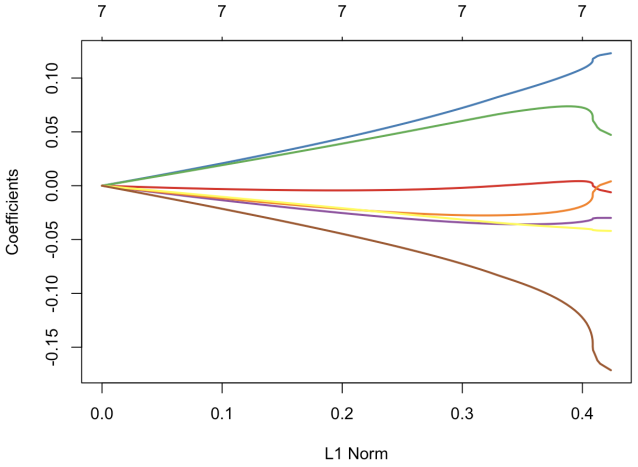
```
v_lambda=c(exp(seq(-2,5,length=61)))
est_ridge=Vectorize(function(x) newton_ridge(x
    )$sd)(v_lambda)
library("RColorBrewer")
colrs=brewer.pal(7,"Set1")
plot(v_lambda,est_ridge[1,],col=colrs[1],type=
    "l")
for(i in 2:7) lines(v_lambda,est_ridge[i,],col
    =colrs[i],lwd=2)
```



Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

```
y = myocarde$PRONO
X = myocarde[,1:7]
for(j in 1:7) X[,j] = (X[,j]-mean(X[,j]))/sd(X
    [,j])
X = as.matrix(X)
library(glmnet)
glm_ridge = glmnet(X, y, alpha=0)
plot(glm_ridge,xvar="lambda",col=colrs,lwd=2)
```

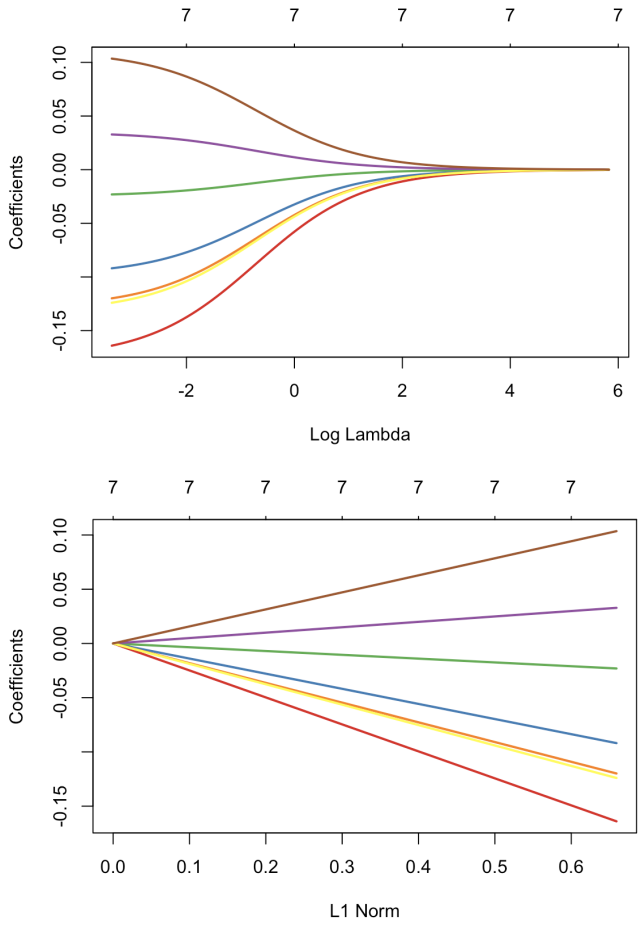




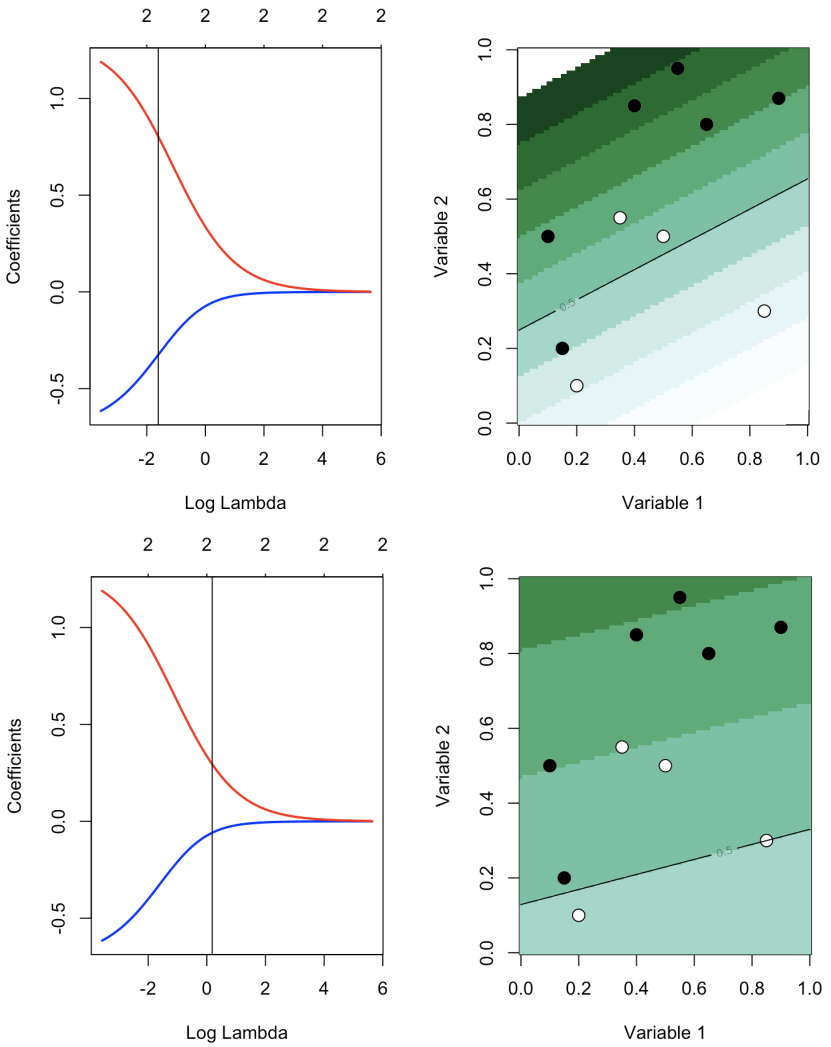
Classification : Penalized Logistic Regression, ℓ_1 norm (Ridge)

- Ridge Regression with Orthogonal Covariates

```
1 library(factoextra)
2 pca = princomp(X)
3 pca_X = get_pca_ind(pca)$coord
4
5 library(glmnet)
6 glm_ridge = glmnet(pca_X, y, alpha=0)
7 plot(glm_ridge,xvar="lambda",col=colrs,lwd=2)
8
9 plot(glm_ridge,col=colrs,lwd=2)
```



```r
df0 = df
df0$y=as.numeric(df$y)-1
plot_lambda = function(lambda){
m = apply(df0,2,mean)
s = apply(df0,2,sd)
for(j in 1:2) df0[,j] = (df0[,j]-m[
    j])/s[j]
reg = glmnet(cbind(df0$x1,df0$x2),
    df0$y==1, alpha=0)
par(mfrow=c(1,2))
plot(reg,xvar="lambda",col=c("blue"
    ,"red"),lwd=2)
abline(v=log(.2))
plot_lambda(.2)
plot(reg,xvar="lambda",col=c("blue"
    ,"red"),lwd=2)
abline(v=log(1.2))
plot_lambda(1.2)
```

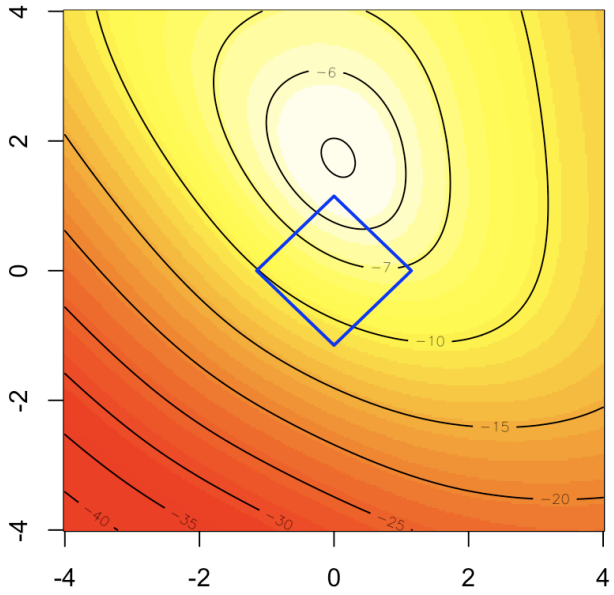


Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

It can be written equivalently (it is a strictly convex problem)

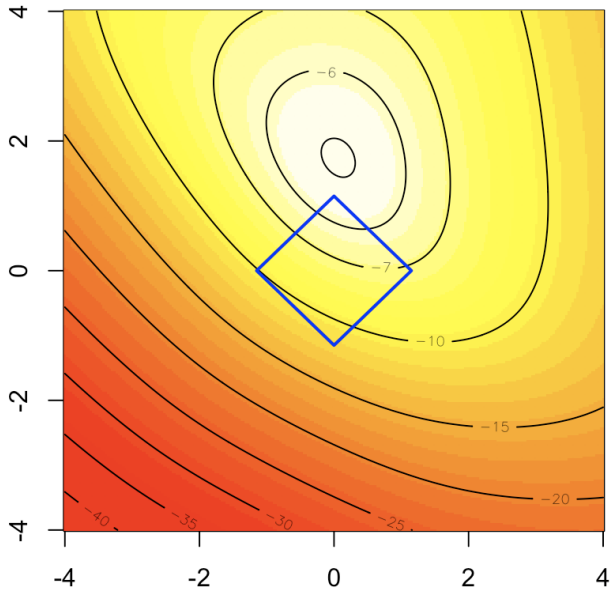
$$\min_{\boldsymbol{\beta},\lambda}\left\{-\sum_{i=1}^{n}\log\mathcal{L}(y_i,\beta_0+\boldsymbol{x}^\mathsf{T}\boldsymbol{\beta})+\lambda\|\boldsymbol{\beta}\|_{\ell_1}\right\}$$

Thus, the constrained maximum should lie in the blue disk.

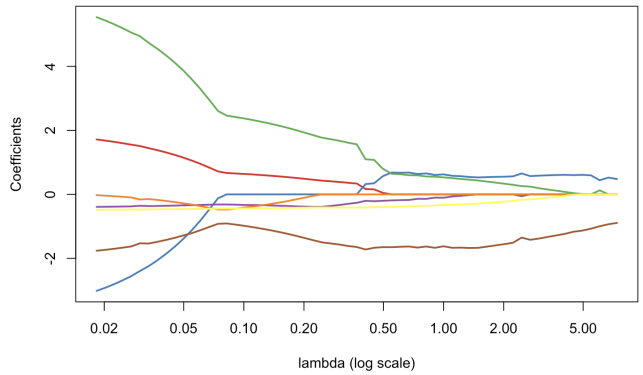


Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

```
y = myocarde$PRONO
X = myocarde[,1:7]
for(j in 1:7) X[,j] = (X[,j]-mean(X[,j]))/sd(X
    [,j])
X = as.matrix(X)
LogLik = function(bbeta){
  b0=bbeta[1]
  beta=bbeta[-1]
  sum(-y*log(1 + exp(-(b0+X%*%beta))) -
  (1-y)*log(1 + exp(b0+X%*%beta)))}
u = seq(-4,4,length=251)
v = outer(u,u,function(x,y) LogLik(c(1,x,y)))
image(u,u,v,col=rev(heat.colors(25)))
contour(u,u,v,add=TRUE)
polygon(c(-1,0,1,0),c(0,1,0,-1),border="blue")
```

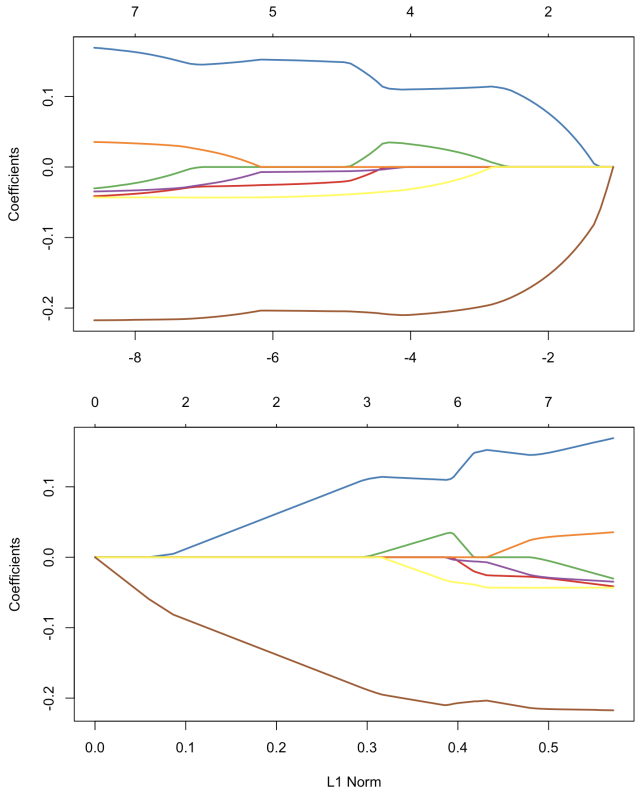


```r
PennegLogLik = function(bbeta,lambda=0){
  b0=bbeta[1]
  beta=bbeta[-1]
 -sum(-y*log(1 + exp(-(b0+X%*%beta))) -
(1-y)*log(1 + exp(b0+X%*%beta)))+lambda*sum(
    abs(beta))
}
opt_lasso = function(lambda){
beta_init = lm(PRONO~.,data=myocarde)$
    coefficients
logistic_opt = optim(par = beta_init*0,
    function(x) PennegLogLik(x,lambda),
hessian=TRUE, method = "BFGS", control=list(
    abstol=1e-9))
logistic_opt$par[-1]
}
v_lambda=c(exp(seq(-4,2,length=61)))
est_lasso=Vectorize(opt_lasso)(v_lambda)
library("RColorBrewer")
colrs=brewer.pal(7,"Set1")
```



Observe that coefficients are not (strictly) null

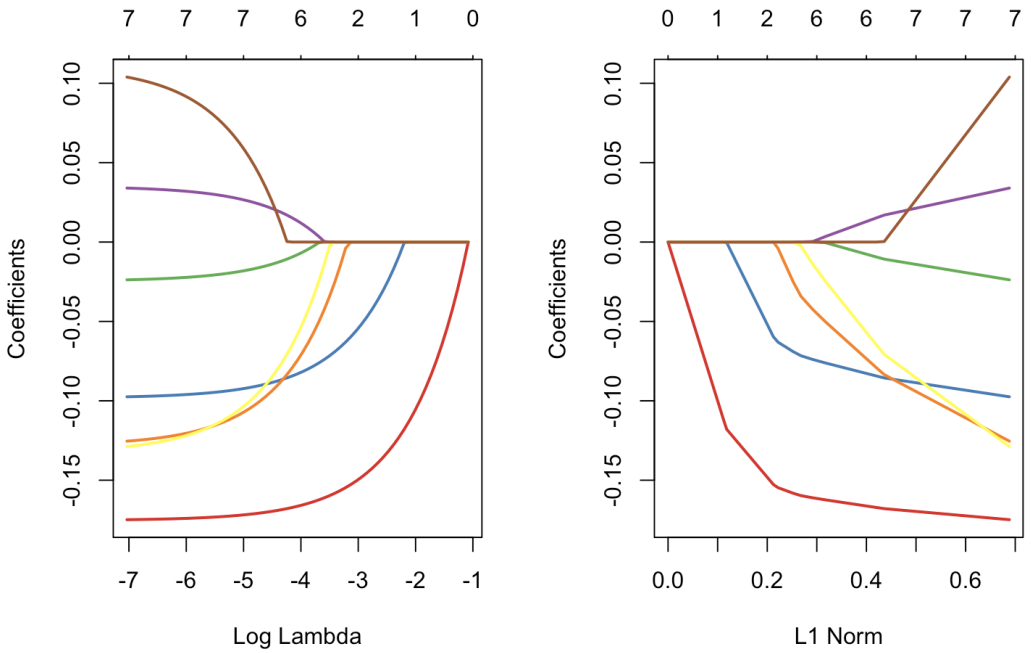
```
library(glmnet)
glm_lasso = glmnet(X, y, alpha=1)
plot(glm_lasso,xvar="lambda",col=colrs,lwd=2)

plot(glm_lasso,col=colrs,lwd=2)
glmnet(X, y, alpha=1,lambda=exp(-4))$beta
7x1 sparse Matrix of class "dgCMatrix"
                    s0
FRCAR   .
INCAR   0.11005070
INSYS   0.03231929
PRDIA   .
PAPUL   .
PVENT  -0.03138089
REPUL  -0.20962611
```



Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

With orthogonal covariates

```
4 library(factoextra)
5 pca = princomp(X)
6 pca_X = get_pca_ind(pca)$coord
7 glm_lasso = glmnet(pca_X, y,
      alpha=1)
8 plot(glm_lasso,xvar="lambda",
      col=colrs)
9 plot(glm_lasso,col=colrs)
```



Before focusing on the LASSO for binary variables, consider the standard OLD-LASSO

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

If we get back to the original Lasso approach, the goal was to solve

$$\min\left\{\frac{1}{2n}\sum_{i=1}^{n}[y_i-(\beta_0+\boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta})]^2+\lambda\sum_j|\beta_j|\right\}$$

Since the intercept is not subject to the penalty. The first order condition is then

$$\frac{\partial}{\partial\beta_0}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\beta_0\mathbf{1}\|^2=(\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{y})^{\mathsf{T}}\mathbf{1}+\beta_0\|\mathbf{1}\|^2=0$$

i.e.

$$\beta_0=\frac{1}{n^2}(\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{y})^{\mathsf{T}}\mathbf{1}$$

For other parameters, assuming tha KKT conditions are satisfied, we can check if $\mathbf{0}$ contains the subdifferential at the minimum,

$$\mathbf{0}\in\partial\left(\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2+\lambda\|\boldsymbol{\beta}\|_{\ell_1}\right)=\frac{1}{2}\nabla\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2+\partial(\lambda\|\boldsymbol{\beta}\|_{\ell_1})$$

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

For the term on the left, we recognize

$$\frac{1}{2}\nabla\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2=-\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})=-\boldsymbol{g}$$

so that the previous equation can be writen

$$g_k\in\partial(\lambda|\beta_k|)=\begin{cases}\{+\lambda\} \text{ if } \beta_k>0\\ \{-\lambda\} \text{ if } \beta_k<0\\ (-\lambda,+\lambda) \text{ if } \beta_k=0\end{cases}$$

i.e. if $\beta_k\neq 0$, then $g_k=\text{sign}(\beta_k)\cdot\lambda$.

We can split β_j into a sum of its positive and negative parts by replacing β_j with $\beta_j^+-\beta_j^-$ where $\beta_j^+,\beta_j^-\geq 0$.

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

Then the Lasso problem becomes

$$-\log \mathcal{L}(\boldsymbol{\beta}) + \lambda \sum_j (\beta_j^+ - \beta_j^-)$$

with constraints $\beta_j^+ - \beta_j^-$. Let α_j^+, α_j^- denote the Lagrange multipliers for β_j^+, β_j^-, respectively.

$$L(\boldsymbol{\beta}) + \lambda \sum_j (\beta_j^+ - \beta_j^-) - \sum_j \alpha_j^+ \beta_j^+ - \sum_j \alpha_j^- \beta_j^-.$$

To satisfy the stationarity condition, we take the gradient of the Lagrangian with respect to β_j^+ and set it to zero to obtain

$$\nabla L(\boldsymbol{\beta})_j + \lambda - \alpha_j^+ = 0$$

We do the same with respect to β_j^- to obtain

$$-\nabla L(\boldsymbol{\beta})_j + \lambda - \alpha_j^- = 0.$$

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

For conveniency, introduce the soft-thresholding function

$$S(z,\gamma)=\text{sign}(z)\cdot(|z|-\gamma)_+=\begin{cases}z-\gamma & \text{if } \gamma<|z| \text{ and } z>0\\ z+\gamma & \text{if } \gamma<|z| \text{ and } z<0\\ 0 & \text{if } \gamma\geq|z|\end{cases}$$

Noticing that the optimization problem

$$\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|_{\ell_2}^2+\lambda\|\boldsymbol{\beta}\|_{\ell_1}$$

can also be written

$$\min\left\{\sum_{j=1}^{p}-\widehat{\beta}_j^{\text{ols}}\cdot\beta_j+\frac{1}{2}\beta_j^2+\lambda|\beta_j|\right\}$$

observe that $\widehat{\beta}_{j,\lambda}=S(\widehat{\beta}_j^{\text{ols}},\lambda)$ which is a coordinate-wise update.

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

Now, if we consider a (slightly) more general problem, with weights in the first part

$$\min\left\{\frac{1}{2n}\sum_{i=1}^{n}\omega_i[y_i-(\beta_0+\boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta})]^2+\lambda\sum_j|\beta_j|\right\}$$

the coordinate-wise update becomes

$$\widehat{\beta}_{j,\lambda,\omega}=S(\widehat{\beta}_j^{\omega-\text{ols}},\lambda)$$

An alternative is to set

$$\boldsymbol{r}_j=\boldsymbol{y}-\left(\beta_0\mathbf{1}+\sum_{k\neq j}\beta_k\boldsymbol{x}_k\right)=\boldsymbol{y}-\widehat{\boldsymbol{y}}^{(j)}$$

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

so that the optimization problem can be written, equivalently

$$\min\left\{\frac{1}{2n}\sum_{j=1}^{p}[\boldsymbol{r}_j-\beta_j\boldsymbol{x}_j]^2+\lambda|\beta_j|\right\}$$

hence

$$\min\left\{\frac{1}{2n}\sum_{j=1}^{p}\beta_j^2\|\boldsymbol{x}_j\|-2\beta_j\boldsymbol{r}_j{}^{\mathsf{T}}\boldsymbol{x}_j+\lambda|\beta_j|\right\}$$

and one gets

$$\beta_{j,\lambda}=\frac{1}{\|\boldsymbol{x}_j\|^2}S(\boldsymbol{r}_j{}^{\mathsf{T}}\boldsymbol{x}_j,n\lambda)$$

or, if we develop

$$\beta_{j,\lambda}=\frac{1}{\sum_i x_{ij}^2}S\left(\sum_i x_{i,j}[y_i-\widehat{y}_i^{(j)}],n\lambda\right)$$

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

Again, if there are weights $\boldsymbol{\omega} = (\omega_i)$, the coordinate-wise update becomes

$$\beta_{j,\lambda,\omega} = \frac{1}{\sum_i \omega_i x_{ij}^2} S\left(\sum_i \omega_i x_{i,j} [y_i - \widehat{y}_i^{(j)}], n\lambda \right)$$

The code to compute this componentwise descent is

```
soft_thresholding = function(x,a){
  result  a)]  a)] - a
  result[which(x < -a)] = x[which(x < (-a))] + a
  return(result)}
```

and the code

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

```
lasso_coord_desc = function(X,y,beta,lambda,tol=1e-6,maxiter=1000){
  beta = as.matrix(beta)
  X = as.matrix(X)
  omega = rep(1/length(y),length(y))
  obj = numeric(length=(maxiter+1))
  betalist = list(length(maxiter+1))
  betalist[[1]] = beta
    beta0list = numeric(length(maxiter+1))
    beta0 = sum(y-X%*%beta)/(length(y))
    beta0list[1] = beta0
    for (j in 1:maxiter){
      for (k in 1:length(beta)){
        r = y - X[,-k]%*%beta[-k] - beta0*rep(1,length(y))
        beta[k] = (1/sum(omega*X[,k]^2))*soft_thresholding(t(omega*r)
    %*%X[,k],length(y)*lambda)
      }
      beta0 = sum(y-X%*%beta)/(length(y))
```

```
17        beta0list[j+1] = beta0
18        betalist[[j+1]] = beta
19        obj[j] = (1/2)*(1/length(y))*norm(omega*(y - X%*%beta -
20 beta0*rep(1,length(y))),'F')^2 + lambda*sum(abs(beta))
21        if (norm(rbind(beta0list[j],betalist[[j]]) - rbind(beta0,beta),
      'F') < tol) { break }
22      }
23 return(list(obj=obj[1:j],beta=beta,intercept=beta0)) }
```

If we get back to our logistic problem, the log-likelihood is here

$$\log \mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} y_i \cdot (\beta_0 + \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta}) - \log[1+\exp(\beta_0 + \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta})]$$

which is a concave function of the parameters. Hence, one can use a quadratic approximation of the log-likelihood - using Taylor expansion,

$$\log \mathcal{L} \approx \log \mathcal{L}' = \frac{1}{n}\sum_{i=1}^{n} \omega_i \cdot [z_i - (\beta_0 + \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta})]^2$$

where z_i is the working response

$$z_i = (\beta_0 + \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}) + \frac{y_i - p_i}{p_i[1-p_i]}$$

p_i is the prediction

$$p_i = \frac{\exp[\beta_0 + \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}]}{1+\exp[\beta_0 + \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}]}$$

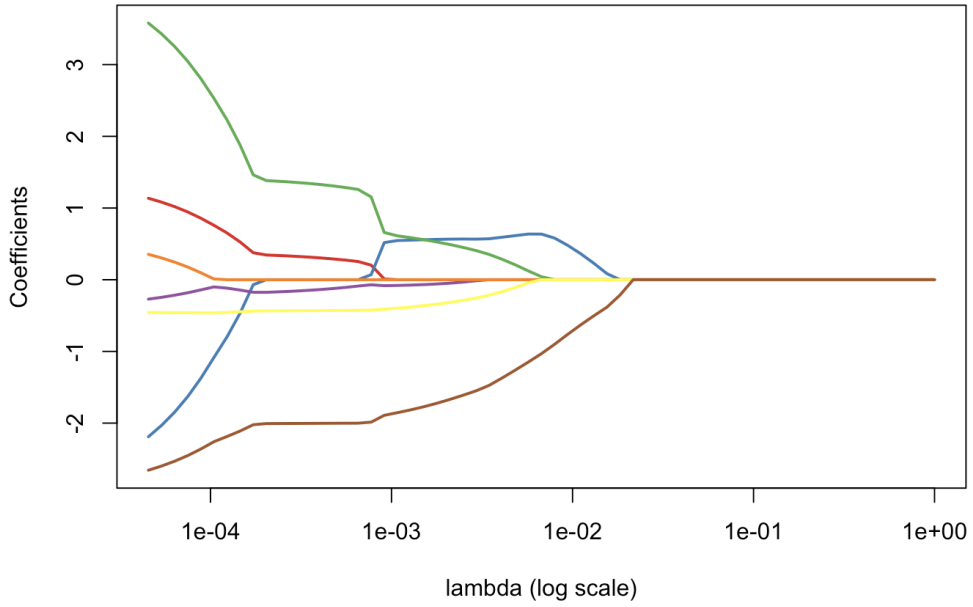
and ω_i are weights $\omega_i = p_i[1-p_i]$.

Thus, we obtain a penalized least-square problem.

```
lasso_coord_desc = function(X,y,beta,lambda,tol=1e-6,maxiter=1000){
  beta = as.matrix(beta)
  X = as.matrix(X)
  obj = numeric(length=(maxiter+1))
  betalist = list(length(maxiter+1))
  betalist[[1]] = beta
  beta0 = sum(y-X%*%beta)/(length(y))
  p = exp(beta0*rep(1,length(y)) + X%*%beta)/(1+exp(beta0*rep(1,
    length(y)) + X%*%beta))
```

```
  z = beta0*rep(1,length(y)) + X%*%beta + (y-p)/(p*(1-p))
  omega = p*(1-p)/(sum((p*(1-p))))
    beta0list = numeric(length(maxiter+1))
    beta0 = sum(y-X%*%beta)/(length(y))
    beta0list[1] = beta0
    for (j in 1:maxiter){
      for (k in 1:length(beta)){
        r = z - X[,-k]%*%beta[-k] - beta0*rep(1,length(y))
       beta[k] = (1/sum(omega*X[,k]^2))*soft_thresholding(t(omega*r)%
    *%X[,k],length(y)*lambda)
      }
      beta0 = sum(y-X%*%beta)/(length(y))
      beta0list[j+1] = beta0
      betalist[[j+1]] = beta
      obj[j] = (1/2)*(1/length(y))*norm(omega*(z - X%*%beta -
beta0*rep(1,length(y))),'F')^2 + lambda*sum(abs(beta))
  p = exp(beta0*rep(1,length(y)) + X%*%beta)/(1+exp(beta0*rep(1,
    length(y)) + X%*%beta))
```

```
  z = beta0*rep(1,length(y)) + X%*%beta + (y-p)/(p*(1-p))
  omega = p*(1-p)/(sum((p*(1-p))))
      if (norm(rbind(beta0list[j],betalist[[j]]) -
rbind(beta0,beta),'F') < tol) { break }
        }
return(list(obj=obj[1:j],beta=beta,intercept=beta0)) }
```

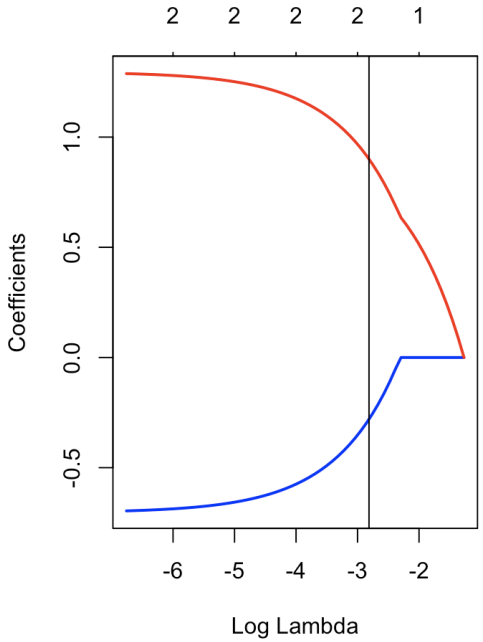


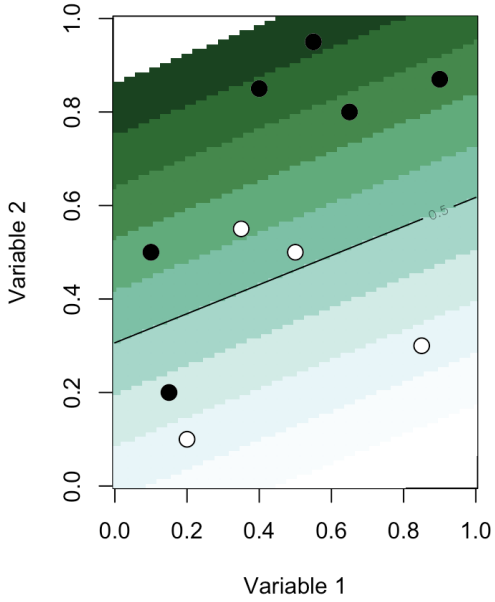
Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

```
df0 = df
df0$y = as.numeric(df$y)-1
plot_lambda = function(lambda){
m = apply(df0,2,mean)
s = apply(df0,2,sd)
for(j in 1:2) df0[,j] <- (df0[,j]-m[j])/s[j]
reg = glmnet(cbind(df0$x1,df0$x2), df0$y==1, alpha=1,lambda=lambda)
u = seq(0,1,length=101)
p = function(x,y){
  xt = (x-m[1])/s[1]
  yt = (y-m[2])/s[2]
  predict(reg,newx=cbind(x1=xt,x2=yt),type="response")}
```

Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

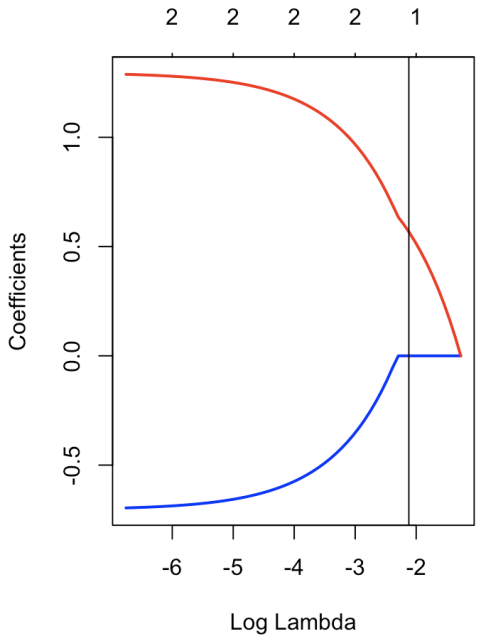
```
reg = glmnet(cbind(df0$x1,df0$x2), df0$y==1, alpha=1)
par(mfrow=c(1,2))
plot(reg,xvar="lambda",col=c("blue","red"),lwd=2)
abline(v=exp(-2.8))
plot_lambda(exp(-2.8))
```

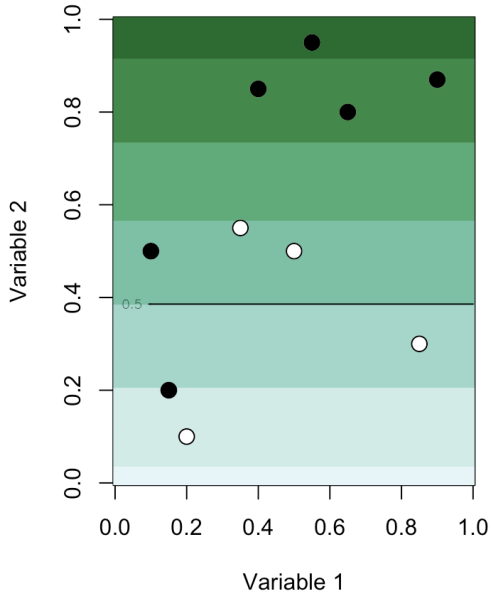




Classification : Penalized Logistic Regression, ℓ_2 norm (LASSO)

```
reg = glmnet(cbind(df0$x1,df0$
    x2), df0$y==1, alpha=1)
par(mfrow=c(1,2))
plot(reg,xvar="lambda",col=c("
    blue","red"),lwd=2)
abline(v=exp(-2.1))
plot_lambda(exp(-2.1))
```





Classification : (Linear) Fisher's Discrimination

Consider the follwing naive classification rule

$$m^\star(\boldsymbol{x}) = \text{argmin}_y\{\mathbb{P}[Y = y|\boldsymbol{X} = \boldsymbol{x}]\}$$

or

$$m^\star(\boldsymbol{x}) = \text{argmin}_y \left\{ \frac{\mathbb{P}[\boldsymbol{X} = \boldsymbol{x}|Y = y]}{\mathbb{P}[\boldsymbol{X} = \boldsymbol{x}]} \right\}$$

(where $\mathbb{P}[\boldsymbol{X} = \boldsymbol{x}]$ is the density in the continuous case).

In the case where y takes two values, that will be standard $\{0, 1\}$ here, one can rewrite the later as

$$m^\star(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \mathbb{E}(Y|\boldsymbol{X} = \boldsymbol{x}) > \dfrac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

Classification : (Linear) Fisher's Discrimination

the set

$$\mathcal{D}_S = \left\{ \boldsymbol{x}, \mathbb{E}(Y|\boldsymbol{X}=\boldsymbol{x}) = \frac{1}{2} \right\}$$

is called the decision boundary.

Assume that $\boldsymbol{X}|Y=0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma})$ and $\boldsymbol{X}|Y=1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$, then explicit expressions can be derived.

$$m^\star(\boldsymbol{x}) = \begin{cases} 1 \text{ if } r_1^2 < r_0^2 + 2\log\dfrac{\mathbb{P}(Y=1)}{\mathbb{P}(Y=0)} + \log\dfrac{|\boldsymbol{\Sigma}_0|}{|\boldsymbol{\Sigma}_1|} \\ 0 \text{ otherwise} \end{cases}$$

where r_y^2 is the Manalahobis distance,

$$r_y^2 = [\boldsymbol{X} - \boldsymbol{\mu}_y]^\mathsf{T} \boldsymbol{\Sigma}_y^{-1} [\boldsymbol{X} - \boldsymbol{\mu}_y]$$

Classification : (Linear) Fisher's Discrimination

Let δ_y be defined as

$$\delta_y(\boldsymbol{x}) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_y| - \frac{1}{2}[\boldsymbol{x}-\boldsymbol{\mu}_y]^\mathsf{T}\boldsymbol{\Sigma}_y^{-1}[\boldsymbol{x}-\boldsymbol{\mu}_y] + \log\mathbb{P}(Y=y)$$

the decision boundary of this classifier is

$$\{\boldsymbol{x} \text{ such that } \delta_0(\boldsymbol{x}) = \delta_1(\boldsymbol{x})\}$$

which is quadratic in $\boldsymbol{x}$. This is the quadratic discriminant analysis.

In Fisher (1936, The use of multile measurements in taxinomic problems , it was assumed that $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_1$

In that case, actually,

$$\delta_y(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y - \frac{1}{2}\boldsymbol{\mu}_y^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y + \log\mathbb{P}(Y=y)$$

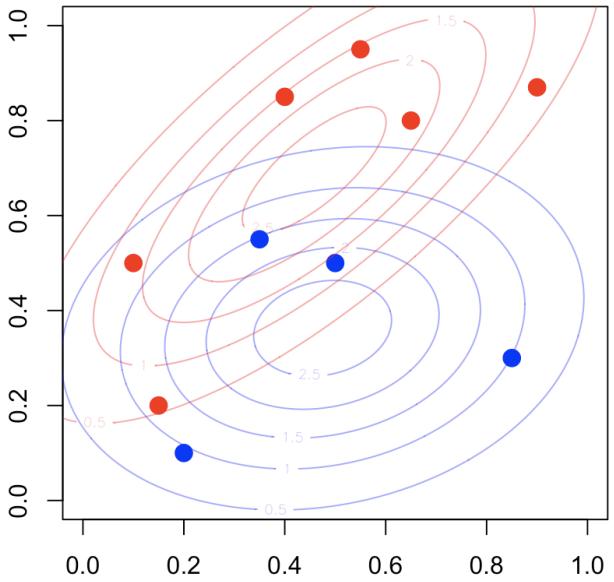
and the decision frontier is now linear in $\boldsymbol{x}$.

Classification : (Linear) Fisher's Discrimination

```
m0 = apply(myocarde[myocarde$PRONO=="0",1:7],2,mean)
m1 = apply(myocarde[myocarde$PRONO=="1",1:7],2,mean)
Sigma = var(myocarde[,1:7])
omega = solve(Sigma)%*%(m1-m0)
omega
                    [,1]
FRCAR  -0.012909708542
INCAR   1.088582058796
INSYS  -0.019390084344
PRDIA  -0.025817110020
PAPUL   0.020441287970
PVENT  -0.038298291091
REPUL  -0.001371677757
b = (t(m1)%*%solve(Sigma)%*%m1-t(m0)%*%solve(Sigma)%*%m0)/2
```

Classification : (Linear) Fisher's Discrimination

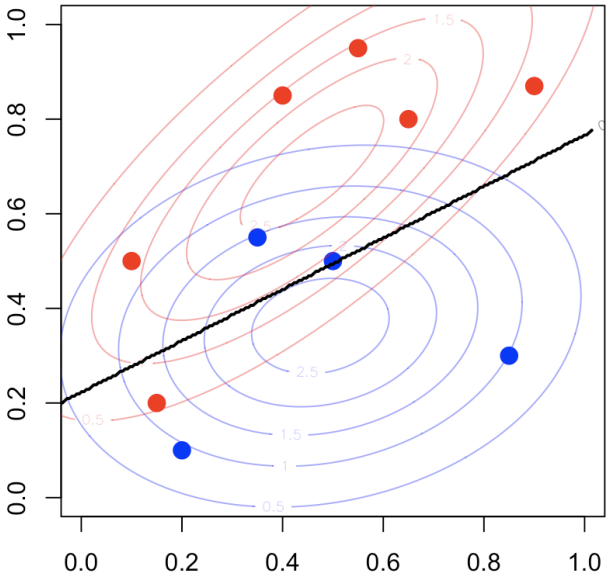
```
x = c(.4,.55,.65,.9,.1,.35,.5,.15,.2,.85)
y = c(.85,.95,.8,.87,.5,.55,.5,.2,.1,.3)
z = c(1,1,1,1,1,0,0,1,0,0)
df = data.frame(x1=x,x2=y,y=as.factor(z))
m0 = apply(df[df$y=="0",1:2],2,mean)
m1 = apply(df[df$y=="1",1:2],2,mean)
Sigma = var(df[,1:2])
omega = solve(Sigma)%*%(m1-m0)
omega
           [,1]
x1 -2.640613174
x2  4.858705676
```



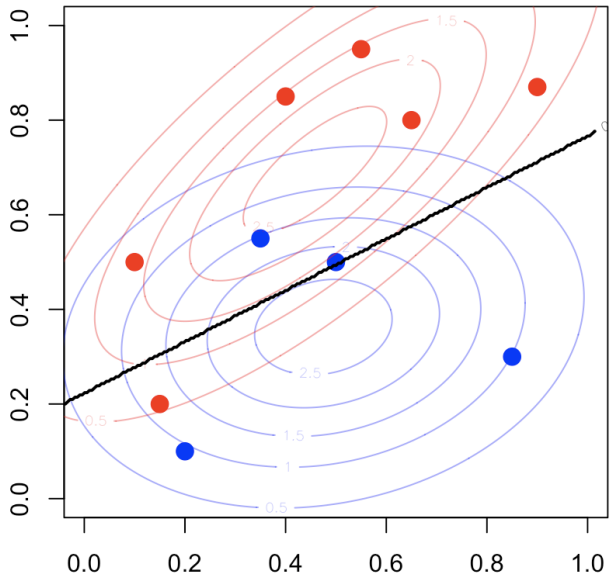
```
library(MASS)
fit_lda = lda(y ~x1+x2 , data=df)
fit_lda

Coefficients of linear discriminants:
             LD1
x1 -2.588389554
x2  4.762614663
```

and for the intercept

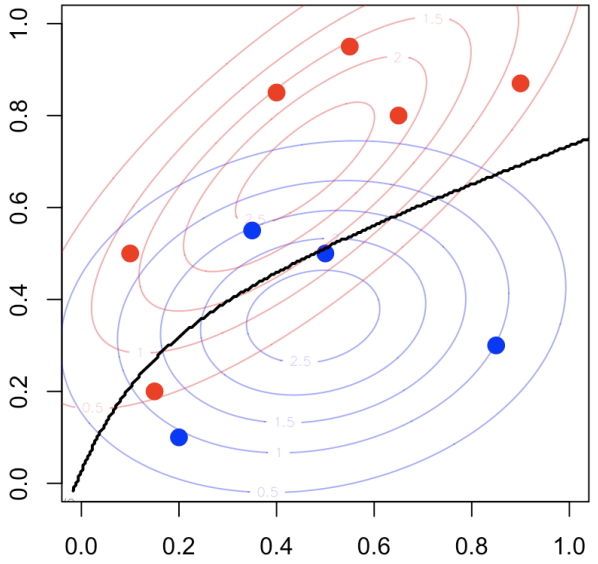
```
b = (t(m1)%*%solve(Sigma)%*%m1-t(m0)%*%solve(
    Sigma)%*%m0)/2
```



```
predlda = function(x,y) predict(fit_lda, data.
    frame(x1=x,x2=y))$class==1
vv=outer(vu,vu,predlda)
contour(vu,vu,vv,add=TRUE,lwd=2,levels = .5)
```




```
fit_qda = qda(y ~x1+x2 , data=df)
plot(df$x1,df$x2,pch=19,
col=c("blue","red")[1+(df$y=="1")])
predqda=function(x,y) predict(fit_qda, data.
    frame(x1=x,x2=y))$class==1
vv=outer(vu,vu,predlda)
contour(vu,vu,vv,add=TRUE,lwd=2,levels = .5)
```



Note that there are strong links with the logistic regression.

Assume as previously that $\boldsymbol{X}|Y=0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma})$ and $\boldsymbol{X}|Y=1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ then

$$\log \frac{\mathbb{P}(Y=1|\boldsymbol{X}=\boldsymbol{x})}{\mathbb{P}(Y=0|\boldsymbol{X}=\boldsymbol{x})}$$

is equal to

$$\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}[\boldsymbol{\mu}_y] - \frac{1}{2}[\boldsymbol{\mu}_1-\boldsymbol{\mu}_0]^\mathsf{T}\boldsymbol{\Sigma}^{-1}[\boldsymbol{\mu}_1-\boldsymbol{\mu}_0] + \log \frac{\mathbb{P}(Y=1)}{\mathbb{P}(Y=0)}$$

which is linear in $\boldsymbol{x}$

$$\log \frac{\mathbb{P}(Y=1|\boldsymbol{X}=\boldsymbol{x})}{\mathbb{P}(Y=0|\boldsymbol{X}=\boldsymbol{x})} = \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}$$

Hence, when each groups have Gaussian distributions with identical variance matrix, then LDA and the logistic regression lead to the same classification rule.

Classification : (Linear) Fisher's Discrimination

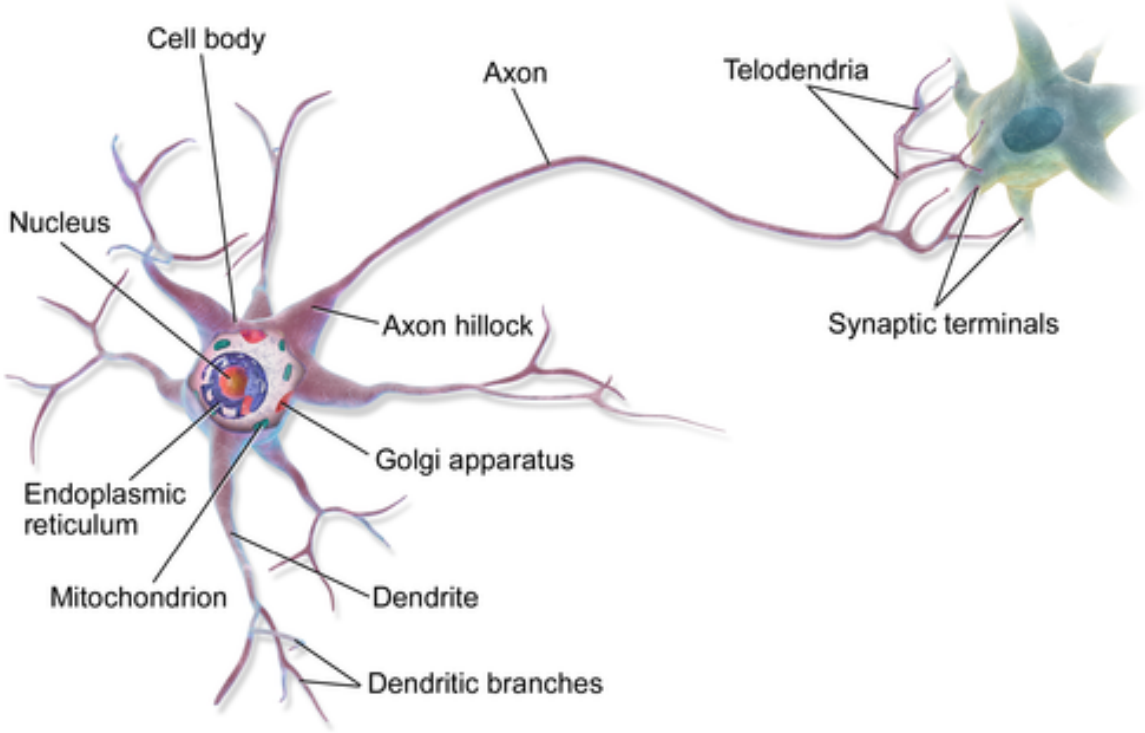
Observe furthermore that the slope is proportional to $\boldsymbol{\Sigma}^{-1}[\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0]$ as stated in Fisher's article.

But to obtain such a relationship, he observe that the ratio of between and within variances (in the two groups) was

$$\frac{\text{variance between}}{\text{variance within}} = \frac{[\boldsymbol{\omega}\boldsymbol{\mu}_1 - \boldsymbol{\omega}\boldsymbol{\mu}_0]^2}{\boldsymbol{\omega}^\mathsf{T}\boldsymbol{\Sigma}_1\boldsymbol{\omega} + \boldsymbol{\omega}^\mathsf{T}\boldsymbol{\Sigma}_0\boldsymbol{\omega}}$$

which is maximal when $\boldsymbol{\omega}$ is proportional to $\boldsymbol{\Sigma}^{-1}[\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0]$.

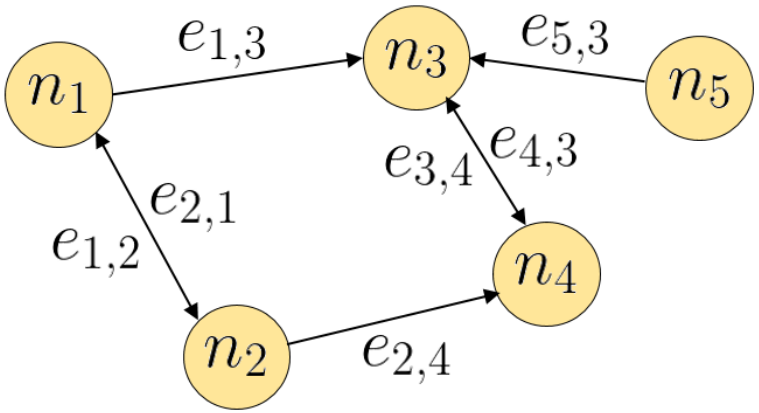
Classification : Neural Nets



The most interesting part is not 'neural' but 'network'.

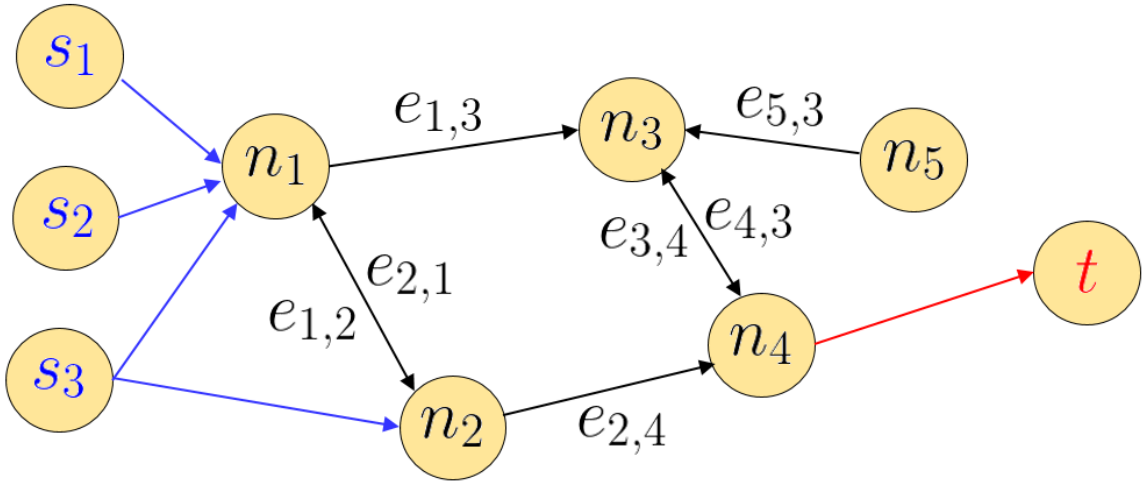
Classification : Neural Nets

Networks have nodes, and edges (possibly connected) that connect nodes,



or maybe, to more specific some sort of flow network,

Classification : Neural Nets



In such a network, we usually have sources (here multiple) sources (here $\{s_1, s_2, s_3\}$), on the left, on a sink (here $\{t\}$), on the right. To continue with this metaphorical introduction, information from the sources should reach the sink. An usually, sources are explanatory variables, $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_p\}$, and the sink is our variable of interest $\boldsymbol{y}$.

The output here is a binary variable $y \in \{0, 1\}$

Consider the sigmoid function $f(x) = \dfrac{1}{1+e^{-x}} = \dfrac{e^x}{e^x+1}$ which is actually the

logistic function.

```
1 sigmoid = function(x) 1 / (1 + exp(-x))
```

This function f is called the activation function.

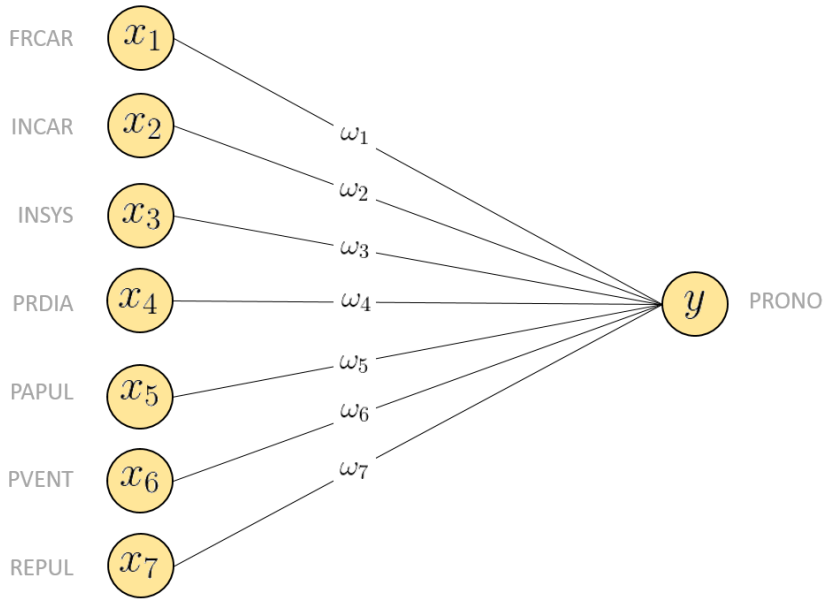
If $y \in \{-1, +1\}$, one can consider the hyperbolic tangent $f(x) = \tanh(x) = \dfrac{(e^x - e^{-x})}{(e^x + e^{-x})}$ or the inverse tangent function $f(x) = \tan^{-1}(x)$.

And as input for such function, we consider a weighted sum of incoming nodes. So here

$$y_i = f\left(\sum_{j=1}^{p} \omega_j x_{j,i}\right)$$

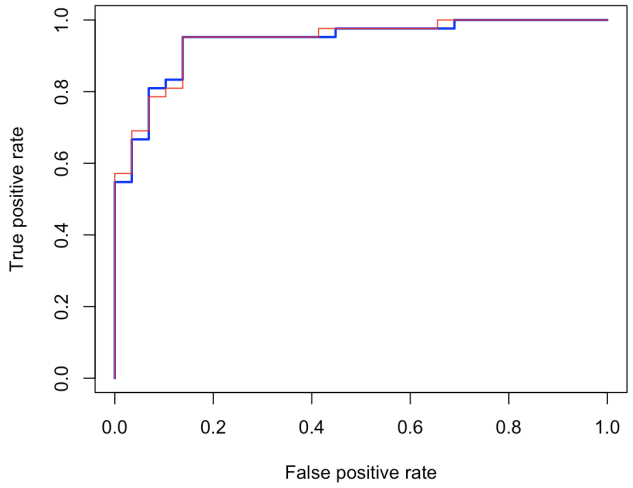
We can also add a constant actually

$$y_i = f\left(\omega_0 + \sum_{j=1}^{p} \omega_j x_{j,i}\right)$$



Classification : Neural Nets

```
weights_0 = lm(PRONO~.,data=myocarde)$
    coefficients
X = as.matrix(cbind(1,myocarde[,1:7]))
y_5_1 = sigmoid(X %*% weights_0)
library(ROCR)
pred = ROCR::prediction(y_5_1,myocarde$PRONO)
perf = ROCR::performance(pred,"tpr", "fpr")
plot(perf,col="blue",lwd=2)
reg = glm(PRONO~.,data=myocarde,family=
    binomial(link = "logit"))
y_0 = predict(reg,type="response")
pred0 = ROCR::prediction(y_0,myocarde$PRONO)
perf0 = ROCR::performance(pred0,"tpr", "fpr")
plot(perf0,add=TRUE,col="red")
```



Classification : Neural Nets

The error of that model can be computed used the quadratic loss function,

$$\sum_{i=1}^{n}(y_i - p(\boldsymbol{x}_i))^2$$

or cross-entropy

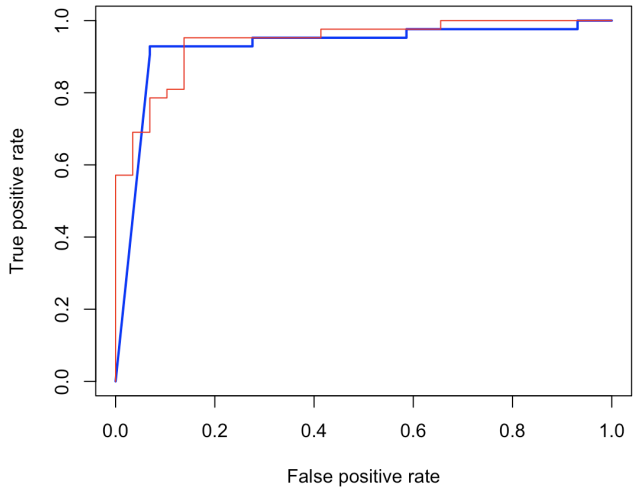
$$\sum_{i=1}^{n}(y_i \log p(\boldsymbol{x}_i) + [1-y_i]\log[1-p(\boldsymbol{x}_i)])$$

```
loss = function(weights){
  mean( (myocarde$PRONO-sigmoid(X %*% weights))^2) }
```

we want to solve

$$\boldsymbol{\omega}^\star = \text{argmin}\left\{\frac{1}{n}\sum_{i=1}^{n}\ell\left(y_i, f(\omega_0 + \boldsymbol{x}_i{}^\mathsf{T}\boldsymbol{\omega})\right)\right\}$$

```
weights_1 = optim(weights_0,loss)$par
y_5_2 = sigmoid(X %*% weights_1)
pred = ROCR::prediction(y_5_2,myocarde$PRONO)
perf = ROCR::performance(pred,"tpr", "fpr")
plot(perf,col="blue",lwd=2)
plot(perf0,add=TRUE,col="red")
```

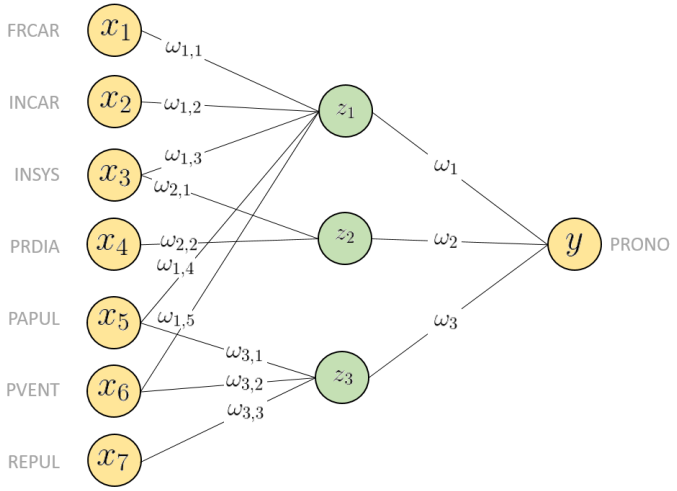


Consider a single hidden layer, with 3 different neurons, so that

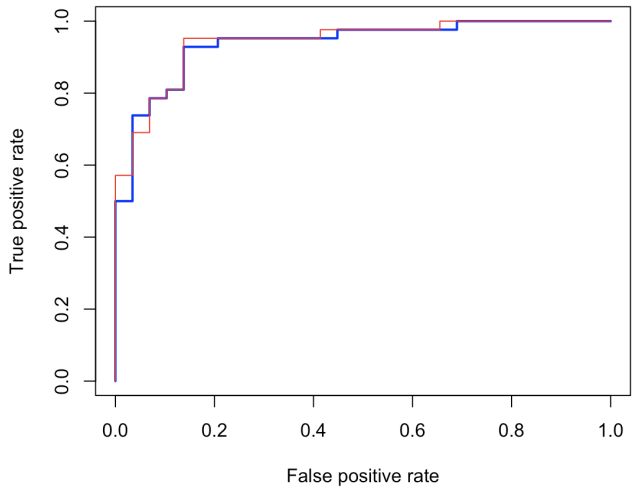
$$p(\boldsymbol{x}) = f\left(\omega_0 + \sum_{h=1}^{3} \omega_h f_h\left(\omega_{h,0} + \sum_{j=1}^{p} \omega_{h,j} x_j\right)\right)$$

or

$$p(\boldsymbol{x}) = f\left(\omega_0 + \sum_{h=1}^{3} \omega_h f_h\left(\omega_{h,0} + \boldsymbol{x}^\mathsf{T}\boldsymbol{\omega}_h\right)\right)$$



```
weights_1 = lm(PRONO~1+FRCAR+INCAR+INSYS+PAPUL
    +PVENT,data=myocarde)$coefficients
X1 = as.matrix(cbind(1,myocarde[,c("FRCAR","
    INCAR","INSYS","PAPUL","PVENT")]))
weights_2 = lm(PRONO~1+INSYS+PRDIA,data=
    myocarde)$coefficients
X2 = as.matrix(cbind(1,myocarde[,c("INSYS","
    PRDIA")]))
weights_3 = lm(PRONO~1+PAPUL+PVENT+REPUL,data=
    myocarde)$coefficients
X3 = as.matrix(cbind(1,myocarde[,c("PAPUL","
    PVENT","REPUL")]))
X = cbind(sigmoid(X1 %*% weights_1), sigmoid(
    X2 %*% weights_2), sigmoid(X3 %*% weights_
    3))
weights = c(1/3,1/3,1/3)
y_5_3 = sigmoid(X %*% weights)
pred = ROCR::prediction(y_5_3,myocarde$PRONO)
perf = ROCR::performance(pred,"tpr","fpr")
plot(perf,col="blue",lwd=2)
```



If

$$p(\boldsymbol{x}) = f\left(\omega_0 + \sum_{h=1}^{3} \omega_h f_h\left(\omega_{h,0} + \boldsymbol{x}^\mathsf{T}\boldsymbol{\omega}_h\right)\right)$$

we want to solve, with back propagation,

$$\boldsymbol{\omega}^\star = \text{argmin}\left\{\frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, p(\boldsymbol{x}_i)\right)\right\}$$

For practical issues, let us center all variables

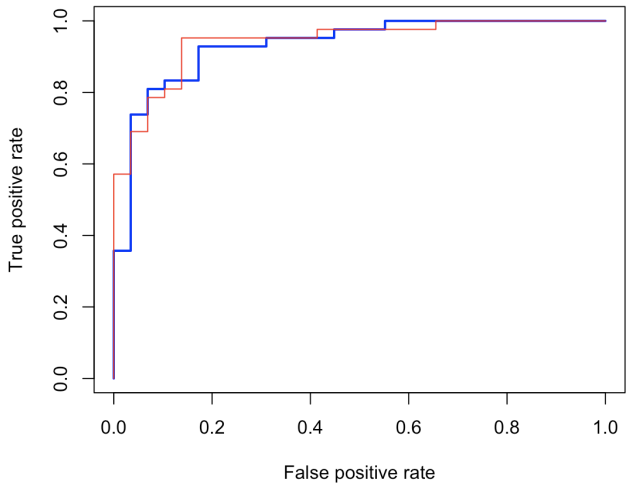
```
center = function(z) (z-mean(z))/sd(z)
```

The loss function is here

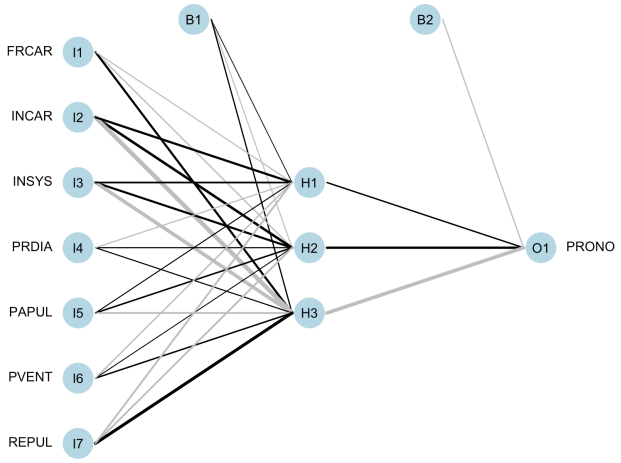
```
loss = function(weights){
weights_1 = weights[0+(1:7)]
weights_2 = weights[7+(1:7)]
weights_3 = weights[14+(1:7)]
weights_  = weights[21+1:4]
```

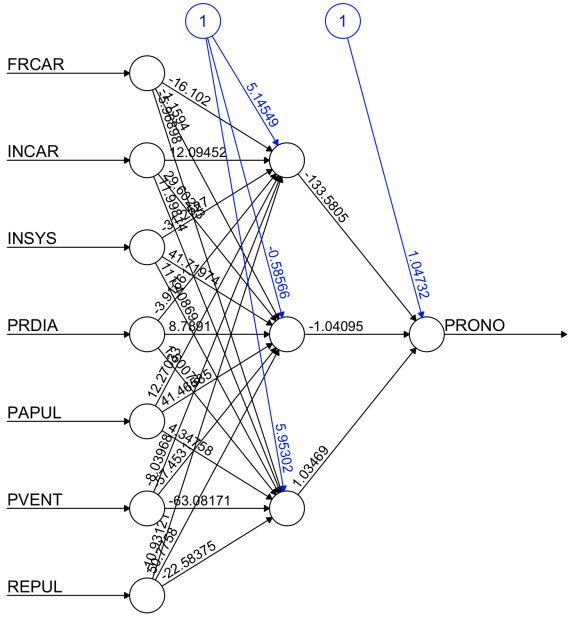
```
X1=X2=X3=as.matrix(myocarde[,1:7])
Z1 = center(X1 %*% weights_1)
Z2 = center(X2 %*% weights_2)
Z3 = center(X3 %*% weights_3)
X = cbind(1,sigmoid(Z1), sigmoid(Z2), sigmoid(Z3))
mean( (myocarde$PRONO-sigmoid(X %*% weights_))^2)}
```

```
pca = princomp(myocarde[,1:7])
W = get_pca_var(pca)$contrib
weights_0 = c(W[,1],W[,2],W[,3],c(-1,rep(1,3)/
    3))
weights_opt = optim(weights_0,loss)$par
weights_1 = weights_opt[0+(1:7)]
weights_2 = weights_opt[7+(1:7)]
weights_3 = weights_opt[14+(1:7)]
weights_  = weights_opt[21+1:4]
X1=X2=X3=as.matrix(myocarde[,1:7])
Z1 = center(X1 %*% weights_1)
Z2 = center(X2 %*% weights_2)
Z3 = center(X3 %*% weights_3)
X = cbind(1,sigmoid(Z1), sigmoid(Z2), sigmoid(
    Z3))
y_5_4 = sigmoid(X %*% weights_)
pred = ROCR::prediction(y_5_4,myocarde$PRONO)
perf = ROCR::performance(pred,"tpr", "fpr")
plot(perf,col="blue",lwd=2)
plot(perf,add=TRUE,col="red")
```



```
library(nnet)
myocarde_minmax = myocarde
minmax = function(z) (z-min(z))/(max(z)-min(z)
    )
for(j in 1:7) myocarde_minmax[,j] = minmax(
    myocarde_minmax[,j])
model_nnet = nnet(PRONO~.,data=myocarde_minmax
    ,size=3)
summary(model_nnet)
library(devtools)
source_url('https://gist.githubusercontent.com
    /fawda123/7471137/raw/466
    c1474d0a505ff044412703516c34f1a4684a5/nnet
    _plot_update.r')
plot.nnet(model_nnet)
library(neuralnet)
model_nnet = neuralnet(formula(glm(PRONO~.,
    data=myocarde_minmax)),
myocarde_minmax,hidden=3, act.fct = sigmoid)
plot(model_nnet)
```





Classification : Neural Nets

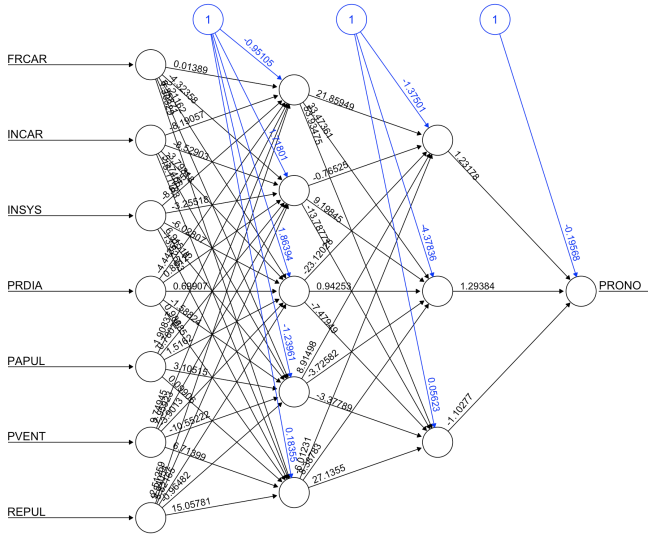
Of course, one can consider a multiple layer network,

$$p(\boldsymbol{x}) = f\left(\omega_0 + \sum_{h=1}^{3} \omega_h f_h\left(\omega_{h,0} + \boldsymbol{z}_h{}^{\mathsf{T}}\boldsymbol{\omega}_h\right)\right)$$

where

$$\boldsymbol{z}_h = f\left(\omega_{h,0} + \sum_{j=1}^{k_h} \omega_{h,j} f_{h,j}\left(\omega_{h,j,0} + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\omega}_{h,j}\right)\right)$$

```
1 library(neuralnet)
2 model_nnet = neuralnet(formula(glm(PRONO~.,
      data=myocarde_minmax)),
3 myocarde_minmax,hidden=3, act.fct = sigmoid)
4 plot(model_nnet)
```



Classification : Support Vector Machine (SVM)

Here y takes values in $\{-1, +1\}$. Our model will be $m(\boldsymbol{x}) = \text{sign}[\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b]$.

Thus, the space is divided by a (linear) border

$$\Delta : \{\boldsymbol{x} \in \mathbb{R}^p : \boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b = 0\}$$

The distance from point $\boldsymbol{x}_i$ to Δ is

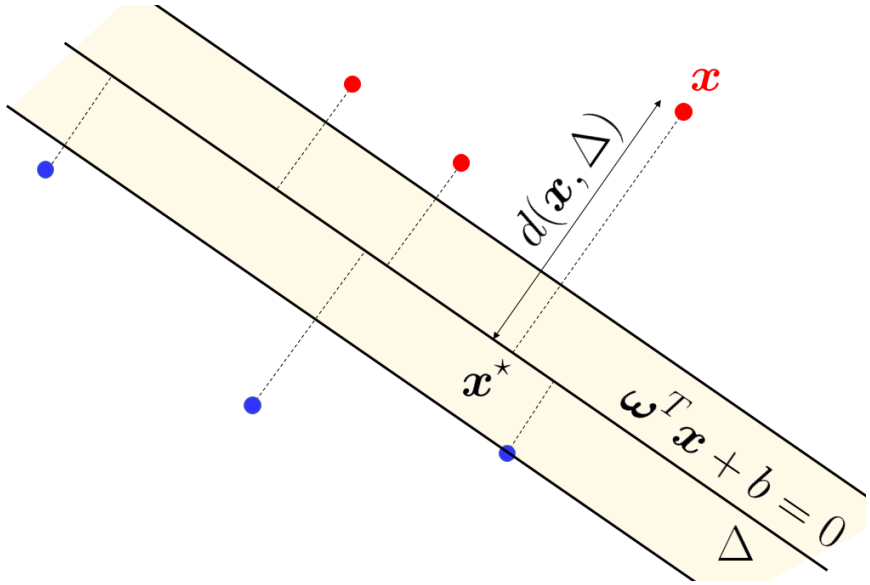
$$d(\boldsymbol{x}_i, \Delta) = \frac{\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b}{\|\boldsymbol{\omega}\|}$$

If the space is linearly separable, the problem is ill posed (there is an infinite number of solutions).

Classification : Support Vector Machine (SVM)

So consider

$$\max_{\boldsymbol{\omega},b}\left\{\min_{i=1,\cdots,n}\{\text{distance}(\boldsymbol{x}_i,\Delta)\}\right\}$$



The strategy is to maximize the margin. One can prove that we want to solve

$$\max_{\boldsymbol{\omega},m}\left\{\frac{m}{\|\boldsymbol{\omega}\|}\right\}$$

subject to $y_i\cdot(\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i)=m,\ \forall i=1,\cdots,n.$

Classification : Support Vector Machine (SVM)

Again, the problem is ill posed (non identifiable), and we can consider $m = 1$

$$\max_{\boldsymbol{\omega}} \left\{ \frac{1}{\|\boldsymbol{\omega}\|} \right\}$$

subject to $y_i \cdot (\boldsymbol{\omega}^\mathsf{T} \boldsymbol{x}_i) = 1, \forall i = 1, \cdots, n$. The optimization objective can be written

$$\min_{\boldsymbol{\omega}} \left\{ \|\boldsymbol{\omega}\|^2 \right\}$$

Classification : Support Vector Machine (SVM) - Primal Problem

In the separable case, consider the following primal problem,

$$\min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}} \left\{ \frac{1}{2}\|\boldsymbol{\omega}\|^2 \right\}$$

subject to $y_i \cdot (\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) \geq 1, \forall i = 1, \cdots, n$

In the non-separable case, introduce slack (error) variables $\boldsymbol{\xi}$: if $y_i \cdot (\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) \geq 1$, there is no error $\xi_i = 0$.

Let C denote the cost of mis-classification. The optimization problem becomes

$$\min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} \left\{ \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n}\xi_i \right\}$$

subject to $y_i \cdot (\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) \geq 1 - \xi_i$, with $\xi_i \geq 0, \forall i = 1, \cdots, n$

```
n = length(myocarde[,"PRONO"])
myocarde0 = myocarde
myocarde0$PRONO = myocarde$PRONO*2-1
```

```
C = .5
f = function(param){
  w  = param[1:7]
  b  = param[8]
  xi = param[8+1:nrow(myocarde)]
  .5*sum(w^2) + C*sum(xi)}
grad_f = function(param){
  w  = param[1:7]
  b  = param[8]
  xi = param[8+1:nrow(myocarde)]
  c(2*w,0,rep(C,length(xi)))}
```

(linear) constraints are written as $\boldsymbol{U\theta} - \boldsymbol{c} \geq \boldsymbol{0}$

```
U = rbind(cbind(myocarde0[,"PRONO"]*as.matrix(myocarde[,1:7]),diag(n)
    ,myocarde0[,"PRONO"]),
cbind(matrix(0,n,7),diag(n,n),matrix(0,n,1)))
C = c(rep(1,n),rep(0,n))
```

then use

```
constrOptim(theta=p_init, f, grad_f, ui = U,ci = C)
```

One can recognize in the separable case, but also in the non-separable case, a classic quadratic program

$$\min_{\boldsymbol{z}\in\mathbb{R}^d}\left\{\frac{1}{2}\boldsymbol{z}^\top\boldsymbol{D}\boldsymbol{z}-\boldsymbol{d}\boldsymbol{z}\right\}$$

subject to $\boldsymbol{A}\boldsymbol{z}\geq\boldsymbol{b}$.

```
library(quadprog)
eps = 5e-4
y = myocarde[,&quot;PRONO&quot;]*2-1
X = as.matrix(cbind(1,myocarde[,1:7]))
n = length(y)
D = diag(n+7+1)
diag(D)[8+0:n] = 0
d = matrix(c(rep(0,7),0,rep(C,n)), nrow=n+7+1)
A = Ui
b = Ci
```

Classification : Support Vector Machine (SVM) - Primal Problem

```
sol = solve.QP(D+eps*diag(n+7+1), d, t(A), b, meq=1)
qpsol = sol$solution
(omega = qpsol[1:7])
[1] -0.106642005446 -0.002026198103 -0.022513312261 -0.018958578746
    -0.023105767847 -0.018958578746 -1.080638988521
(b     = qpsol[n+7+1])
[1] 997.6289927
```

Given an observation $\boldsymbol{x}$, the prediction is $y = \text{sign}[\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b]$

```
y_pred = 2*((as.matrix(myocarde0[,1:7])%*%omega+b)>0)-1
```

Classification : Support Vector Machine (SVM) - Dual Problem

The Lagrangian of the separable problem could be written introducing Lagrange multipliers $\boldsymbol{\alpha} \in \mathbb{R}^n$, $\boldsymbol{\alpha} \geq \mathbf{0}$ as

$$\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i \big(y_i(\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) - 1\big)$$

Somehow, α_i represents the influence of the observation $(y_i, \boldsymbol{x}_i)$. Consider the Dual Problem, with $\boldsymbol{G} = [G_{ij}]$ and $G_{ij} = y_i y_j \boldsymbol{x}_j{}^\mathsf{T}\boldsymbol{x}_i$.

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left\{ \frac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{G}\boldsymbol{\alpha} - \mathbf{1}^\mathsf{T}\boldsymbol{\alpha} \right\}$$

subject to $\boldsymbol{y}^\mathsf{T}\boldsymbol{\alpha} = \mathbf{0}$.

The Lagrangian of the non-separable problem could be written introducing Lagrange multipliers $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^n$, $\boldsymbol{\alpha}, \boldsymbol{\beta} \geq \mathbf{0}$,

Classification : Support Vector Machine (SVM) - Dual Problem

and define the Lagrangian $\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ as

$$\frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y_i(\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x}_i + b) - 1 + \xi_i\right) - \sum_{i=1}^{n}\beta_i\xi_i$$

l

Somehow, α_i represents the influence of the observation $(y_i, \boldsymbol{x}_i)$. The Dual Problem become with $\boldsymbol{G} = [G_{ij}]$ and $G_{ij} = y_i y_j \boldsymbol{x}_j{}^\mathsf{T}\boldsymbol{x}_i$.

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n}\left\{\frac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{G}\boldsymbol{\alpha} - \mathbf{1}^\mathsf{T}\boldsymbol{\alpha}\right\}$$

subject to $\boldsymbol{y}^\mathsf{T}\boldsymbol{\alpha} = \mathbf{0}$, $\boldsymbol{\alpha} \geq \mathbf{0}$ and $\boldsymbol{\alpha} \leq C$. As previously, one can also use quadratic programming

Classification : Support Vector Machine (SVM) - Dual Problem

```
library(quadprog)
eps = 5e-4
y = myocarde[,"PRONO"]*2-1
X = as.matrix(cbind(1,myocarde[,1:7]))
n = length(y)
Q = sapply(1:n, function(i) y[i]*t(X)[,i])
D = t(Q)%*%Q
d = matrix(1, nrow=n)
A = rbind(y,diag(n),-diag(n))
C = .5
b = c(0,rep(0,n),rep(-C,n))
sol = solve.QP(D+eps*diag(n), d, t(A), b, meq=1, factorized=FALSE)
qpsol = sol$solution
```

Classification : Support Vector Machine (SVM) - Dual Problem

The two problems are connected in the sense that for all
$\boldsymbol{x}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{x} + b = \sum_{i=1}^{n} \alpha_i y_i (\boldsymbol{x}^\mathsf{T}\boldsymbol{x}_i) + b$

To recover the solution of the primal problem, $\boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ thus

```
omega = apply(qpsol*y*X,2,sum)
omega
                             1          FRCAR          INCAR          INSYS
 0.00000000000000243907  0.0550138658 -0.092016323  0.3609571899422
                         PRDIA          PAPUL          PVENT          REPUL
-0.109401796528869235669 -0.0485213403 -0.066005864  0.0010093656567
```

More generally

```
svm.fit = function(X, y, C=NULL) {
 n.samples = nrow(X)
 n.features = ncol(X)
 K = matrix(rep(0, n.samples*n.samples), nrow=n.samples)
 for (i in 1:n.samples){
```

```
  for (j in 1:n.samples){
    K[i,j] = X[i,] %*% X[j,] }}
 Dmat = outer(y,y) * K
 Dmat = as.matrix(nearPD(Dmat)$mat)
 dvec = rep(1, n.samples)
 Amat = rbind(y, diag(n.samples), -1*diag(n.samples))
 bvec = c(0, rep(0, n.samples), rep(-C, n.samples))
 res = solve.QP(Dmat,dvec,t(Amat),bvec=bvec, meq=1)
 a = res$solution
 bomega = apply(a*y*X,2,sum)
 return(bomega)
}
```

Classification : Support Vector Machine (SVM) - Dual Problem

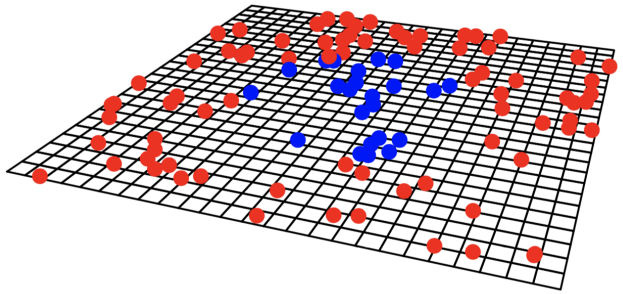
To relate this duality optimization problem to OLS, recall that $y = \boldsymbol{x}^\mathsf{T}\boldsymbol{\omega} + \varepsilon$, so that $\widehat{y} = \boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\omega}}$, where $\widehat{\boldsymbol{\omega}} = [\boldsymbol{X}^\mathsf{T}\boldsymbol{X}]^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}$.

But one can also write $y = \boldsymbol{x}^\mathsf{T}\widehat{\boldsymbol{\omega}} = \sum_{i=1}^n \widehat{\alpha}_i \cdot \boldsymbol{x}^\mathsf{T}\boldsymbol{x}_i$ where $\widehat{\boldsymbol{\alpha}} = \boldsymbol{X}[\boldsymbol{X}^\mathsf{T}\boldsymbol{X}]^{-1}\widehat{\boldsymbol{\omega}}$, or conversely, $\widehat{\boldsymbol{\omega}} = \boldsymbol{X}^\mathsf{T}\widehat{\boldsymbol{\alpha}}$.

Classification : Support Vector Machine (SVM) - Kernel Approach

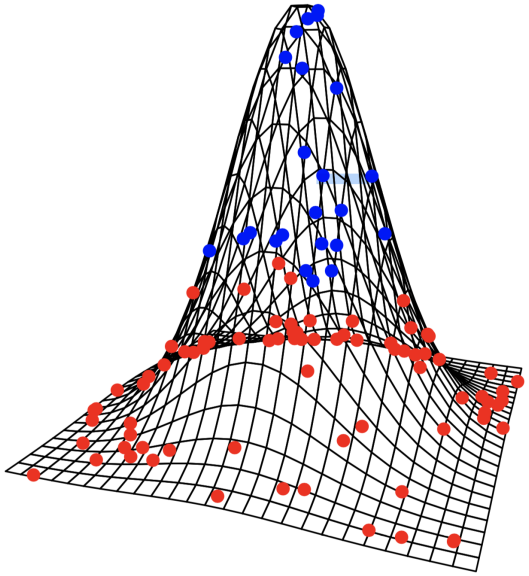
A positive kernel on $\mathcal{X}$ is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ symmetric, and such that for any n, $\forall \alpha_1, \cdots, \alpha_n$ and $\forall \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$$



For example, the linear kernel is $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_j$. That's what we've been using here, so far. One can also define the product kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \kappa(\boldsymbol{x}_i) \cdot \kappa(\boldsymbol{x}_j)$ where κ is some function $\mathcal{X} \to \mathbb{R}$.

Classification : Support Vector Machine (SVM) - Kernel Approach



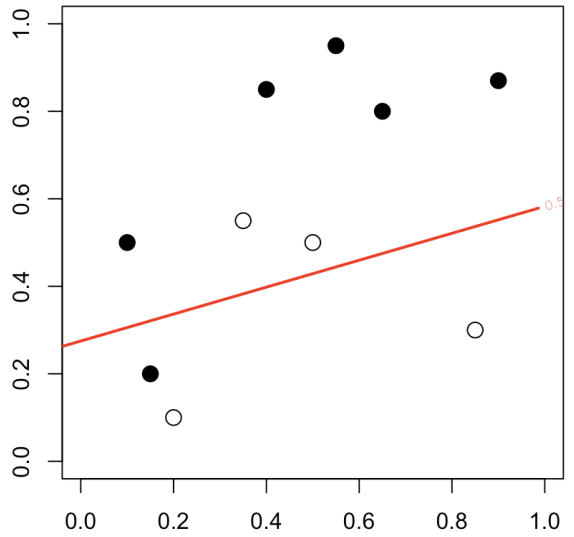
Finally, the Gaussian kernel is $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp[-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2]$.

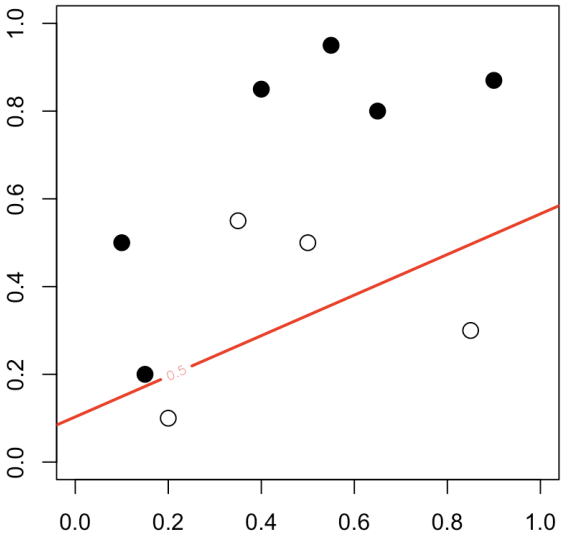
Since it is a function of $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, it is also called a radial kernel.

```
linear.kernel = function(x1, x2) {
 return (x1%*%x2)
}
svm.fit = function(X, y, FUN=linear.kernel, C=NULL) {
```

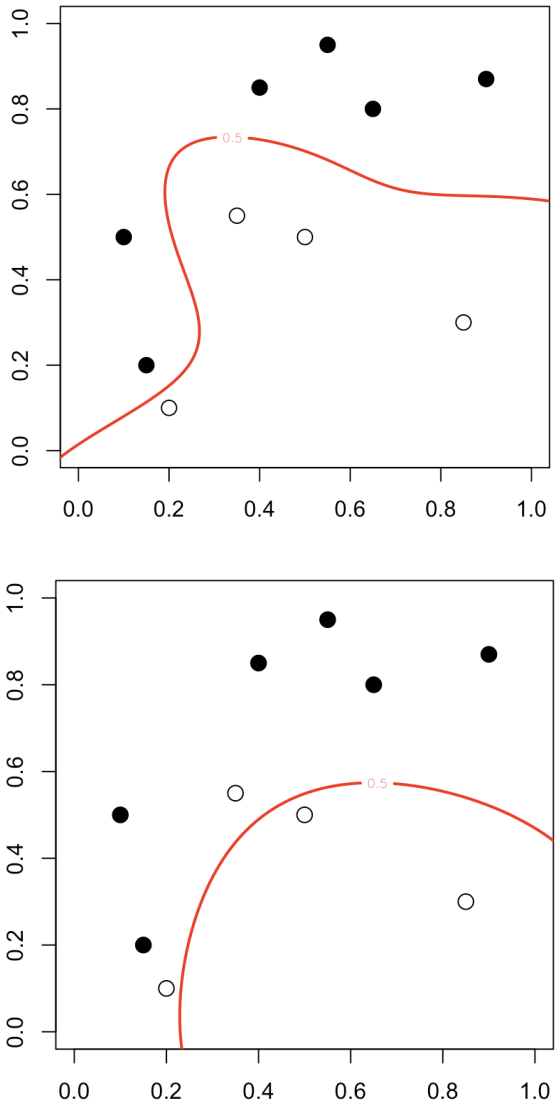
```
  n.samples = nrow(X)
  n.features = ncol(X)
  K = matrix(rep(0, n.samples*n.samples), nrow=n.samples)
  for (i in 1:n.samples){
   for (j in 1:n.samples){
    K[i,j] = FUN(X[i,], X[j,])
   }
  }
  Dmat = outer(y,y) * K
  Dmat = as.matrix(nearPD(Dmat)$mat)
  dvec = rep(1, n.samples)
  Amat = rbind(y, diag(n.samples), -1*diag(n.samples))
  bvec = c(0, rep(0, n.samples), rep(-C, n.samples))
  res = solve.QP(Dmat,dvec,t(Amat),bvec=bvec, meq=1)
  a = res$solution
  bomega = apply(a*y*X,2,sum)
  return(bomega)
}
```

```
SVM2 = ksvm(y ~ x1 + x2, data = df0, C=2,
    kernel = "vanilladot" , prob.model=TRUE,
     type="C-svc")
pred_SVM2 = function(x,y){
return(predict(SVM2,newdata=data.frame(x1=x,
    x2=y), type="probabilities")[,2])}
plot(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],
     cex=1.5,xlab="",
     ylab="",xlim=c(0,1),ylim=c(0,1))
vu = seq(-.1,1.1,length=251)
vv = outer(vu,vu,function(x,y) pred_SVM2(x,y
    ))
contour(vu,vu,vv,add=TRUE,lwd=2,levels = .5,
    col="red")
```



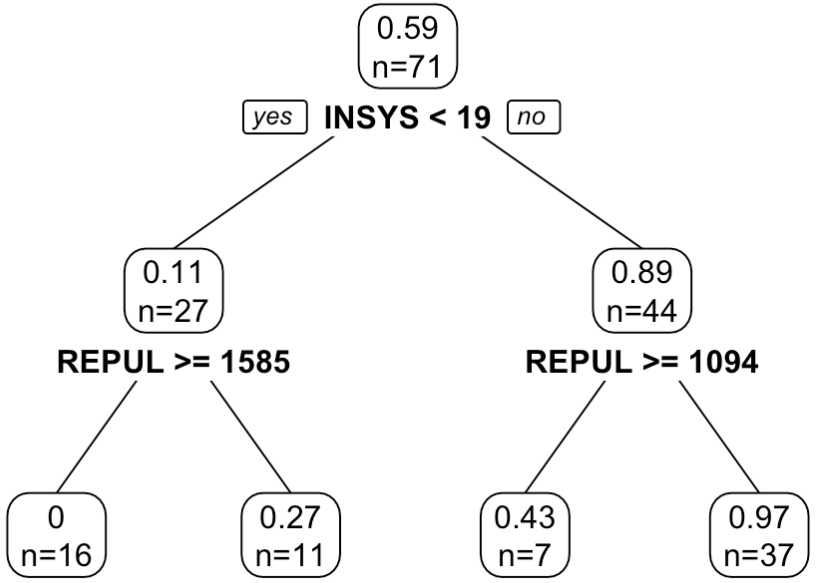


```
SVM3 = ksvm(y ~ x1 + x2, data = df0, C=2,
    kernel = "rbfdot" , prob.model=TRUE,
    type="C-svc")
pred_SVM3 = function(x,y){
return(predict(SVM3,newdata=data.frame(x1=x,
    x2=y), type="probabilities")[,2])}
plot(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],
     cex=1.5,xlab="",
     ylab="",xlim=c(0,1),ylim=c(0,1))
vu = seq(-.1,1.1,length=251)
vv = outer(vu,vu,function(x,y) pred_SVM2(x,y
    ))
contour(vu,vu,vv,add=TRUE,lwd=2,levels = .5,
    col="red")
```



Classification : Classification Trees

```
1 library(rpart)
2 cart = rpart(PRONO~.,data=
      myocarde)
3 library(rpart.plot)
4 prp(cart,type=2,extra=1)
```



A (binary) split is based on one specific variable – say x_j – and a cutoff, say s. Then, there are two options:

- either $x_{i,j} \leq s$, then observation i goes on the left, in $\mathcal{I}_L$
- or $x_{i,j} > s$, then observation i goes on the right, in $\mathcal{I}_R$

Thus, $\mathcal{I} = \mathcal{I}_L \cup \mathcal{I}_R$.

Classification : Classification Trees

Gini for node $\mathcal{I}$ is defined as

$$G(\mathcal{I}) = -\sum_{y\in\{0,1\}} p_y(1-p_y)$$

where p_y is the proportion of individuals in the leaf of type y,

$$G(\mathcal{I}) = -\sum_{y\in\{0,1\}} \frac{n_{y,\mathcal{I}}}{n_{\mathcal{I}}}\left(1-\frac{n_{y,\mathcal{I}}}{n_{\mathcal{I}}}\right)$$

```
gini = function(y,classe){
T. = table(y,classe)
nx = apply(T,2,sum)
n. = sum(T)
pxy = T/matrix(rep(nx,each=2),nrow=2)
omega = matrix(rep(nx,each=2),nrow=2)/n
g. = -sum(omega*pxy*(1-pxy))
return(g)}
```

Classification : Classification Trees

```
-2*mean(myocarde$PRONO)*(1-mean(myocarde$PRONO))
[1] -0.4832375
gini(y=myocarde$PRONO,classe=myocarde$PRONO<Inf)
[1] -0.4832375
gini(y=myocarde$PRONO,classe=myocarde[,1]<=100)
[1] -0.4640415
```

Classification : Classification Trees

if we split, define index

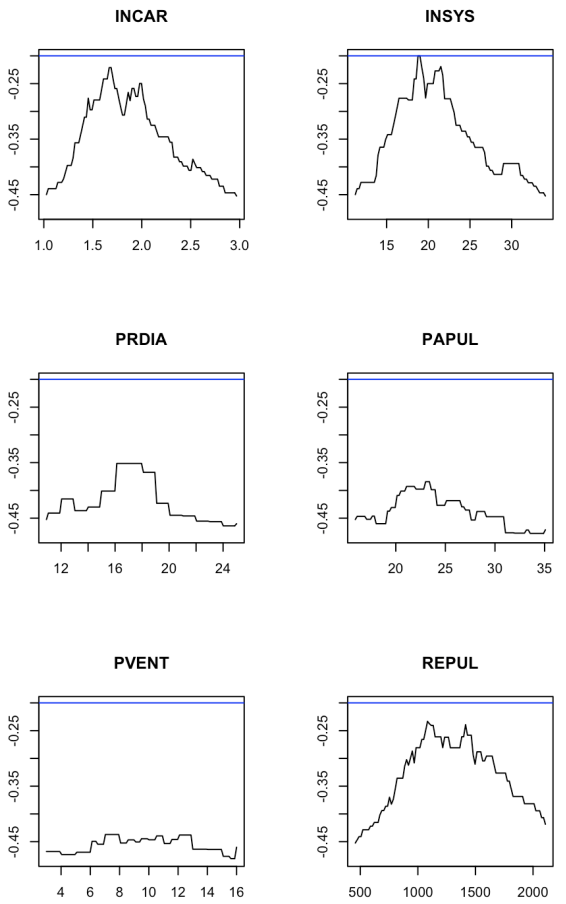
$$G(\mathcal{I}_L, \mathcal{I}_R) = - \sum_{x \in \{L,R\}} \frac{n_x}{n_{\mathcal{I}_x}} n_{\mathcal{I}} \sum_{y \in \{0,1\}} \frac{n_{y,\mathcal{I}_x}}{n_{\mathcal{I}_x}} \left(1 - \frac{n_{y,\mathcal{I}_x}}{n_{\mathcal{I}_x}} \right)$$

the entropic measure is

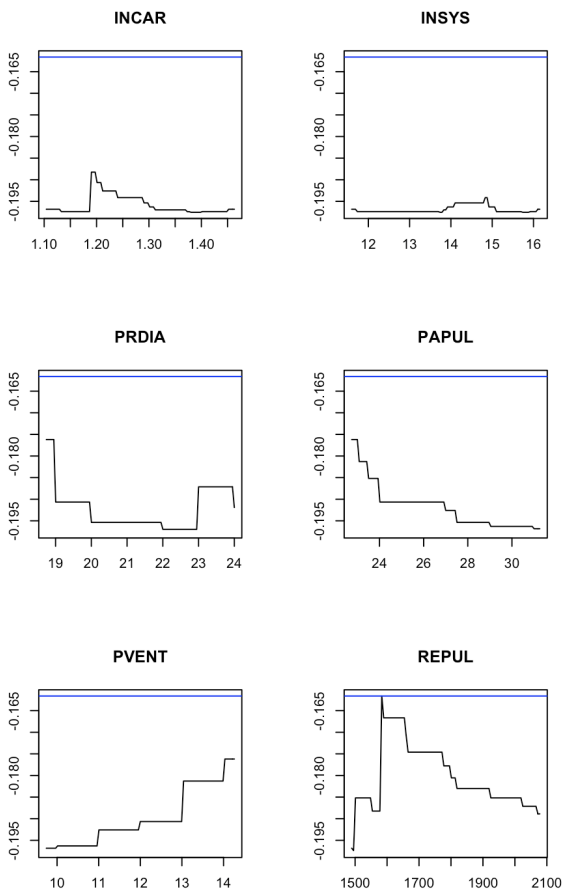
$$E(\mathcal{I}) = - \sum_{y \in \{0,1\}} \frac{n_{y,\mathcal{I}}}{n_{\mathcal{I}}} \log \left(\frac{n_{y,\mathcal{I}}}{n_{\mathcal{I}}} \right)$$

```
entropy = function(y,classe){
  T   = table(y,classe)
  nx  = apply(T,2,sum)
  pxy = T/matrix(rep(nx,each=2),nrow=2)
  omega = matrix(rep(nx,each=2),nrow=2)/sum(T)
  g   = sum(omega*pxy*log(pxy))
return(g)}
```

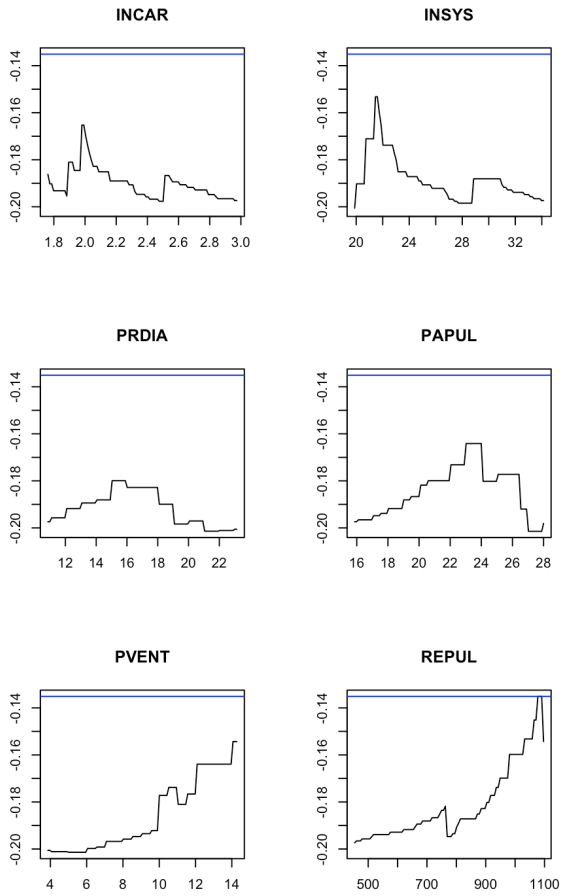
```
mat_gini = mat_v=matrix(NA,7,101)
for(v in 1:7){
  variable=myocarde[,v]
  v_seuil=seq(quantile(myocarde[,v],
6/length(myocarde[,v])),
quantile(myocarde[,v],1-6/length(
myocarde[,v])),length=101)
  mat_v[v,]=v_seuil
  for(i in 1:101){
CLASSE=variable<=v_seuil[i]
mat_gini[v,i]=
gini(y=myocarde$PRONO,classe=CLASSE)}}
-(gini(y=myocarde$PRONO,classe=(myocarde
    [,3]<19))-
gini(y=myocarde$PRONO,classe=(myocarde[,3]<
    Inf)))/
gini(y=myocarde$PRONO,classe=(myocarde[,3]<
    Inf))
[1] 0.5882131
```



```r
idx = which(myocarde$INSYS<19)
mat_gini = mat_v = matrix(NA,7,101)
for(v in 1:7){
  variable = myocarde[idx,v]
  v_seuil = seq(quantile(myocarde[idx,v],
7/length(myocarde[idx,v])),
quantile(myocarde[idx,v],1-7/length(
myocarde[idx,v])), length=101)
  mat_v[v,] = v_seuil
  for(i in 1:101){
    CLASSE = variable<=v_seuil[i]
    mat_gini[v,i]=
      gini(y=myocarde$PRONO[idx],classe=
    CLASSE)}}
par(mfrow=c(3,2))
for(v in 2:7){
plot(mat_v[v,],mat_gini[v,])
}
```



```r
idx = which(myocarde$INSYS>=19)
mat_gini = mat_v = matrix(NA,7,101)
for(v in 1:7){
  variable=myocarde[idx,v]
  v_seuil=seq(quantile(myocarde[idx,v],
6/length(myocarde[idx,v])),
quantile(myocarde[idx,v],1-6/length(
myocarde[idx,v])), length=101)
  mat_v[v,]=v_seuil
  for(i in 1:101){
    CLASSE=variable<=v_seuil[i]
    mat_gini[v,i]=
      gini(y=myocarde$PRONO[idx],
           classe=CLASSE)}}
par(mfrow=c(3,2))
for(v in 2:7){
 plot(mat_v[v,],mat_gini[v,])
}
```



Classification : Variable Importance, on Trees

```
cart = rpart(PRONO~., myocarde)
(split = summary(cart)$splits)
```

```
       count ncat     improve     index        adj
INSYS     71   -1 0.58621312    18.850 0.0000000
REPUL     71    1 0.55440034  1094.500 0.0000000
INCAR     71   -1 0.54257020     1.690 0.0000000
PRDIA     71    1 0.27284114    17.000 0.0000000
PAPUL     71    1 0.20466714    23.250 0.0000000
```

```
REPUL     27    1 0.18181818  1585.000 0.0000000
PVENT     27   -1 0.10803571    14.500 0.0000000
PRDIA     27    1 0.10803571    18.500 0.0000000
PAPUL     27    1 0.10803571    22.500 0.0000000
INCAR     27    1 0.04705882     1.195 0.0000000
```

```
cart$variable.importance
```

```
    INSYS     REPUL     INCAR     PAPUL     PRDIA     FRCAR     PVENT
10.364984 10.05108   8.21212   3.24415   2.82761   1.86230   0.33737
```

Classification : Boosting (adabost)

Assume here that $y \in \{-1, +1\}$. The loss function is $e^{-y \cdot m(\boldsymbol{x})}$ ($y \cdot m(\boldsymbol{x})$ was already discussed for the SVM algorithm)

The loss function related to the logistic model would be $\log(1 + e^{-y \cdot m(\boldsymbol{x})})$.

Using residuals is more complicated, so here, we use an alternative strategy : weights.

We start with $\boldsymbol{\omega}_0 = \mathbf{1}/n$, then at each step fit a model (a classification tree) with weights $\boldsymbol{\omega}_k$ (we did not discuss weights in the algorithms of trees, but it is straightforward in the formula actually).

Let $\widehat{h}_{\boldsymbol{\omega}_k}$ denote that model (i.e. the probability in each leaves).

Classification : Boosting (adabost)

Then consider the classifier $2\ \mathbf{1}[\widehat{h}_{\boldsymbol{\omega}_k}(\cdot) > 0.5] - 1$ which returns a value in $\{-1, +1\}$. Then set $\varepsilon_k = \sum_{i \in \mathcal{I}_k} \omega_i$ where $\mathcal{I}_k$ is the set of mis-classified individuals,

$$\mathcal{I}_k = \left\{ i : 2\ \mathbf{1}[\widehat{h}_{\boldsymbol{\omega}_k}(\boldsymbol{x}_i) > 0.5] - 1 \neq y_i \right\}$$

Then set

$$\alpha_k = \frac{1}{2} \ln \left(\frac{1 - \epsilon_k}{\epsilon_k} \right)$$

and update finally the model using

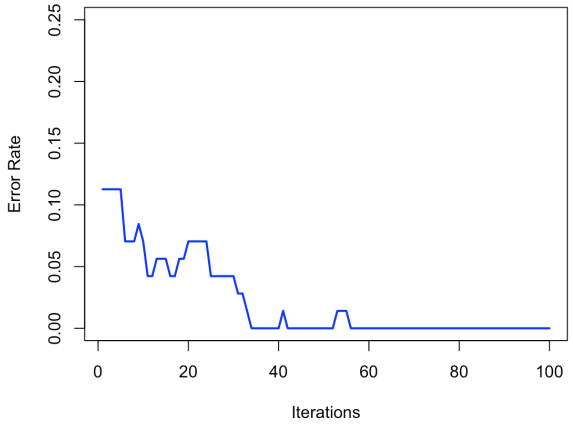
$$m_{k=1} = m_k + \alpha_k \widehat{h}_{\boldsymbol{\omega}_k}$$

as well as the weights

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k e^{-\boldsymbol{y} \alpha_k \widehat{h}_{\boldsymbol{\omega}_k}(\boldsymbol{x}_i)}$$

(of course, divide by the sum to insure that the total sum is then 1).

```r
n_iter = 100
y = (myocarde[,"PRONO"]==1)*2-1
x = myocarde[,1:7]
error = rep(0,n_iter)
f = rep(0,length(y))
w = rep(1,length(y)) #
alpha = 1
library(rpart)
for(i in 1:n_iter){
  w = exp(-alpha*y*f) *w
  w = w/sum(w)
  rfit = rpart(y~., x, w, method="class")
  g = -1 + 2*(predict(rfit,x)[,2]>.5)
  e = sum(w*(y*g<0))
  alpha = .5*log ( (1-e) / e )
  alpha = 0.1*alpha
  f = f + alpha*g
  error[i] = mean(1*f*y<0)
}
plot(seq(1,n_iter),error)
```

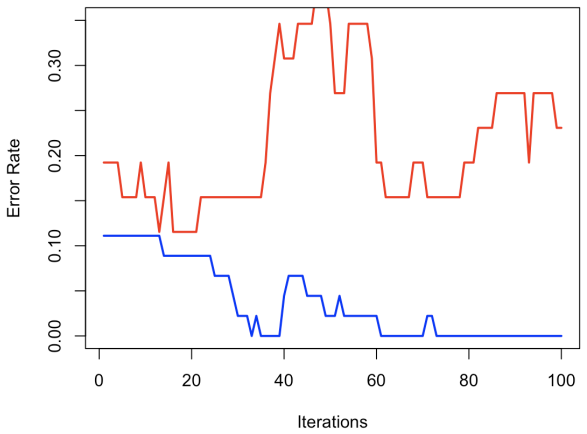


Classification : Boosting (adabost)

It is necessary to use a training / validation dataset to avoid overfit...

```
set.seed(123)
id_train = sample(1:nrow(myocarde), size=45, replace=FALSE)
train_myocarde = myocarde[id_train,]
test_myocarde = myocarde[-id_train,]
```

```r
for(i in 1:n_iter){
  w_train = w_train*exp(-alpha*y_train*f_train)
  w_train = w_train/sum(w_train)
  rfit = rpart(y_train~., x_train, w_train, method="class")
  g_train = -1 + 2*(predict(rfit,x_train)[,2]>.5)
  g_test = -1 + 2*(predict(rfit,x_test)[,2]>.5)
  e_train = sum(w_train*(y_train*g_train<0))
  alpha = .5*log ( (1-e_train) / e_train )
  alpha = 0.1*alpha
  f_train = f_train + alpha*g_train
  f_test = f_test + alpha*g_test
  train_error[i] = mean(1*f_train*y_train<0)
  test_error[i] = mean(1*f_test*y_test<0)}
```



Classification : Boosting (adabost)

```
library(gbm)
gbmWithCrossValidation = gbm(PRONO ~ .,
    distribution = "bernoulli",
data = myocarde,n.trees = 2000,shrinkage =
    .01,cv.folds = 5,n.cores = 1)
bestTreeForPrediction = gbm.perf(
    gbmWithCrossValidation)
```