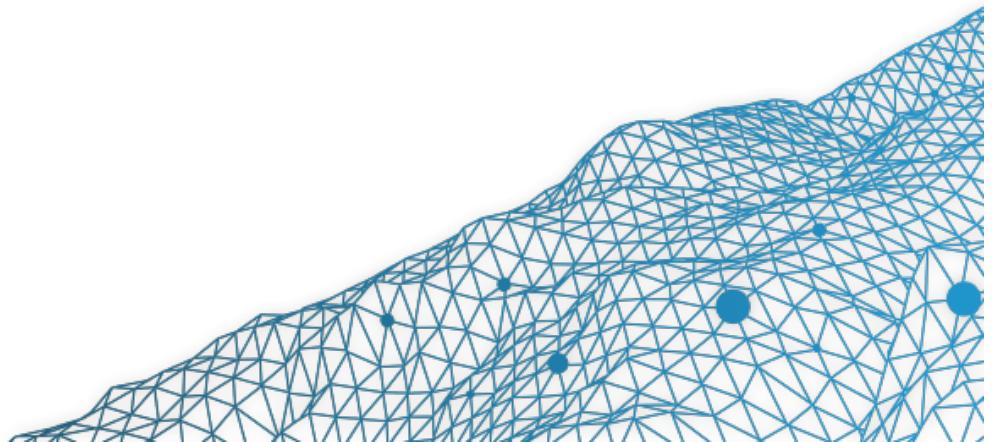


Wrap-Up - Classification # 13

Arthur Charpentier (Université du Québec à Montréal, Canada)

Machine Learning & Econometrics

SIDE Summer School, July 2019



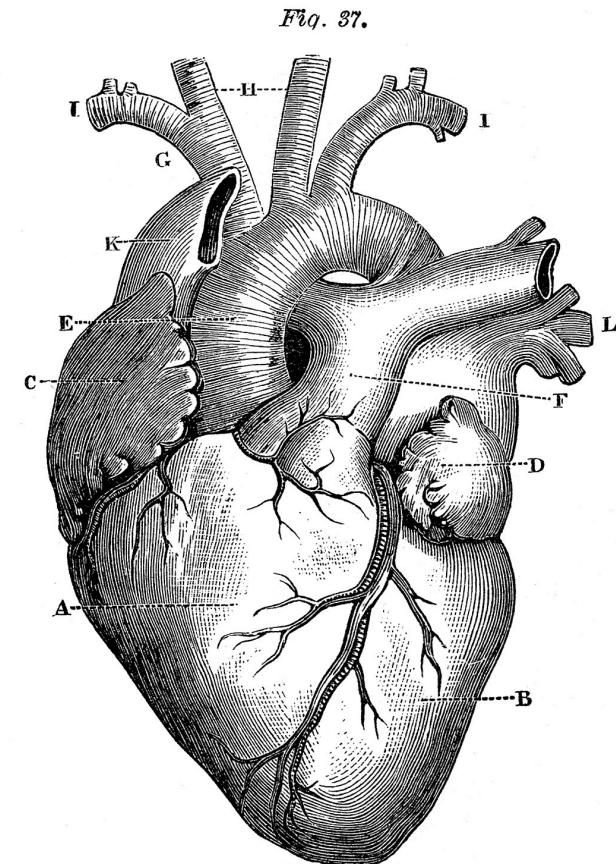
Modeling a 0/1 random variable

Myocardial infarction of patients admitted in E.R.

- heart rate (FRCAR) : x_1
- heart index (INCAR) : x_2
- stroke index (INSYS) : x_3
- diastolic pressure (PRDIA) : x_4
- pulmonary arterial pressure (PAPUL) : x_5
- ventricular pressure (PVENT) : x_6
- lung resistance (REPUL) : x_7
- death or survival (PRONO) : y

dataset $\{(x_i, y_i)\}$ where $i = 1, \dots, n$, $n = 73$.

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



Modeling a 0/1 random variable

```
1 > myocarde = read.table("http://freakonometrics.free.fr/myocarde.csv"  
2   ,head=TRUE , sep=";")  
3  
4 > head(myocarde)  
5  
6   FRCAR INCAR INSYS PRDIA PAPUL PVENT REPUL PRONO  
7 1    90  1.71  19.0     16  19.5  16.0    912 SURVIE  
8 2    90  1.68  18.7     24  31.0  14.0   1476 DECES  
9 3   120  1.40  11.7     23  29.0   8.0   1657 DECES  
10 4    82  1.79  21.8     14  17.5  10.0    782 SURVIE  
11 5    80  1.58  19.7     21  28.0  18.5   1418 DECES  
12 6    80  1.13  14.1     18  23.5   9.0   1664 DECES  
13 > myocarde$PRONO = (myocarde$PRONO=="SURVIE")*1
```

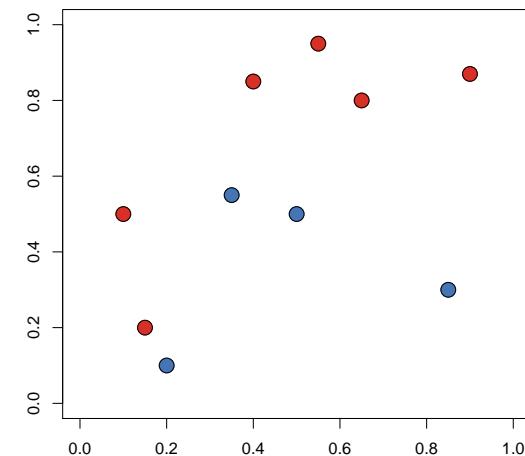
Classification : Logistic Regression (and GLM)

Simulated data, $y \in \{0, 1\}$

$$\mathbb{P}(Y = 1 | \mathbf{X} = \mathbf{x}) = \frac{\exp[\mathbf{x}^\top \boldsymbol{\beta}]}{1 + \exp[\mathbf{x}^\top \boldsymbol{\beta}]}$$

Inference using maximum likelihood techniques

$$\hat{\boldsymbol{\beta}} = \operatorname{argmin} \left\{ \sum_{i=1}^n \log[\mathbb{P}(Y = 1 | \mathbf{X} = \mathbf{x}_i)] \right\}$$



and the score model is then

$$s(\mathbf{x}) = \frac{\exp[\mathbf{x}^\top \hat{\boldsymbol{\beta}}]}{1 + \exp[\mathbf{x}^\top \hat{\boldsymbol{\beta}}]}$$

```

1 x1 = c(.4,.55,.65,.9,.1,.35,.5,.15,.2,.85)
2 x2 = c(.85,.95,.8,.87,.5,.55,.5,.2,.1,.3)
3 y   = c(1,1,1,1,1,0,0,1,0,0)
4 df  = data.frame(x1=x1,x2=x2,y=as.factor(y))

```

Classification : Logistic Regression (and GLM)

The logistic regression is based on the assumption that

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

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

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

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

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

Mathematical Statistics Courses in a Nutshell

Consider observations $\{y_1, \dots, y_n\}$ from iid random variables $Y_i \sim F_{\theta}$ (with “density” f_{θ}).

Likelihood is

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{y}) \mapsto \prod_{i=1}^n f_{\theta}(y_i)$$

Maximum likelihood estimate is

$$\hat{\boldsymbol{\theta}}^{\text{mle}} \in \arg \max_{\boldsymbol{\theta} \in \Theta} \{\mathcal{L}(\boldsymbol{\theta}; \mathbf{y})\}$$

or $\hat{\boldsymbol{\theta}}^{\text{mle}} \in \arg \max_{\boldsymbol{\theta} \in \Theta} \{\log \mathcal{L}(\boldsymbol{\theta}; \mathbf{y})\}$

Fisher (1912, On an absolute criterion for fitting frequency curves).

ON AN ABSOLUTE CRITERION FOR FITTING FREQUENCY CURVES.

By R. A. Fisher, Gonville and Caius College, Cambridge.

theoretical curve, so the probability of any particular set of θ 's is proportional to P , where

$$\log P = \sum_1^n \log f.$$

The most probable set of values for the θ 's will make P a maximum.

If a continuous curve is observed—e.g., the period during which a barometer is above any level during the year is a continuous function from which may be derived the relative frequency with which it stands at any height—we should use the expression

$$\log P = \int_{-\infty}^{\infty} y \log f dx.$$

Mathematical Statistics Courses in a Nutshell

Under standard assumptions (Identification of the model, Compactness, Continuity and Dominance), the maximum likelihood estimator is **consistent** $\widehat{\boldsymbol{\theta}}^{\text{mle}} \xrightarrow{\mathbb{P}} \boldsymbol{\theta}$. With additional assumptions, it can be shown that the maximum likelihood estimator **converges to a normal distribution**

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

where I is Fisher information matrix (i.e. $\widehat{\boldsymbol{\theta}}^{\text{mle}}$ is **asymptotically efficient**).

Eg. if $\mathbf{Y} \sim \mathcal{N}(\boldsymbol{\theta}, \sigma^2)$, $\log \mathcal{L} \propto -\sum_{i=1}^n \left(\frac{y_i - \theta}{\sigma} \right)^2$, and $\widehat{\theta}^{\text{mle}} = \bar{y}$ (see also method of moments).

$$\max \{ \log \mathcal{L} \} \iff \min \left\{ \sum_{i=1}^n \varepsilon_i^2 \right\} = \text{least squares}$$

Classification : Logistic Regression

- Numerical Optimization

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. (if $x \rightarrow x_{\text{old}}$ and $x_0 \rightarrow x_{\text{new}}$)

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

or, in higher dimension

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_{\text{old}} - \nabla g(\boldsymbol{x}_{\text{old}})^{-1} \cdot g(\boldsymbol{x}_{\text{old}}) \text{ from some starting point}$$

Classification : Logistic Regression

$x_{\text{new}} = x_{\text{old}} - \nabla g(x_{\text{old}})^{-1} \cdot g(x_{\text{old}})$ from some starting point

Here g is the gradient of the log-likelihood,

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

Classification : Logistic Regression

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

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

```

1 y = myocarde$PRONO
2 X = cbind(1, as.matrix(myocarde[, 1:7]))
3 negLogLik = function(beta){
4   -sum(-y*log(1 + exp(-(X%*%beta))) - (1-y)*log(1 + exp(X%*%beta)))
5 }
6 beta_init = lm(PRONO ~ ., data=myocarde)$coefficients
7 logistic_opt = optim(par = beta_init, negLogLik, hessian=TRUE, method
= "BFGS", control=list(abstol=1e-9))

```

Classification : Logistic Regression

Here, we obtain

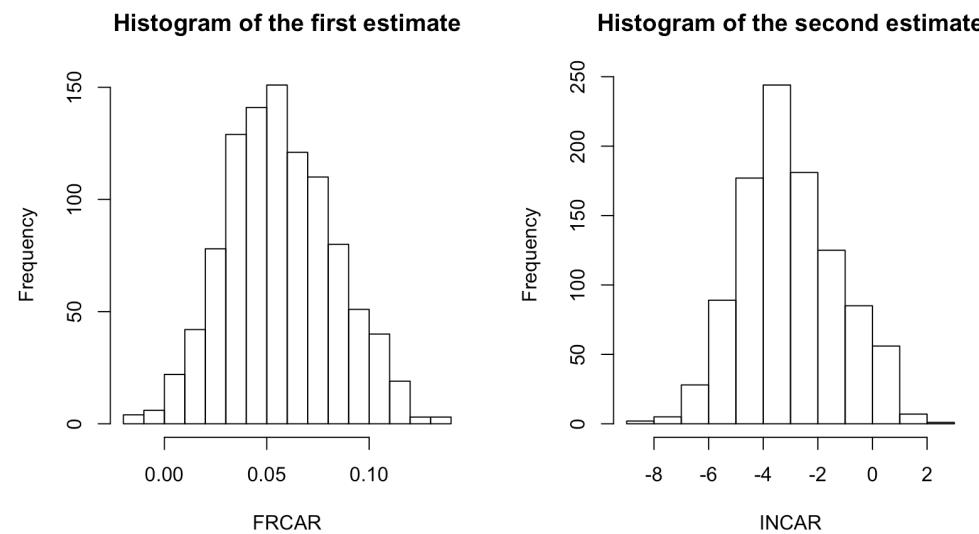
```

1 logistic_opt$par
2 (Intercept)          FRCAR           INCAR           INSYS
3 1.656926397   0.045234029 -2.119441743   0.204023835
4 PRDIA            PAPUL           PVENT           REPUL
5 -0.102420095   0.165823647 -0.081047525 -0.005992238
6 simu = function(i){
7 logistic_opt_i = optim(par = rnorm(8,0,3)*beta_init,
8 negLogLik, hessian=TRUE, method = "BFGS",
9 control=list(abstol=1e-9))
10 logistic_opt_i$par[2:3]
11 }
12 v_beta = t(Vectorize(simu)(1:1000))

```

Classification : Logistic Regression

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



- Fisher Algorithm

We want to use Newton's algorithm

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

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

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

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

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

```

1 Y=myocarde$PRONO
2 X=cbind(1, as.matrix(myocarde[, 1:7]))
3 colnames(X)=c("Inter", names(myocarde[, 1:7]))
4 beta=as.matrix(lm(Y~0+X)$coefficients, ncol=1)
5 for(s in 1:9){

```

```

6   pi=exp(X%*%beta[,s])/(1+exp(X%*%beta[,s]))
7   gradient=t(X)%*%(Y-pi)
8   omega=matrix(0,nrow(X),nrow(X));diag(omega)=(pi*(1-pi))
9   Hessian=-t(X)%*%omega%*%X
10  beta=cbind(beta,beta[,s]-solve(Hessian)%*%gradient)}
11 beta[,8:10]
12
13 XInter -10.187641685 -10.187641696 -10.187641696
14 XFRCAR  0.138178119  0.138178119  0.138178119
15 XINCAR -5.862429035 -5.862429037 -5.862429037
16 XINSYS  0.717084018  0.717084018  0.717084018
17 XPRDIA -0.073668171 -0.073668171 -0.073668171
18 XPAPUL  0.016756506  0.016756506  0.016756506
19 XPVENT -0.106776012 -0.106776012 -0.106776012
20 XREPUL -0.003154187 -0.003154187 -0.003154187

```

- iteratively reweighted least squares

We want to compute something like

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

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

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

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

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

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

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.

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

```

3 X = cbind(1,as.matrix(myocarde[,1:7]))
4 beta = beta_init
5 for(s in 1:1000){
6 p = exp(X %*% beta) / (1+exp(X %*% beta))
7 omega = diag(nrow(df))
8 diag(omega) = (p*(1-p))
9 df$Z = X %*% beta + solve(omega) %*% (df$PRONO - p)
10 beta = lm(Z~.,data=df[,-8], weights=diag(omega))$coefficients
11 }
12 beta
13 (Intercept)          FRCAR          INCAR          INSYS          PRDIA
14 -10.187641696    0.138178119   -5.862429037    0.717084018   -0.073668171
15 PAPUL              PVENT          REPUL
16 0.016756506     -0.106776012   -0.003154187

```

Classification : Logistic Regression

```
1 summary( lm(Z~.,data=df[,-8], weights=diag(omega)))  
2  
3 Coefficients:  
4  
5 (Intercept) -10.187642   10.668138  -0.955      0.343  
6 FRCAR        0.138178    0.102340   1.350      0.182  
7 INCAR        -5.862429   6.052560  -0.969      0.336  
8 INSYS         0.717084    0.503527   1.424      0.159  
9 PRDIA        -0.073668   0.261549  -0.282      0.779  
10 PAPUL        0.016757    0.306666   0.055      0.957  
11 PVENT        -0.106776   0.099145  -1.077      0.286  
12 REPUL        -0.003154   0.004386  -0.719      0.475
```

Classification : Logistic Regression

```

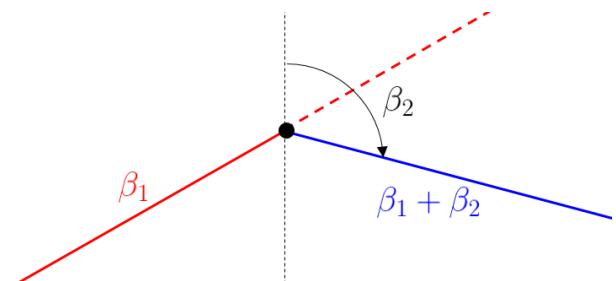
1 summary(glm(PRONO~.,data=myocarde,family=binomial(link = "logit")))
2
3 Coefficients:
4
5             Estimate Std. Error z value Pr(>|z|)
6 (Intercept) -10.187642  11.895227 -0.856   0.392
7 FRCAR        0.138178  0.114112  1.211   0.226
8 INCAR       -5.862429  6.748785 -0.869   0.385
9 INSYS        0.717084  0.561445  1.277   0.202
10 PRDIA       -0.073668  0.291636 -0.253   0.801
11 PAPUL        0.016757  0.341942  0.049   0.961
12 PVENT       -0.106776  0.110550 -0.966   0.334
13 REPUL      -0.003154  0.004891 -0.645   0.519

```

Classification : Logistic Regression with Splines (and GAM)

To get home-made linear splines, use

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



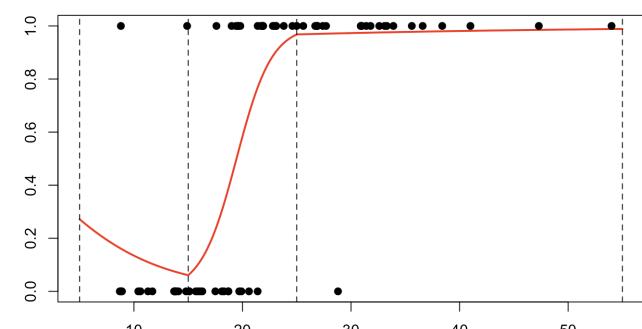
```

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

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

1 summary(reg)
2
3 Coefficients:
4
5             Estimate Std. Error z value Pr(>|z|)
6 (Intercept) -0.1109    3.278   -0.034   0.973
7 INSYS       -0.1751    0.252   -0.693   0.488
8 pos(INS,15)  0.7900    0.374    2.109   0.034 *
9 pos(INS,25) -0.5797    0.290   -1.997   0.045 *

```



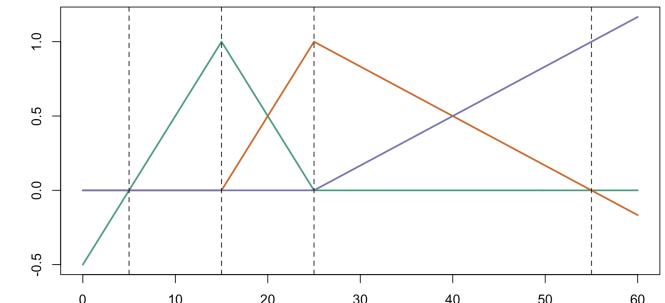
Classification : Logistic Regression with Splines

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

```

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

```

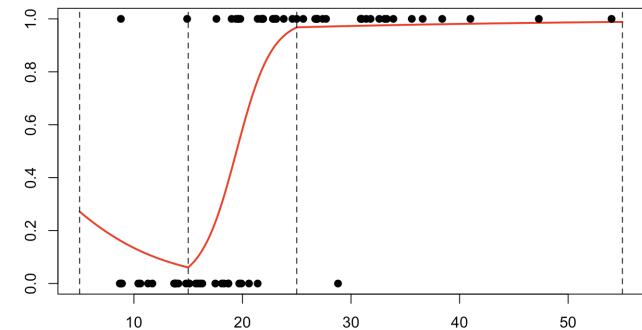


Classification : Logistic Regression with Splines

```

1 reg = glm(PRONO ~ bs(INSYS, knots=c(15,25),
2 Boundary.knots=c(5,55), degree=1),
3 data=myocarde, family=binomial)
4 summary(reg)
5
6 Coefficients:
7             Estimate Std. Error z value Pr(>|z|)
8 (Intercept) -0.9863    2.055   -0.480  0.631
9 bs(INSYS)1   -1.7507    2.526   -0.693  0.488
10 bs(INSYS)2   4.3989    2.061    2.133  0.032 *
11 bs(INSYS)3   5.4572    5.414    1.008  0.313

```



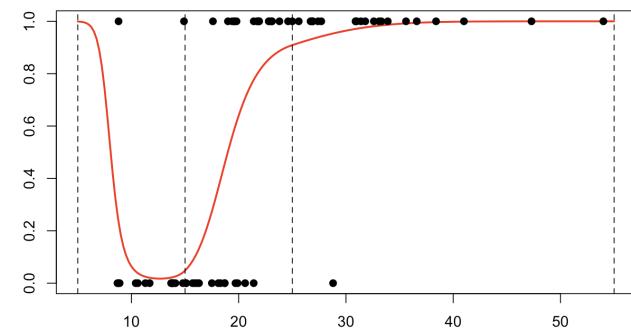
Classification : Logistic Regression with Splines

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

```

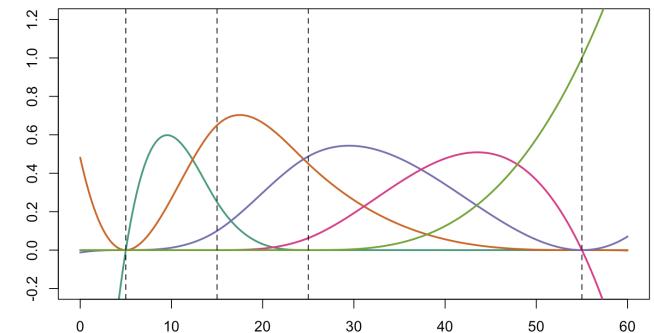
1 pos2 = function(x,s) (x-s)^2*(x>=s)
2 reg = glm(PRONO~poly(INSYS,2)+pos2(INSYS,15) +
            pos2(INSYS,25),
3 data=myocarde,family=binomial)
4 summary(reg)
5
6 Coefficients:
7             Estimate Std. Error z value Pr(>|z|)
8 (Intercept) 29.9842   15.236   1.968  0.049 *
9 poly(2)1    408.7851  202.419   2.019  0.043 *
10 poly(2)2   199.1628  101.589   1.960  0.049 *
11 pos2(15)   -0.2281    0.126  -1.805  0.071 .
12 pos2(25)    0.0439    0.080   0.545  0.585

```



Classification : Logistic Regression with Splines

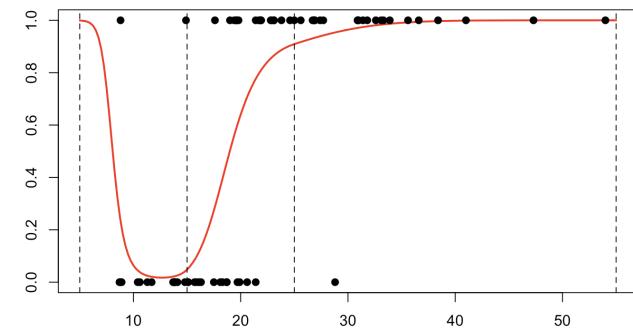
```
1 x = seq(0,60,by=.25)
2 B=bs(x,knots=c(15,25),Boundary.knots=c(5,55),
      degree=2)
3 matplot(x,B,type="l",xlab="INSYS",col=clr6)
```



```

1 reg = glm(PRONO~bs(INSYS,knots=c(15,25),
2 Boundary.knots=c(5,55),degree=2),data=myocarde,
3 family=binomial)
4 summary(reg)
5
6 Coefficients:
7
8 Estimate Std. Error z value Pr(>|z|)
9 (Intercept) 7.186 5.261 1.366 0.172
10 bs(INSYS)1 -14.656 7.923 -1.850 0.064 .
11 bs(INSYS)2 -5.692 4.638 -1.227 0.219
12 bs(INSYS)3 -2.454 8.780 -0.279 0.779
13 bs(INSYS)4 6.429 41.675 0.154 0.877

```

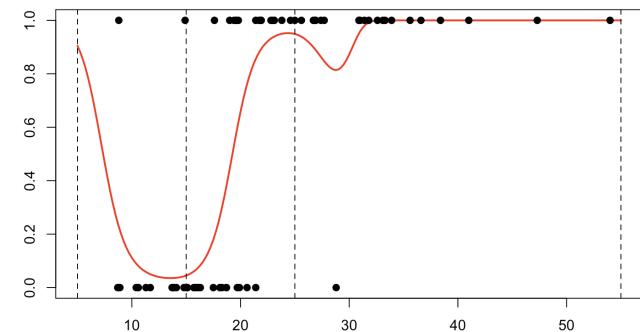


Finally, for cubic splines, use x , x^2 , x^3 , $(x - s_1)^3_+$, $(x - s_2)^3_+$, etc.

```

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

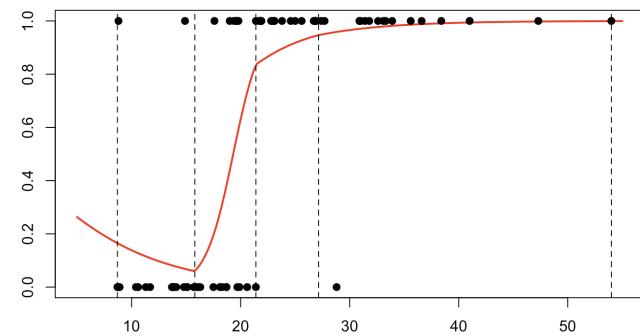
```



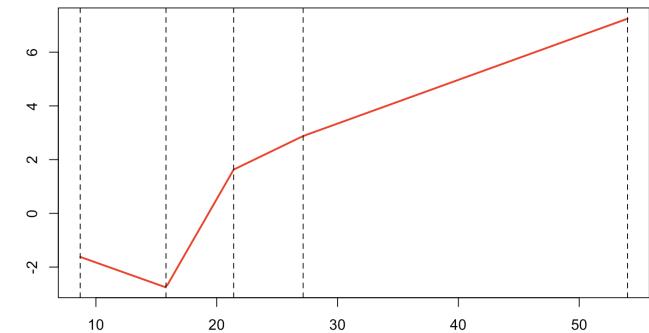
```

1 reg = glm(PRONO~1+bs(INSYS,degree=1,df=4),data
            =myocarde,family=binomial)
2 attr(reg$terms, "predvars")[[3]]
3 bs(INSYS, degree = 1L, knots = c(15.8, 21.4,
        27.15),
4 Boundary.knots = c(8.7, 54), intercept = FALSE
        )
5 quantile(myocarde$INSYS,(0:4)/4)
6      0%    25%    50%    75%   100%
7  8.70 15.80 21.40 27.15 54.00

```



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

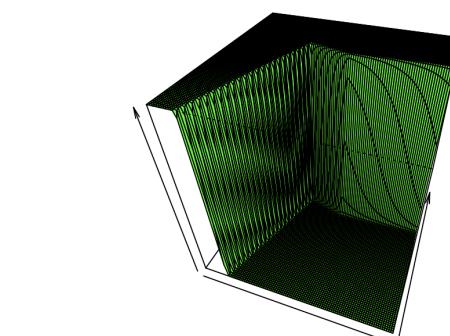
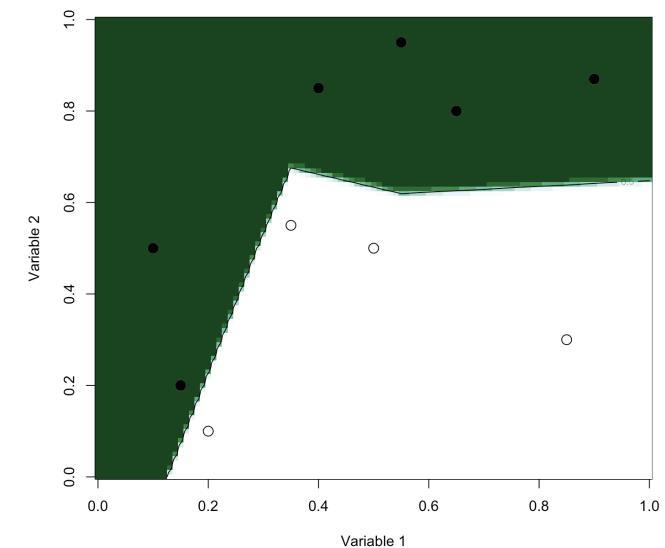


Classification : Logistic Regression with Splines

```

1 reg = glm(y ~ bs(x1, degree=1, df=3) + bs(x2, degree
2   = 1, df=3), data=df, family=binomial(link = "logit"))
3 u = seq(0, 1, length=101)
4 p = function(x, y) predict.glm(reg, newdata=data
5   .frame(x1=x, x2=y), type="response")
6 v = outer(u, u, p)
7 image(u, u, v, xlab="Variable 1", ylab="Variable 2",
8   col=clr10, breaks=(0:10)/10)
9 points(df$x1, df$x2, pch=19, cex=1.5, col="white")
10 points(df$x1, df$x2, pch=c(1, 19)[1+(df$y=="1")],
11   cex=1.5)
12 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

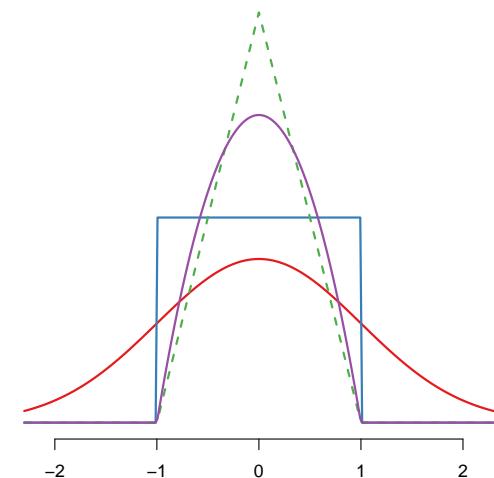
$$\hat{m}(x) = \frac{\sum_{i=1}^n y_i \cdot \mathbf{1}(x_i \in [x \pm h])}{\sum_{i=1}^n \mathbf{1}(x_i \in [x \pm h])}$$

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 ,

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

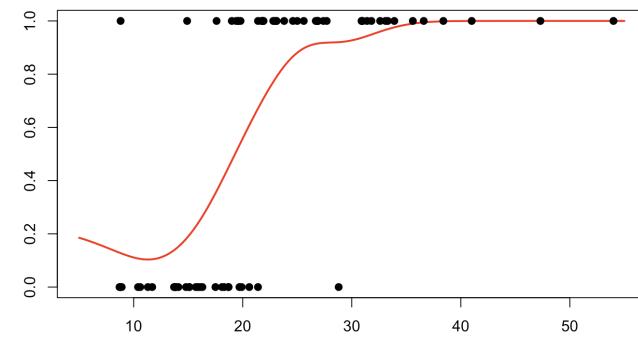


Classification : Regression with Kernels and k -NN

```

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

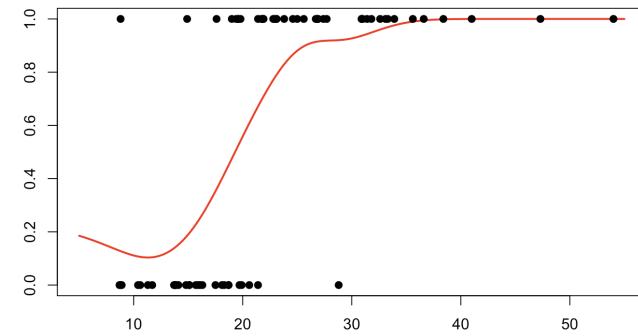
```



```

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

```



Classification : Regression with Kernels and k -NN

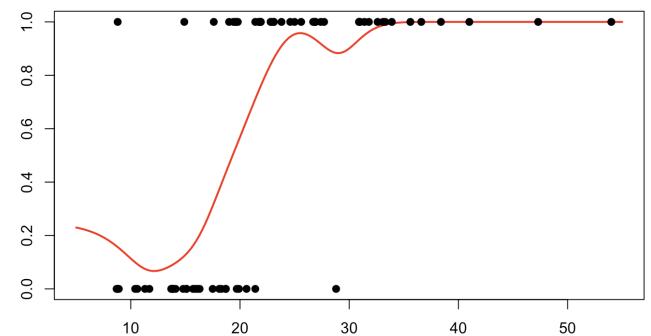
One can also use the `ksmooth()` function of R,

```

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

```

Classical bias / variance tradeoff (below $\hat{m}_h(15)$ and $\hat{m}_h(25)$)

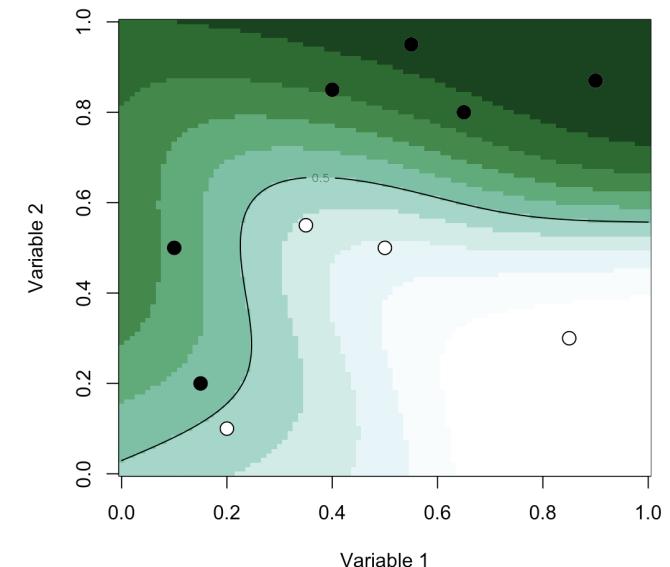


Classification : Regression with Kernels and k -NN

```

1 u = seq(0,1,length=101)
2 p = function(x,y){
3   bw1 = .2; bw2 = .2
4   w = dnorm((df$x1-x)/bw1, mean=0, sd=1) *
5     dnorm((df$x2-y)/bw2, mean=0, sd=1)
6   weighted.mean(df$y=="1",w)
7 }
8 v = outer(u,u,Vectorize(p))
9 image(u,u,v,col=clr10,breaks=(0:10)/10)
10 points(df$x1,df$x2,pch=19,cex=1.5,col="white")
11 points(df$x1,df$x2,pch=c(1,19)[1+(df$y=="1")],cex=1.5)
12 contour(u,u,v,levels = .5,add=TRUE)

```



Classification : Regression with Kernels and k -NN

One can also use k -nearest neighbors

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

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

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

Classically, use Mahalanobis distance

```

1 Sigma = var(myocarde[,1:7])
2 Sigma_Inv = solve(Sigma)
3 d2_mahalanobis = function(x,y,Sinv){as.numeric(x-y) %*% Sinv %*% t(x-y)}
4 k_closest = function(i,k){
5   vect_dist = function(j) d2_mahalanobis(myocarde[i,1:7],myocarde[j,
6     ,1:7],Sigma_Inv)
7   vect = Vectorize(vect_dist)((1:nrow(myocarde)))
8   which((rank(vect)))}
```

```

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

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

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

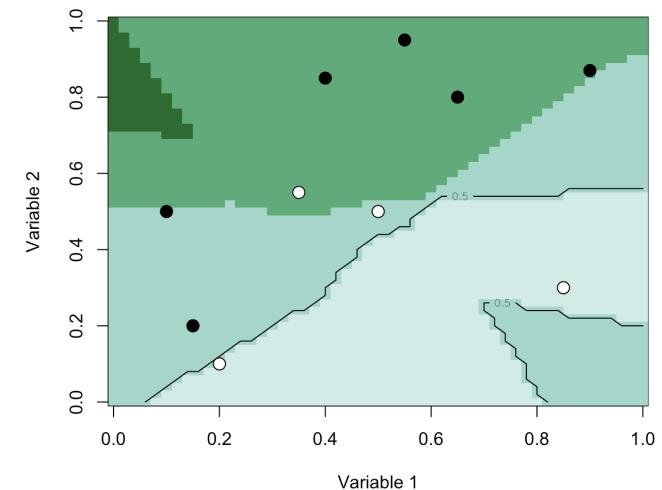
```

```

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

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

```

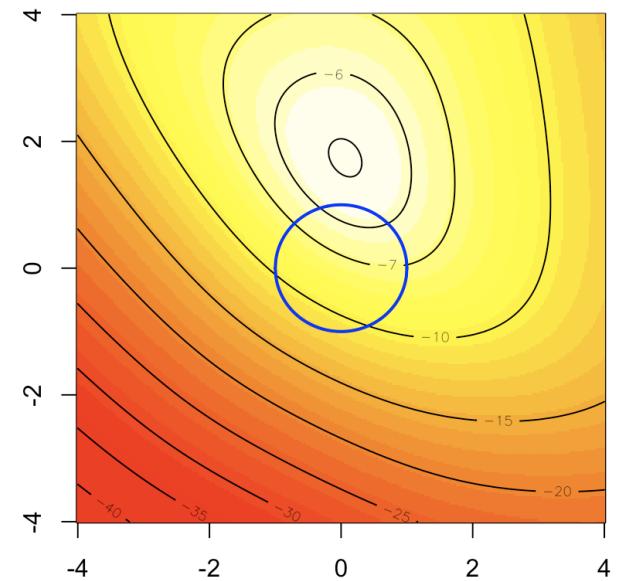


Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

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

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

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



```

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

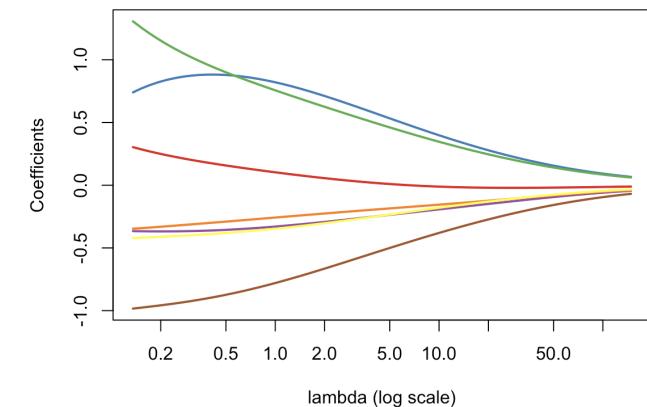
```

Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

```

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

```



Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

- Using Fisher's (penalized) score

On the penalized problem, we can easily prove that

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

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

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

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

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

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

Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

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

```

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

```

16 XInter 0.59619654 0.59619654 0.59619654
17 XFRCAR 0.09217848 0.09217848 0.09217848
18 XINCAR 0.77165707 0.77165707 0.77165707
19 XINSYS 0.69678521 0.69678521 0.69678521
20 XPRDIA -0.29575642 -0.29575642 -0.29575642
21 XPAPUL -0.23921101 -0.23921101 -0.23921101
22 XPVENT -0.33120792 -0.33120792 -0.33120792
23 XREPUL -0.84308972 -0.84308972 -0.84308972

```

Use

$$\text{Var}[\hat{\beta}_\lambda] = [\mathbf{X}^\top \Delta \mathbf{X} + 2\lambda \mathbb{I}]^{-1} \mathbf{X}^\top \Delta \text{Var}[\mathbf{z}] \Delta \mathbf{X} [\mathbf{X}^\top \Delta \mathbf{X} + 2\lambda \mathbb{I}]^{-1}$$

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

```

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

```

```

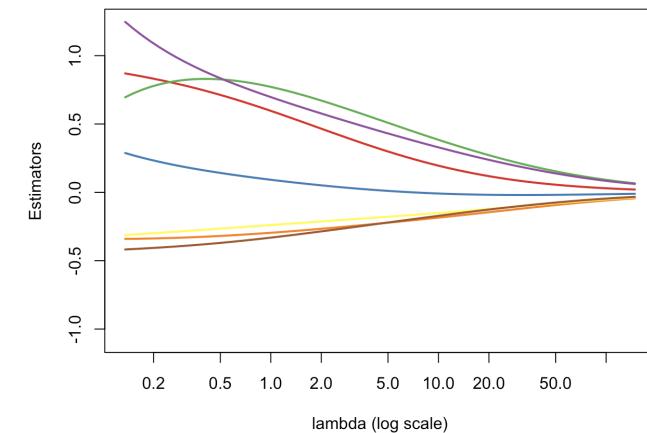
7   B = solve(t(X) *% Delta *% X + 2 * lambda * diag(ncol(X))) *% (t(X) *%
8     Delta *% z)
9   beta = cbind(beta, B)}
10 Varz = solve(Delta)
11 Varb = solve(t(X) *% Delta *% X + 2 * lambda * diag(ncol(X))) *% t(X) *%
12   Delta *% Varz *%
13   Delta *% X *% solve(t(X) *% Delta *% X + 2 * lambda * diag(ncol(X)))
14 return(list(beta=beta[,ncol(beta)], sd=sqrt(diag(Varb))))}

```

```

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

```



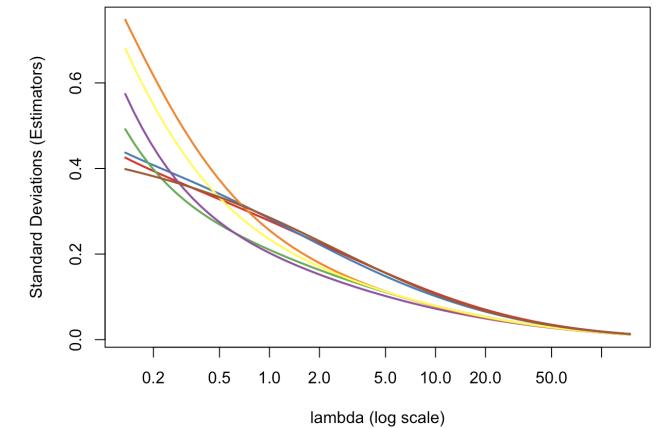
Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

And for the variance

```

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

```

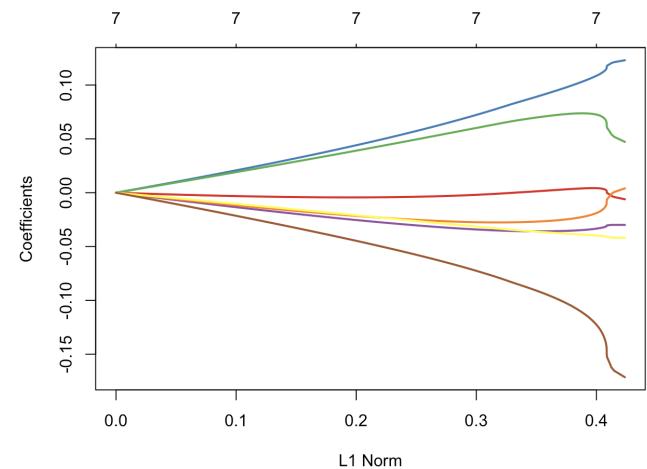
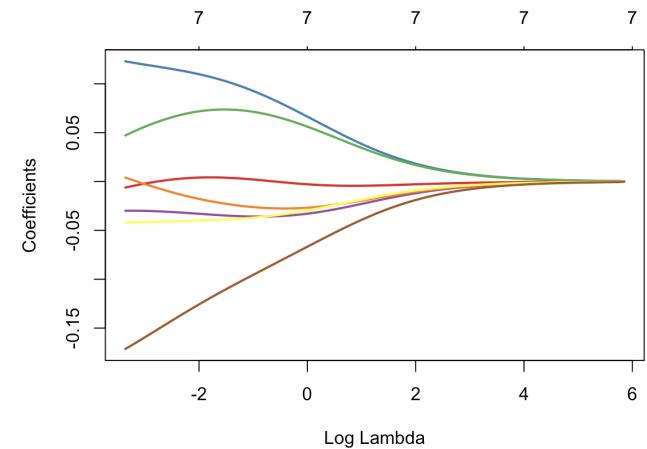


Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

```

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

```



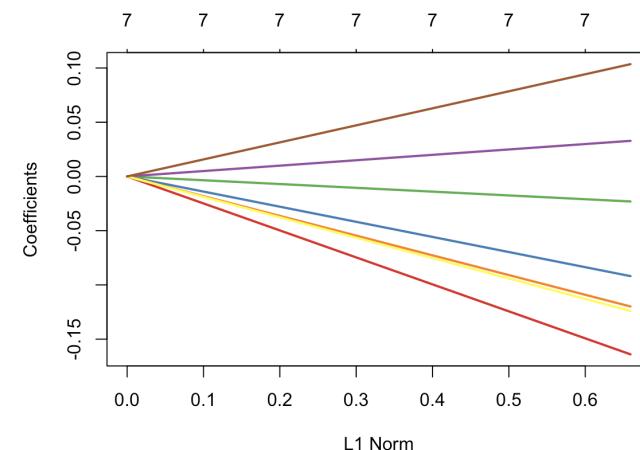
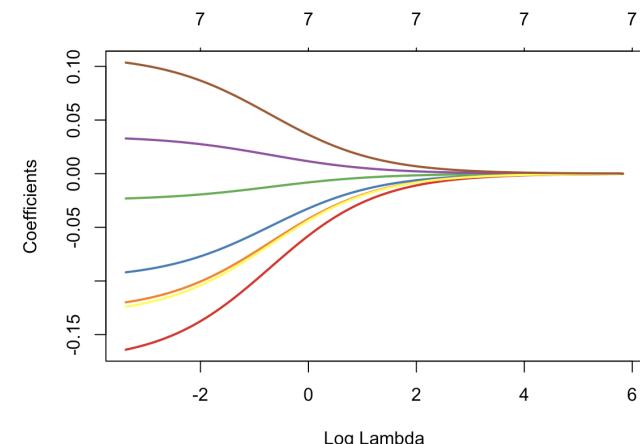
Classification : Penalized Logistic Regression, ℓ_2 norm (Ridge)

- 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)

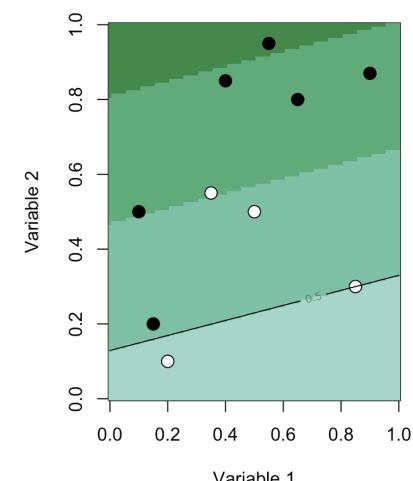
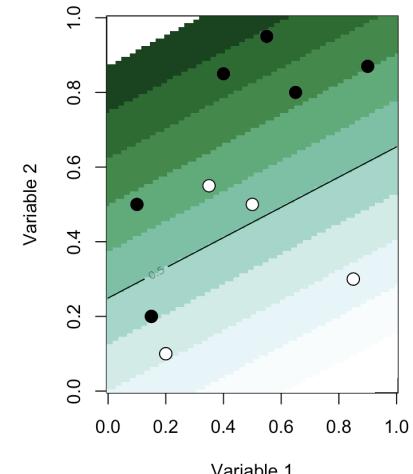
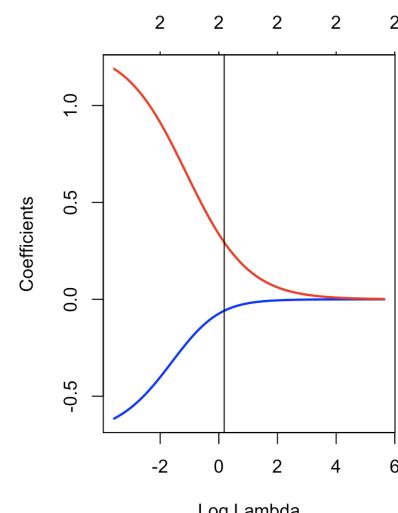
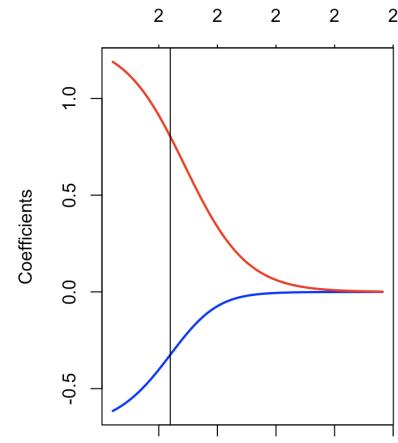
```



```

1 df0 = df
2 df0$y=as.numeric(df$y)-1
3 plot_lambda = function(lambda){
4 m = apply(df0,2,mean)
5 s = apply(df0,2,sd)
6 for(j in 1:2) df0[,j] = (df0[,j]-m[
7 j])/s[j]
7 reg = glmnet(cbind(df0$x1,df0$x2),
8 df0$y==1, alpha=0)
8 par(mfrow=c(1,2))
9 plot(reg,xvar="lambda",col=c("blue",
10 "red"),lwd=2)
11 abline(v=log(.2))
11 plot_lambda(.2)
12 plot(reg,xvar="lambda",col=c("blue",
13 "red"),lwd=2)
13 abline(v=log(1.2))
14 plot_lambda(1.2)

```

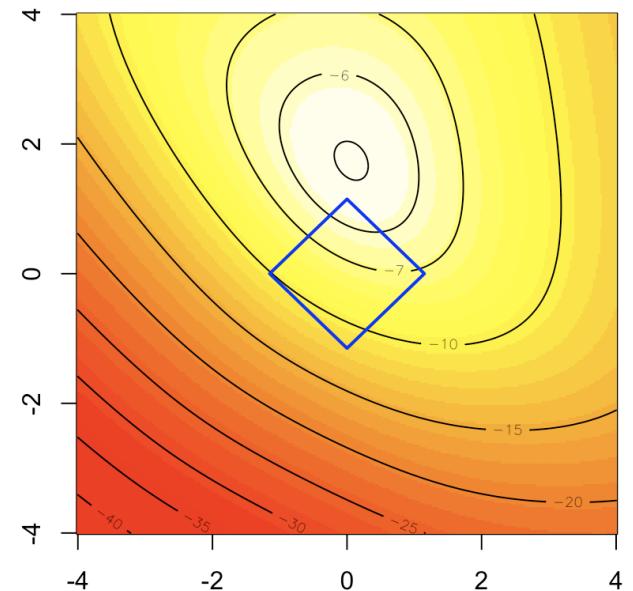


Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

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

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

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

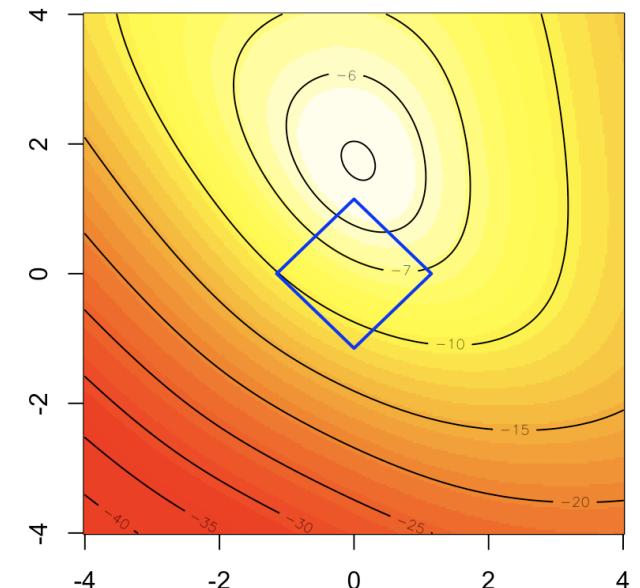


Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

1 y = myocarde$PRONO
2 X = myocarde[,1:7]
3 for(j in 1:7) X[,j] = (X[,j]-mean(X[,j]))/sd(X
[,j])
4 X = as.matrix(X)
5 LogLik = function(bbeta){
6   b0=bbeta[1]
7   beta=bbeta[-1]
8   sum(-y*log(1 + exp(-(b0+X%*%beta))) -
9   (1-y)*log(1 + exp(b0+X%*%beta)))}
10 u = seq(-4,4,length=251)
11 v = outer(u,u,function(x,y) LogLik(c(1,x,y)))
12 image(u,u,v,col=rev(heat.colors(25)))
13 contour(u,u,v,add=TRUE)
14 polygon(c(-1,0,1,0),c(0,1,0,-1),border="blue")

```



Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

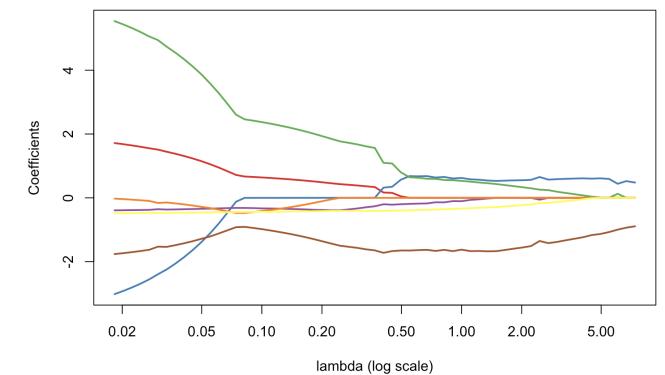
1 PennegLogLik = function(bbeta,lambda=0){
2   b0=bbeta[1]
3   beta=bbeta[-1]
4   -sum(-y*log(1 + exp(-(b0+X%*%beta))) -
5 (1-y)*log(1 + exp(b0+X%*%beta)))+lambda*sum(abs(beta)) }
6 opt_lasso = function(lambda){
7   beta_init = lm(PRONO~.,data=myocarde)$coefficients
8   logistic_opt = optim(par = beta_init*0, function(x) PennegLogLik(x,
9   lambda),
  hessian=TRUE, method = "BFGS", control=list(abstol=1e-9))

```

```

1 logistic_opt$par[-1] }
2 v_lambda=c(exp(seq(-4,2,length=61)))
3 est_lasso=Vectorize(opt_lasso)(v_lambda)
4 plot(v_lambda,est_lasso[1,],col=colrs[1])
5 for(i in 2:7) lines(v_lambda,est_lasso[i,],col
    =colrs[i],lwd=2)

```



Observe that coefficients are not (strictly) null

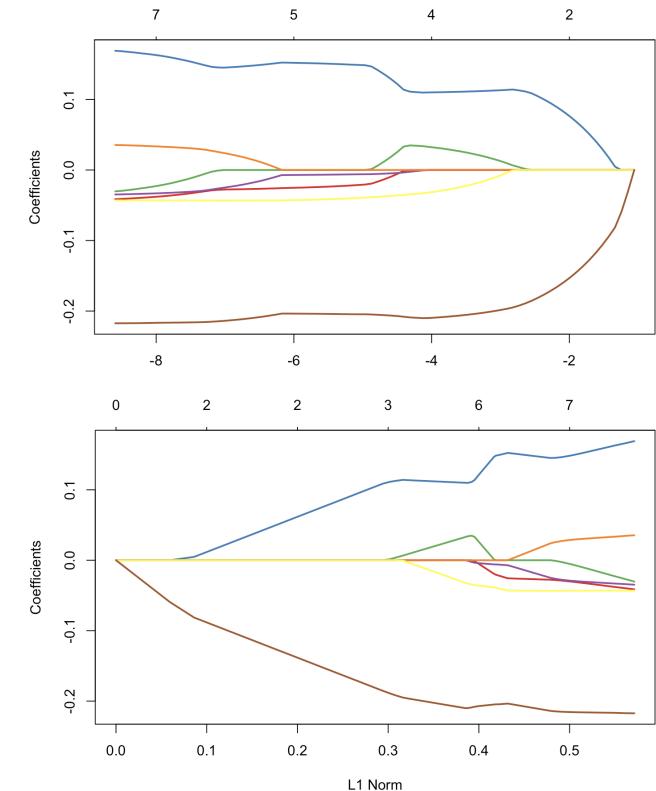
```

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

4 plot(glm_lasso, col=colrs, lwd=2)
5 glmnet(X, y, alpha=1, lambda=exp(-4))$beta
6 7x1 sparse Matrix of class "dgCMatrix"

7 s0
8 FRCAR .
9 INCAR 0.11005070
10 INSYS 0.03231929
11 PRDIA .
12 PAPUL .
13 PVENT -0.03138089
14 REPUL -0.20962611

```



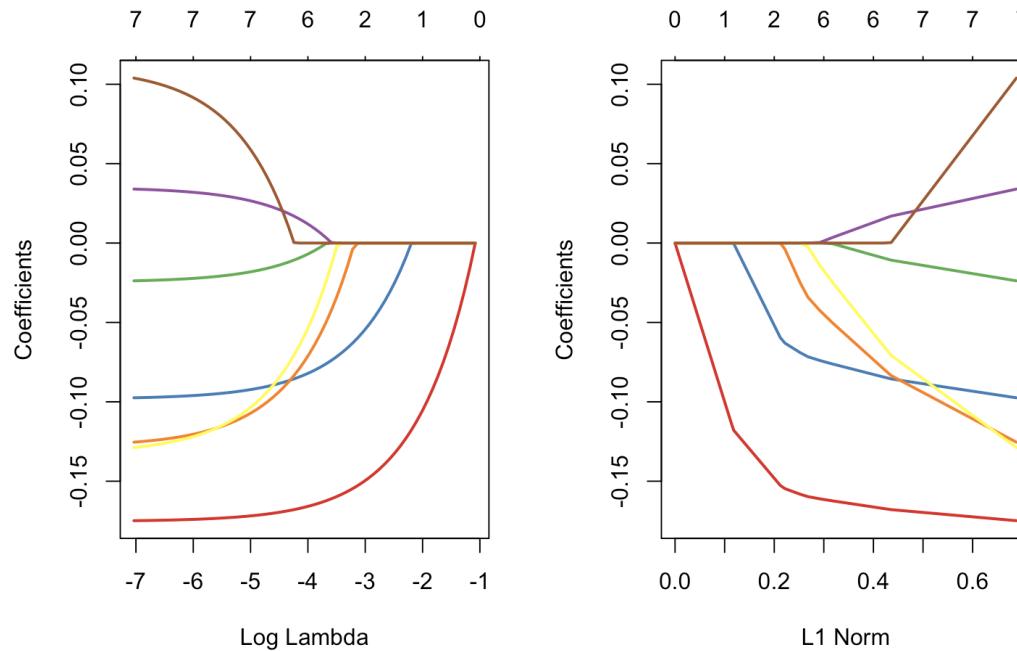
Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

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 OLS-LASSO...

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

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

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

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

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

i.e.

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

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

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

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

For the term on the left, we recognize

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

so that the previous equation can be written

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

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_1 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_1 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

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

can also be written

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

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

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

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

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

the coordinate-wise update becomes

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

An alternative is to set

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

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

so that the optimization problem can be written, equivalently

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

hence

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

and one gets

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

or, if we develop

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

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

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

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

The code to compute this componentwise descent is

```

1 soft_thresholding = function(x,a){
2   result = numeric(length(x))
3   result[which(x > a)] = a
4   result[which(x < -a)] <- x[which(x < -a)] + a
5   return(result)

```

and the code

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

1 lasso_coord_desc = function(X,y,beta,lambda,tol=1e-6,maxiter=1000){
2   beta = as.matrix(beta)
3   X = as.matrix(X)
4   omega = rep(1/length(y),length(y))
5   obj = numeric(length=(maxiter+1))
6   betalist = list(length(maxiter+1))
7   betalist[[1]] = beta
8   beta0list = numeric(length(maxiter+1))
9   beta0 = sum(y-X%*%beta)/(length(y))
10  beta0list[1] = beta0
11  for (j in 1:maxiter){
12    for (k in 1:length(beta)){
13      r = y - X[,-k] %*% beta[-k] - beta0*rep(1,length(y))
14      beta[k] = (1/sum(omega*X[,k]^2))*soft_thresholding(t(omega*r)
15      %*%X[,k],length(y)*lambda)
16    }
17    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),
22          'F') < tol) { break }
23 return(list(obj=obj[1:j], beta=beta, intercept=beta0)) }
```

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

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

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

$$\log \mathcal{L} \approx \log \mathcal{L}' = \frac{1}{n} \sum_{i=1}^n \omega_i \cdot [z_i - (\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})]^2$$

where z_i is the working response

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

p_i is the prediction

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

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

Thus, we obtain a penalized least-square problem.

```

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

```

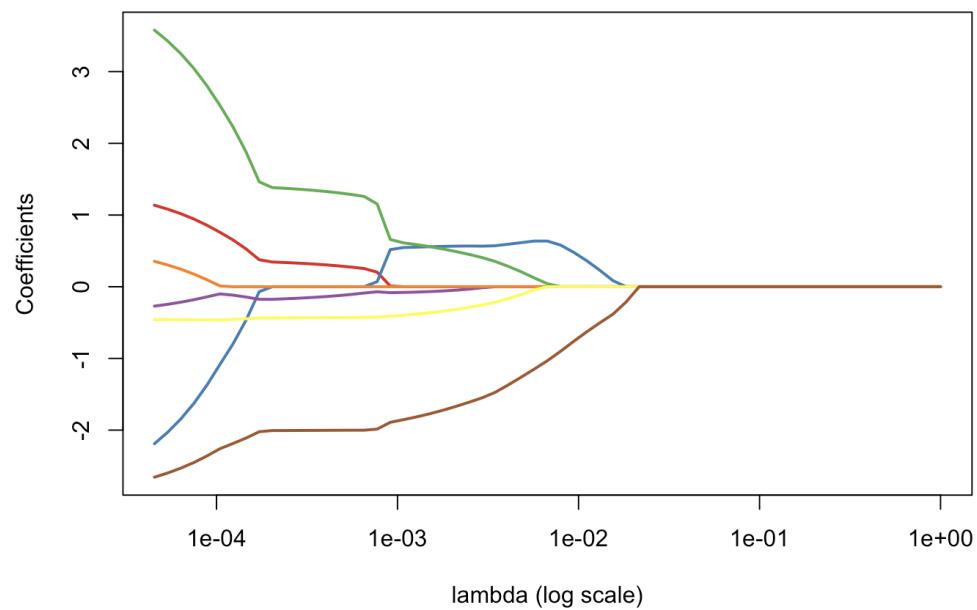
```

9   z = beta0*rep(1,length(y)) + X%*%beta + (y-p)/(p*(1-p))
10  omega = p*(1-p)/(sum((p*(1-p))))
11  beta0list = numeric(length(maxiter+1))
12  beta0 = sum(y-X%*%beta)/(length(y))
13  beta0list[1] = beta0
14  for (j in 1:maxiter){
15    for (k in 1:length(beta)){
16      r = z - X[, -k] %*% beta[-k] - beta0*rep(1,length(y))
17      beta[k] = (1/sum(omega*X[, k]^2))*soft_thresholding(t(omega*r)%
18 *%X[, k], length(y)*lambda)
19    }
20    beta0 = sum(y-X%*%beta)/(length(y))
21    beta0list[j+1] = beta0
22    betalist[[j+1]] = beta
23    obj[j] = (1/2)*(1/length(y))*norm(omega*(z - X%*%beta -
24 beta0*rep(1,length(y))), 'F')^2 + lambda*sum(abs(beta))
25    p = exp(beta0*rep(1,length(y)) + X%*%beta)/(1+exp(beta0*rep(1,
26 length(y)) + X%*%beta))

```

```

25   z = beta0*rep(1,length(y)) + X%*%beta + (y-p)/(p*(1-p))
26   omega = p*(1-p)/(sum((p*(1-p))))
27   if (norm(rbind(beta0list[j],betalist[[j]])) -
28 rbind(beta0,beta),'F') < tol) { break }
29
30 return(list(obj=obj[1:j],beta=beta,intercept=beta0)) }
```



Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

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

Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

4 reg = glmnet(cbind(df0$x1, df0$  

+ x2), df0$y==1, alpha=1)  

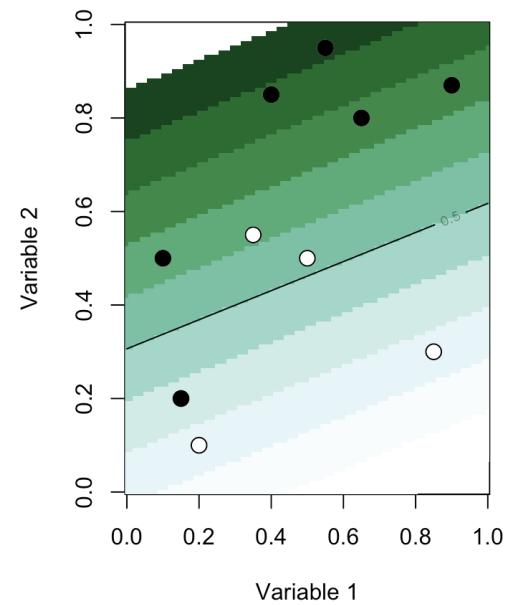
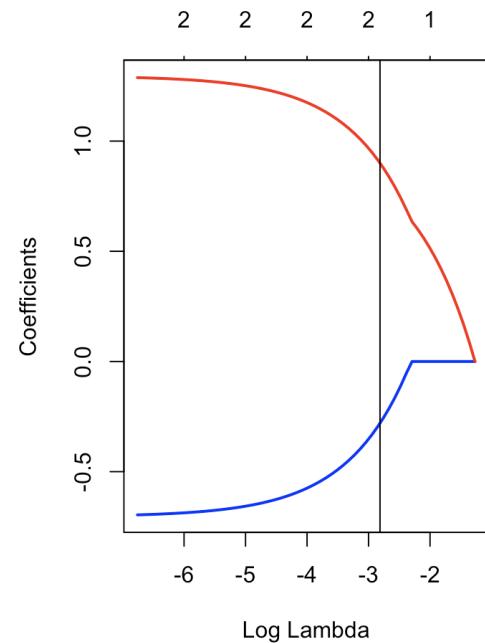
5 par(mfrow=c(1,2))  

6 plot(reg, xvar="lambda", col=c("blue","red"), lwd=2)  

7 abline(v=exp(-2.8))  

8 plot_lambda(exp(-2.8))

```



Classification : Penalized Logistic Regression, ℓ_1 norm (LASSO)

```

4 reg = glmnet(cbind(df0$x1, df0$  

+ x2), df0$y==1, alpha=1)  

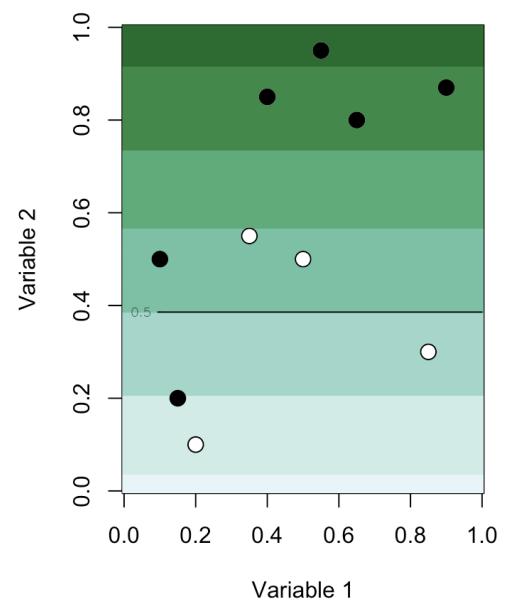
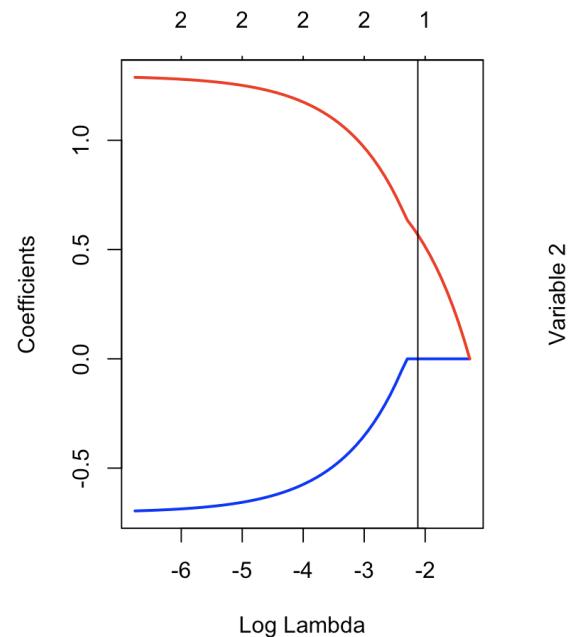
5 par(mfrow=c(1,2))  

6 plot(reg, xvar="lambda", col=c("blue","red"), lwd=2)  

7 abline(v=exp(-2.1))  

8 plot_lambda(exp(-2.1))

```



Classification : (Linear) Fisher's Discrimination

Consider the following naive classification rule

$$m^*(\mathbf{x}) = \operatorname{argmin}_y \{\mathbb{P}[Y = y | \mathbf{X} = \mathbf{x}]\}$$

or

$$m^*(\mathbf{x}) = \operatorname{argmin}_y \left\{ \frac{\mathbb{P}[\mathbf{X} = \mathbf{x} | Y = y]}{\mathbb{P}[\mathbf{X} = \mathbf{x}]} \right\}$$

(where $\mathbb{P}[\mathbf{X} = \mathbf{x}]$ 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

$$m^*(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbb{E}(Y | \mathbf{X} = \mathbf{x}) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

Classification : (Linear) Fisher's Discrimination

the set

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

is called the decision boundary.

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

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

where r_y^2 is the Mahalanobis distance,

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

Classification : (Linear) Fisher's Discrimination

Let δ_y be defined as

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

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,

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

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

Classification : (Linear) Fisher's Discrimination

```

1 m0 = apply(myocarde[myocarde$PRONO=="0",1:7],2,mean)
2 m1 = apply(myocarde[myocarde$PRONO=="1",1:7],2,mean)
3 Sigma = var(myocarde[,1:7])
4 omega = solve(Sigma) *%*(m1-m0)
5 omega
6
7 FRCAR -0.012909708542
8 INCAR 1.088582058796
9 INSYS -0.019390084344
10 PRDIA -0.025817110020
11 PAPUL 0.020441287970
12 PVENT -0.038298291091
13 REPUL -0.001371677757
14 b = (t(m1) *%* solve(Sigma) *%* m1 - t(m0) *%* solve(Sigma) *%* m0) / 2

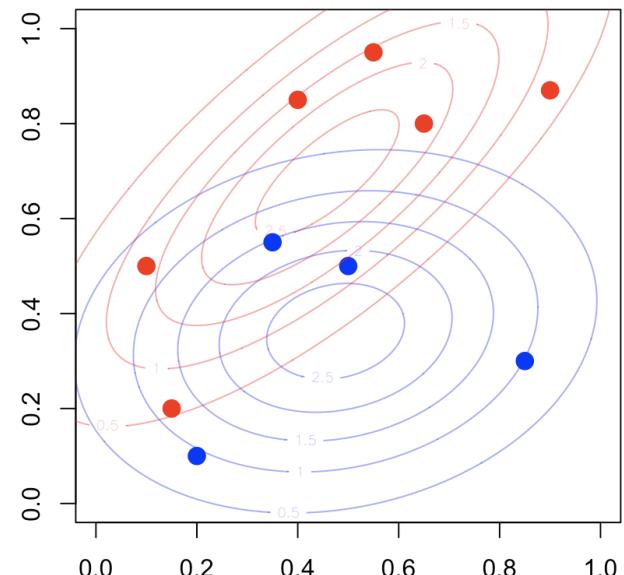
```

Classification : (Linear) Fisher's Discrimination

```

1 x = c(.4,.55,.65,.9,.1,.35,.5,.15,.2,.85)
2 y = c(.85,.95,.8,.87,.5,.55,.5,.2,.1,.3)
3 z = c(1,1,1,1,1,0,0,1,0,0)
4 df = data.frame(x1=x,x2=y,y=as.factor(z))
5 m0 = apply(df[df$y=="0",1:2],2,mean)
6 m1 = apply(df[df$y=="1",1:2],2,mean)
7 Sigma = var(df[,1:2])
8 omega = solve(Sigma)%*%(m1-m0)
9 omega
10 [ ,1]
11 x1 -2.640613174
12 x2 4.858705676

```



```

1 library(MASS)
2 fit_lda = lda(y ~ x1+x2 , data=df)
3 fit_lda
4
5 Coefficients of linear discriminants:
6
7 LD1
8 x1 -2.588389554
x2 4.762614663

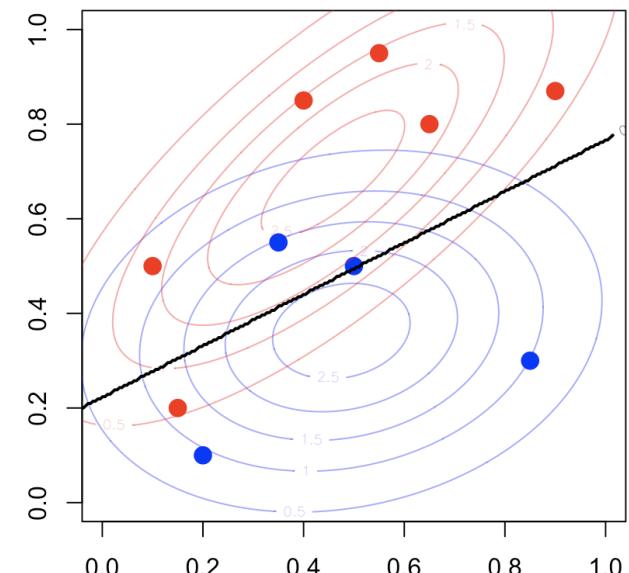
```

and for the intercept

```

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

```



```

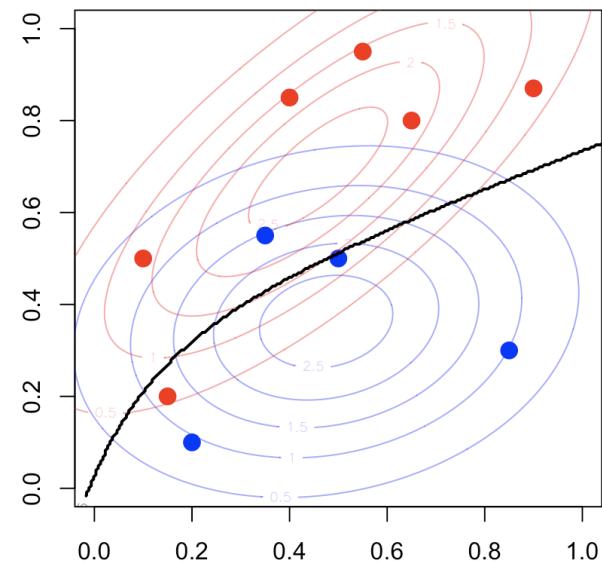
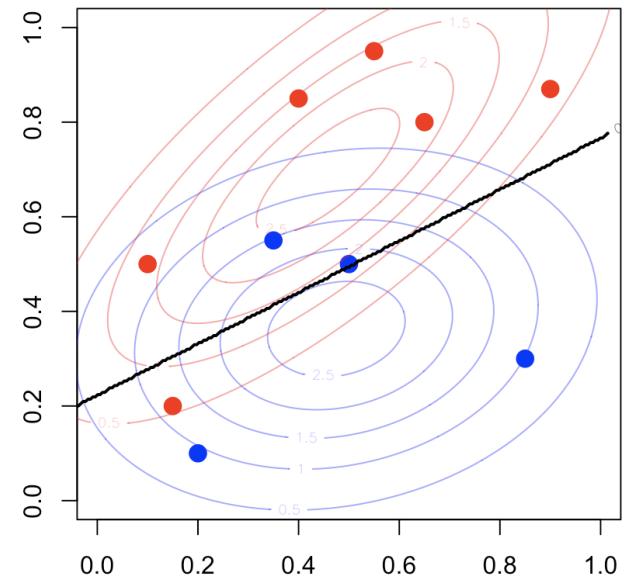
1 predlda = function(x,y) predict(fit_lda, data.
                      frame(x1=x,x2=y))$class==1
2 vv=outer(vu,vu,predlda)
3 contour(vu,vu,vv,add=TRUE,lwd=2,levels = .5)

```

```

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

```



Note that there are strong links with the logistic regression.

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

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

is equal to

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

which is linear in \mathbf{x}

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

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 $\Sigma^{-1}[\mu_1 - \mu_0]$ 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

$$\frac{\text{variance between}}{\text{variance within}} = \frac{[\omega\mu_1 - \omega\mu_0]^2}{\omega^\top \Sigma_1 \omega + \omega^\top \Sigma_0 \omega}$$

which is maximal when ω is proportional to $\Sigma^{-1}[\mu_1 - \mu_0]$.

Classification : Neural Nets

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

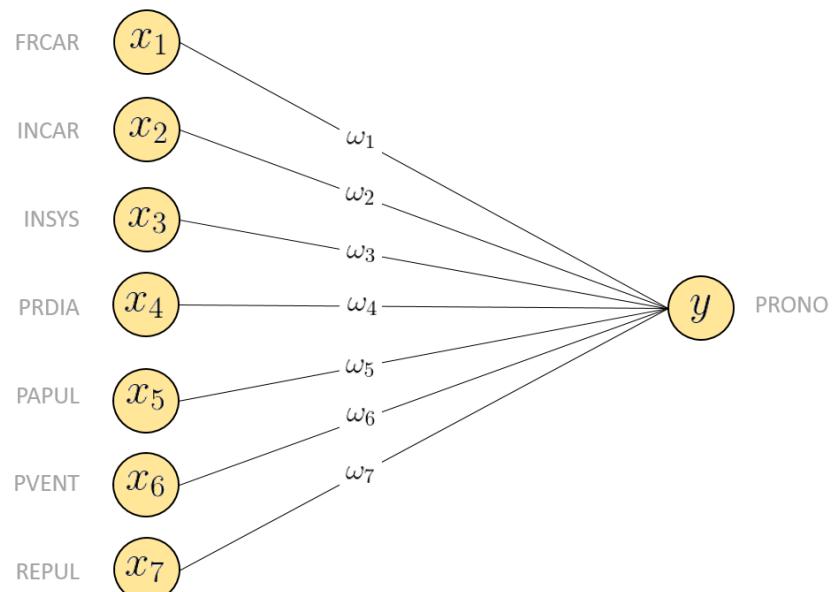
This function f is called the **activation function**.

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

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

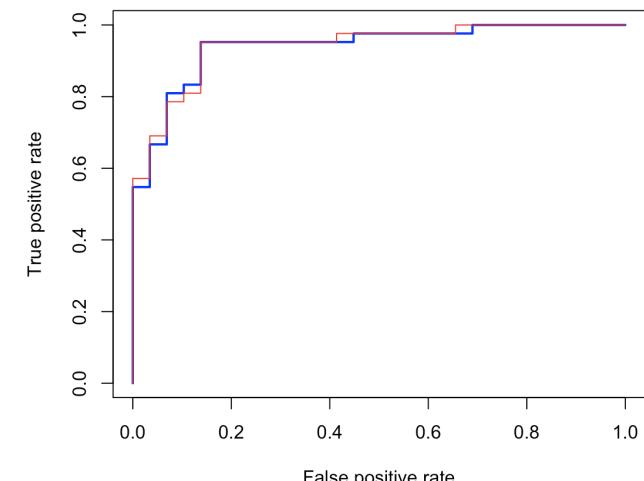


Classification : Neural Nets

```

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

```



Classification : Neural Nets

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

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

or cross-entropy

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

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

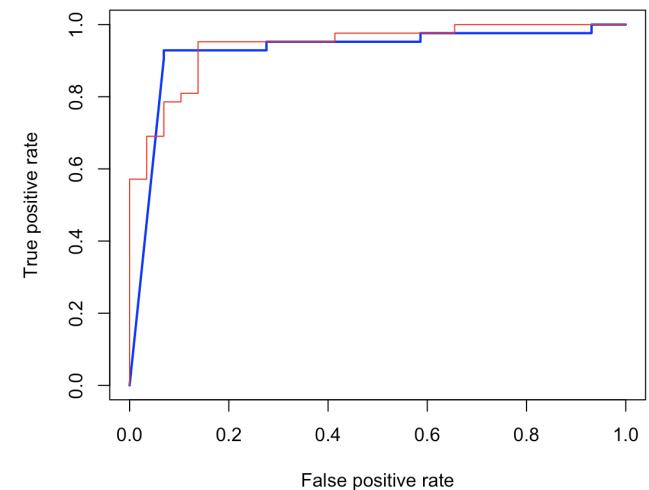
we want to solve

$$\boldsymbol{\omega}^* = \operatorname{argmin} \left\{ \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(\omega_0 + \mathbf{x}_i^\top \boldsymbol{\omega})) \right\}$$

```

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

```

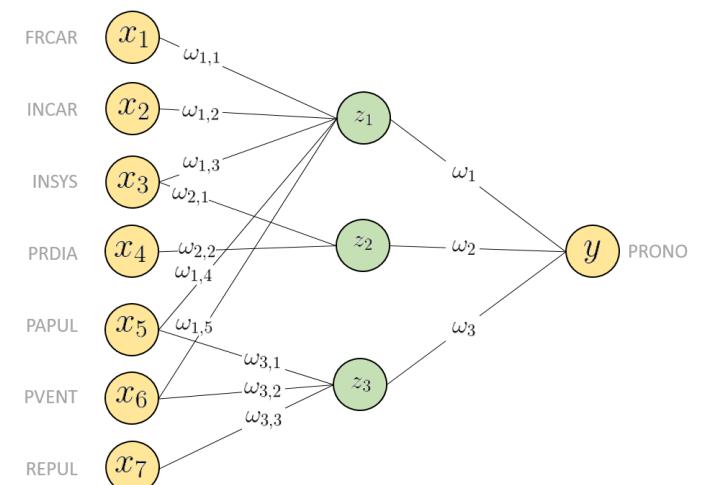


Consider a single hidden layer, with 3 different neurons, so that

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

or

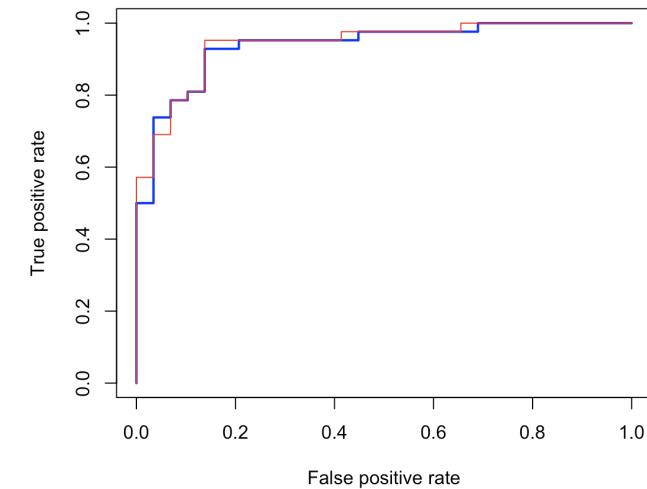
$$p(\mathbf{x}) = f \left(\omega_0 + \sum_{h=1}^3 \omega_h f_h (\omega_{h,0} + \mathbf{x}^\top \boldsymbol{\omega}_h) \right)$$



```

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

```



If

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

we want to solve, with back propagation,

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

For practical issues, let us center all variables

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

The loss function is here

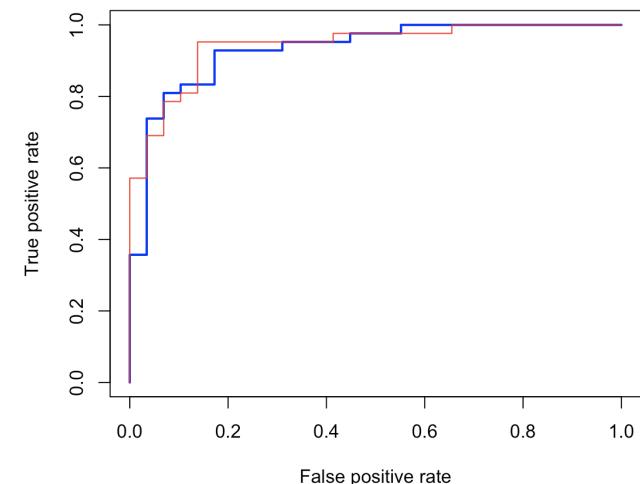
```
1 loss = function(weights){
2   weights_1 = weights[0:(1:7)]
3   weights_2 = weights[7:(1:7)]
4   weights_3 = weights[14:(1:7)]
5   weights_ = weights[21+1:4]
```

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

```

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

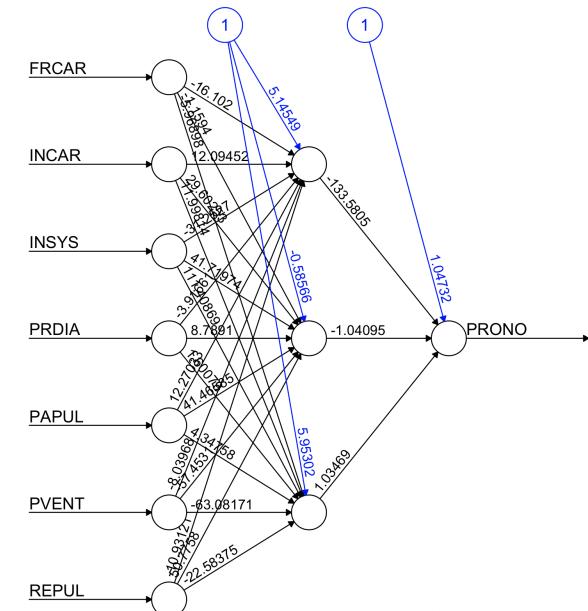
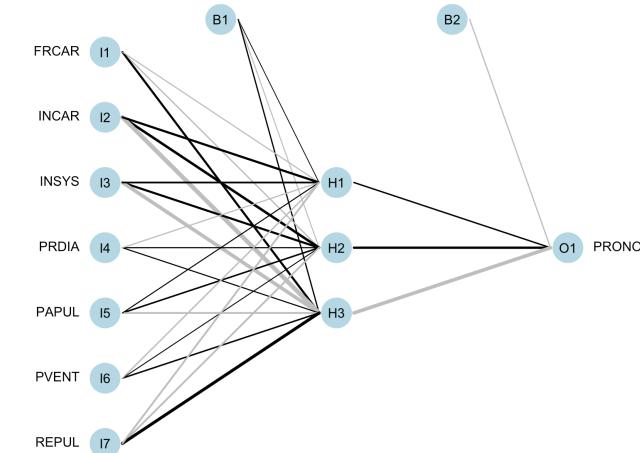
```



```

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

```



Classification : Neural Nets

Of course, one can consider a multiple layer network,

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

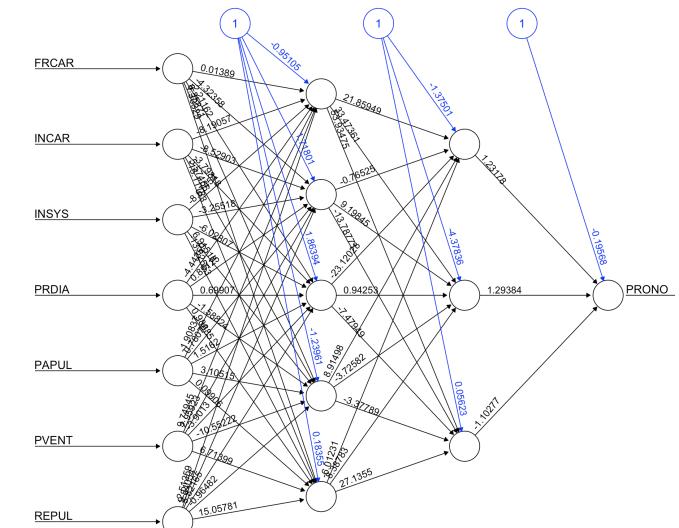
where

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

```

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

```



SVM : Support Vector Machine

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

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

subject to $y_i \cdot (\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$, with $\xi_i \geq 0, \forall i = 1, \dots, n$

```

1 n = length(myocarde[, "PRONO"])
2 myocarde0 = myocarde
3 myocarde0$PRONO = myocarde$PRONO*2-1
4 C = .5
5 f = function(param){
6   w = param[1:7]
7   b = param[8]
8   xi = param[8+1:nrow(myocarde)]
9   .5*sum(w^2) + C*sum(xi)}
10 grad_f = function(param){
11   w = param[1:7]
```

```

12   b = param[8]
13   xi = param[8+1:nrow(myocarde)]
14   c(2*w, 0, rep(C, length(xi)))}
```

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

```

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

then use

```
1 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

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

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

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

Classification : Support Vector Machine (SVM) - Primal Problem

```

1 sol = solve.QP(D+eps*diag(n+7+1), d, t(A), b, meq=1)
2 qpsol = sol$solution
3 (omega = qpsol[1:7])
4 [1] -0.106642005446 -0.002026198103 -0.022513312261 -0.018958578746
     -0.023105767847 -0.018958578746 -1.080638988521
5 (b      = qpsol[n+7+1])
6 [1] 997.6289927

```

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

```

1 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

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

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

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

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

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

$$\frac{1}{2} \|\boldsymbol{\omega}\|^2 + \textcolor{blue}{C} \sum_{i=1}^n \boldsymbol{\xi}_i - \sum_{i=1}^n \alpha_i (y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + b) - 1 + \boldsymbol{\xi}_i) - \sum_{i=1}^n \beta_i \boldsymbol{\xi}_i$$

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

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

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

Classification : Support Vector Machine (SVM) - Dual Problem

```

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

```

Classification : Support Vector Machine (SVM) - Dual Problem

The two problems are directed in the sense that for all

$$\mathbf{x}\boldsymbol{\omega}^\top \mathbf{x} + b = \sum_{i=1}^n \alpha_i y_i (\mathbf{x}^\top \mathbf{x}_i) + b$$

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

```

1 omega = apply(qpsol*y*X, 2, sum)
2 omega
3
4      1          FRCAR          INCAR          INSYS
5 0.0000000000000243907  0.0550138658 -0.092016323  0.3609571899422
6          PRDIA          PAPUL          PVENT          REPUL
7 -0.109401796528869235669 -0.0485213403 -0.066005864  0.0010093656567

```

More generally

```

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

```

```
6   for (j in 1:n.samples){  
7     K[i,j] = X[i,] %*% X[j,] }}  
8 Dmat = outer(y,y) * K  
9 Dmat = as.matrix(nearPD(Dmat)$mat)  
10 dvec = rep(1, n.samples)  
11 Amat = rbind(y, diag(n.samples), -1*diag(n.samples))  
12 bvec = c(0, rep(0, n.samples), rep(-C, n.samples))  
13 res = solve.QP(Dmat,dvec,t(Amat),bvec=bvec, meq=1)  
14 a = res$solution  
15 bomega = apply(a*y*X,2,sum)  
16 return(bomega)  
17 }
```

Classification : Support Vector Machine (SVM) - Dual Problem

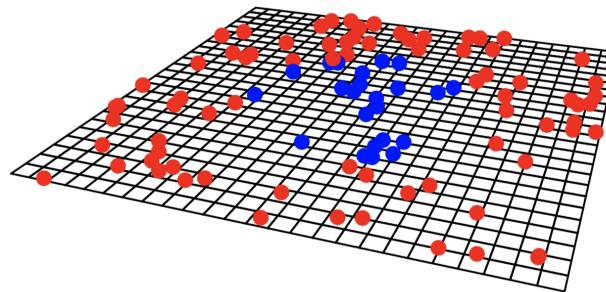
To relate this duality optimization problem to OLS, recall that $y = \mathbf{x}^\top \boldsymbol{\omega} + \varepsilon$, so that $\hat{y} = \mathbf{x}^\top \hat{\boldsymbol{\omega}}$, where $\hat{\boldsymbol{\omega}} = [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$.

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

Classification : Support Vector Machine (SVM) - Kernel Approach

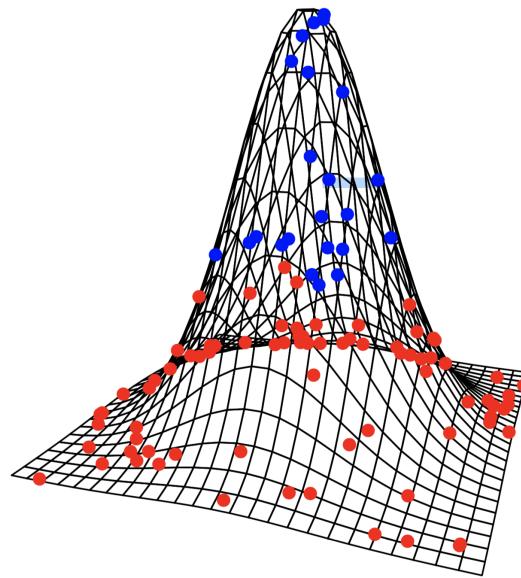
A positive kernel on \mathcal{X} is a function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ symmetric, and such that for any n , $\forall \alpha_1, \dots, \alpha_n$ and $\forall \mathbf{x}_1, \dots, \mathbf{x}_n$,

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



For example, the linear kernel is $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$. That's what we've been using here, so far. One can also define the product kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_i) \cdot \kappa(\mathbf{x}_j)$ where κ is some function $\mathcal{X} \rightarrow \mathbb{R}$.

Classification : Support Vector Machine (SVM) - Kernel Approach



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

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

```

1 linear.kernel = function(x1, x2) {
2   return (x1%*%x2)
3 }
4 svm.fit = function(X, y, FUN=linear.kernel, C=NULL) {

```

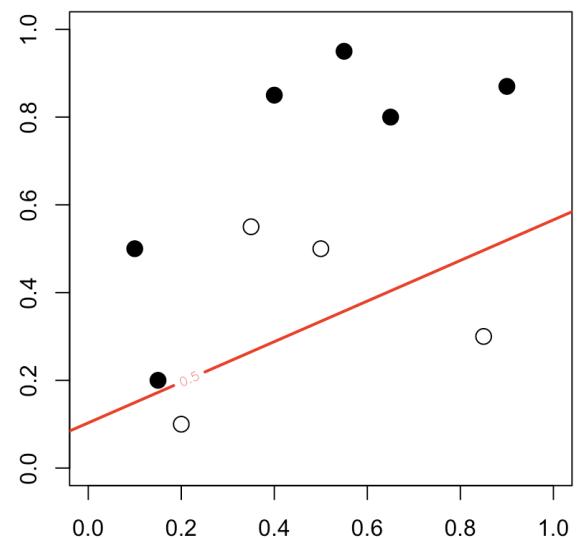
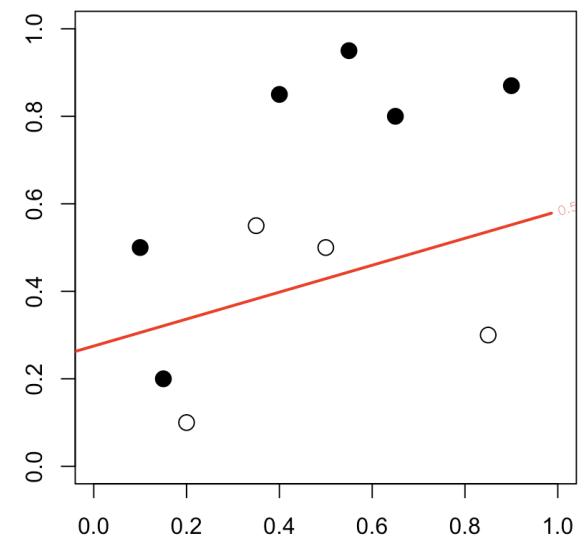
```

5 n.samples = nrow(X)
6 n.features = ncol(X)
7 K = matrix(rep(0, n.samples*n.samples), nrow=n.samples)
8 for (i in 1:n.samples){
9   for (j in 1:n.samples){
10     K[i,j] = FUN(X[i,], X[j,])
11   }
12 }
13 Dmat = outer(y,y) * K
14 Dmat = as.matrix(nearPD(Dmat)$mat)
15 dvec = rep(1, n.samples)
16 Amat = rbind(y, diag(n.samples), -1*diag(n.samples))
17 bvec = c(0, rep(0, n.samples), rep(-C, n.samples))
18 res = solve.QP(Dmat, dvec, t(Amat), bvec=bvec, meq=1)
19 a = res$solution
20 bomega = apply(a*y*X, 2, sum)
21 return(bomega)
22 }
```

```

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

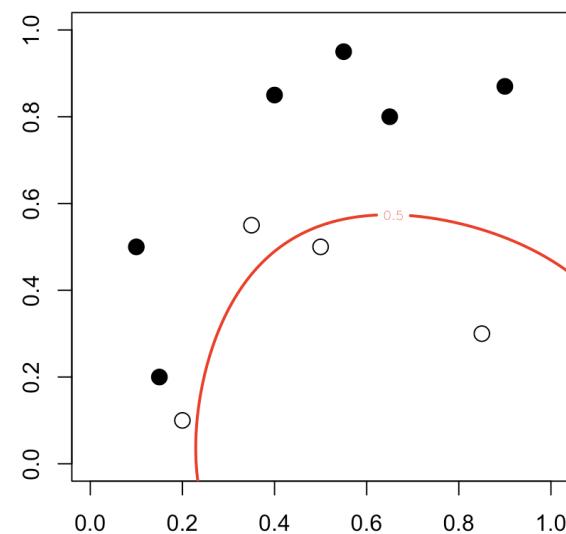
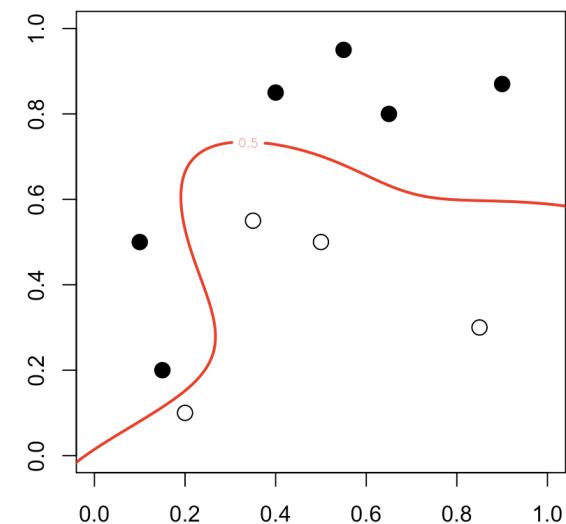
```



```

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

```

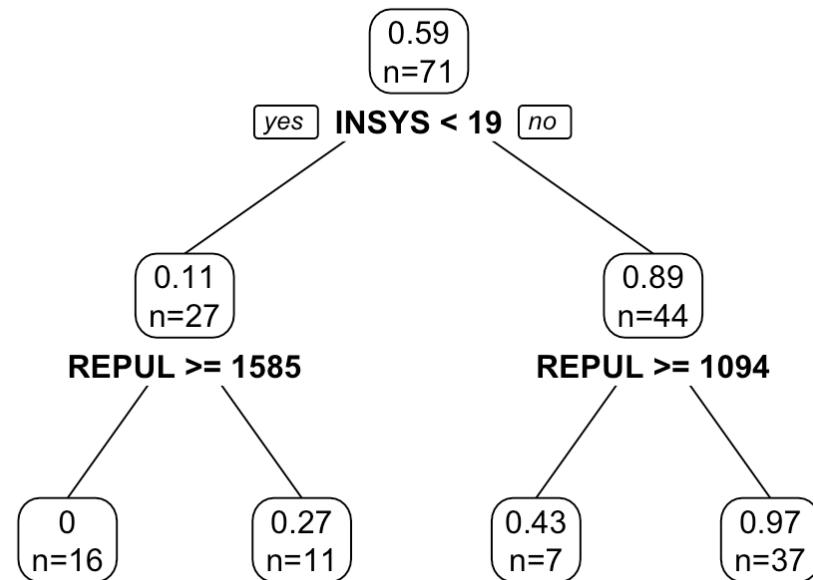


Classification : Classification Trees

```

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 ,

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

```

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

```

Classification : Classification Trees

```
1 -2*mean(myocarde$PRONO)*(1-mean(myocarde$PRONO))
2 [1] -0.4832375
3 gini(y=myocarde$PRONO , classe=myocarde$PRONO<Inf)
4 [1] -0.4832375
5 gini(y=myocarde$PRONO , classe=myocarde[,1]<=100)
6 [1] -0.4640415
```

Classification : Classification Trees

if we split, define index

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

the entropic measure is

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

```

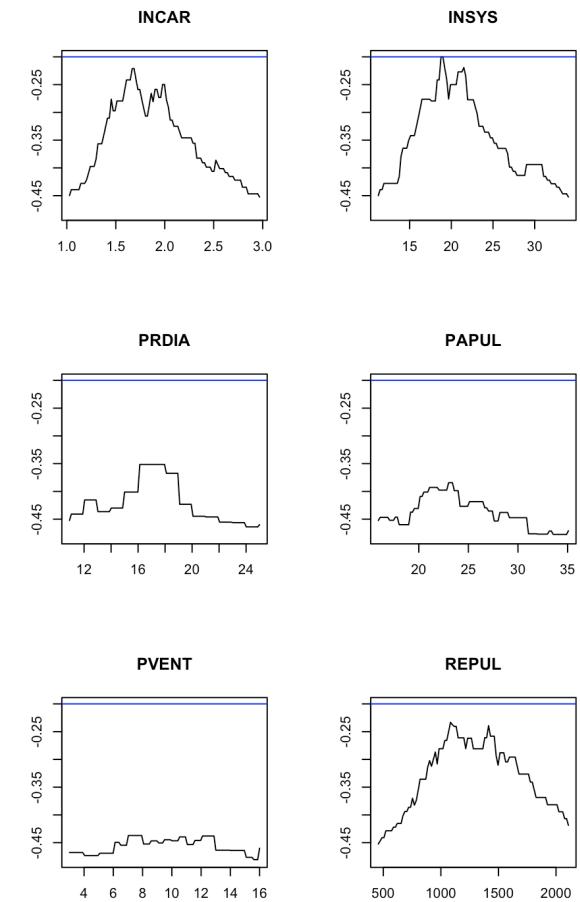
1 entropy = function(y, classe){
2   T = table(y, classe)
3   nx = apply(T, 2, sum)
4   pxy = T/matrix(rep(nx, each=2), nrow=2)
5   omega = matrix(rep(nx, each=2), nrow=2)/sum(T)
6   g = sum(omega*pxy*log(pxy))
7   return(g)

```

```

1 mat_gini = mat_v=matrix(NA,7,101)
2 for(v in 1:7){
3   variable=myocarde[,v]
4   v_seuil=seq(quantile(myocarde[,v],
5 6/length(myocarde[,v])),,
6 quantile(myocarde[,v],1-6/length(
7 myocarde[,v])),length=101)
8   mat_v[v,]=v_seuil
9   for(i in 1:101){
10 CLASSE=variable<=v_seuil[i]
11 mat_gini[v,i]=
12 gini(y=myocarde$PRONO,classe=CLASSE)})}
13 -(gini(y=myocarde$PRONO,classe=(myocarde
14 [,3]<19))-(
14 gini(y=myocarde$PRONO,classe=(myocarde[,3]<
15 Inf))/
15 gini(y=myocarde$PRONO,classe=(myocarde[,3]<
15 Inf)))
16 [1] 0.5002131

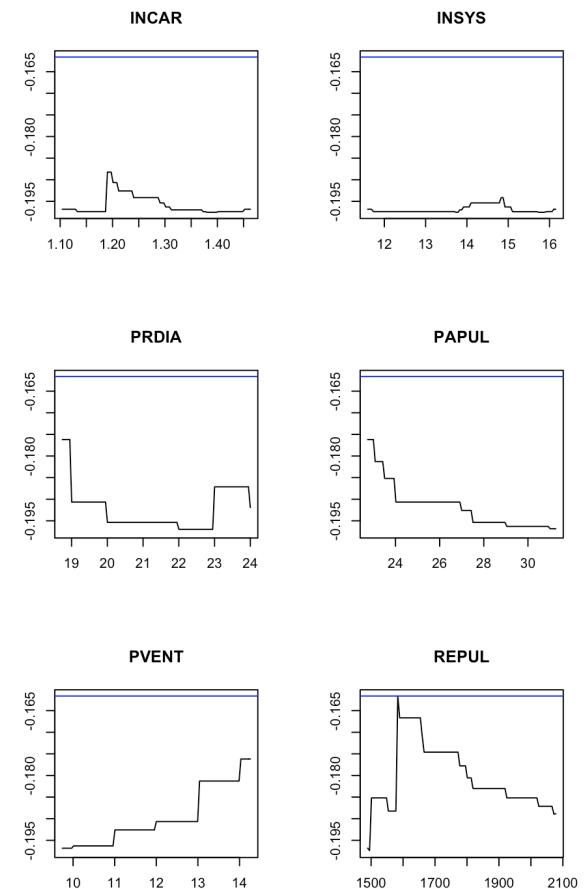
```



```

1 idx = which(myocarde$INSYS<19)
2 mat_gini = mat_v = matrix(NA,7,101)
3 for(v in 1:7){
4   variable = myocarde[idx,v]
5   v_seuil = seq(quantile(myocarde[idx,v],
6 7/length(myocarde[idx,v])),,
7 quantile(myocarde[idx,v],1-7/length(
8 myocarde[idx,v])), length=101)
9   mat_v[v,] = v_seuil
10  for(i in 1:101){
11    CLASSE = variable<=v_seuil[i]
12    mat_gini[v,i]=
13      gini(y=myocarde$PRONO[idx],classe=
14 CLASSE)}}
14 par(mfrow=c(3,2))
15 for(v in 2:7){
16 plot(mat_v[v,],mat_gini[v,])
17 }

```



```

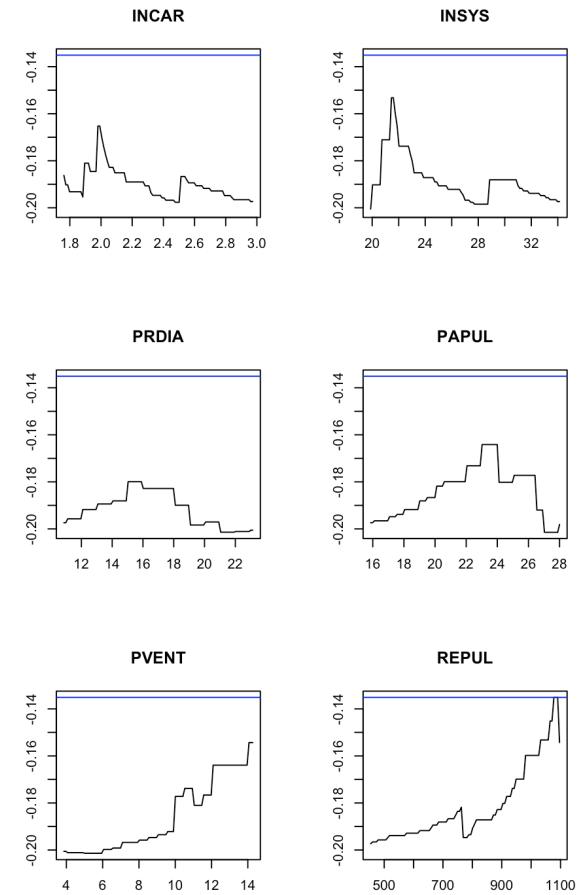
1 idx = which(myocarde$INSYS >= 19)
2 mat_gini = mat_v = matrix(NA, 7, 101)
3 for(v in 1:7){
4   variable=myocarde[idx,v]
5   v_seuil=seq(quantile(myocarde[idx,v],
6   6/length(myocarde[idx,v])), 
7   quantile(myocarde[idx,v],1-6/length(
8 myocarde[idx,v])), length=101)
9   mat_v[v,]=v_seuil
10  for(i in 1:101){
11    CLASSE=variable<=v_seuil[i]
12    mat_gini[v,i]=
13      gini(y=myocarde$PRONO[idx],
14            classe=CLASSE)}}
```

15 par(mfrow=c(3,2))

16 for(v in 2:7){

17 plot(mat_v[v,],mat_gini[v,])

18 }



Classification : Variable Importance, on Trees

```

1 cart = rpart(PRONO~., myocarde)
2 (split = summary(cart)$splits)
3
4      count ncat    improve      index        adj
5 INSYS     71    -1  0.58621312   18.850  0.0000000
6 REPUL     71     1  0.55440034 1094.500  0.0000000
7 INCAR     71    -1  0.54257020   1.690  0.0000000
8 PRDIA     71     1  0.27284114  17.000  0.0000000
9 PAPUL     71     1  0.20466714  23.250  0.0000000
10
11 REPUL    27     1  0.18181818 1585.000  0.0000000
12 PVENT    27    -1  0.10803571  14.500  0.0000000
13 PRDIA    27     1  0.10803571  18.500  0.0000000
14 PAPUL    27     1  0.10803571  22.500  0.0000000
15 INCAR    27     1  0.04705882   1.195  0.0000000
16
17 cart$variable.importance
18
19      INSYS      REPUL      INCAR      PAPUL      PRDIA      FRCAR      PVENT
20 10.364984 10.05108  8.21212  3.24415  2.82761  1.86230  0.33737

```

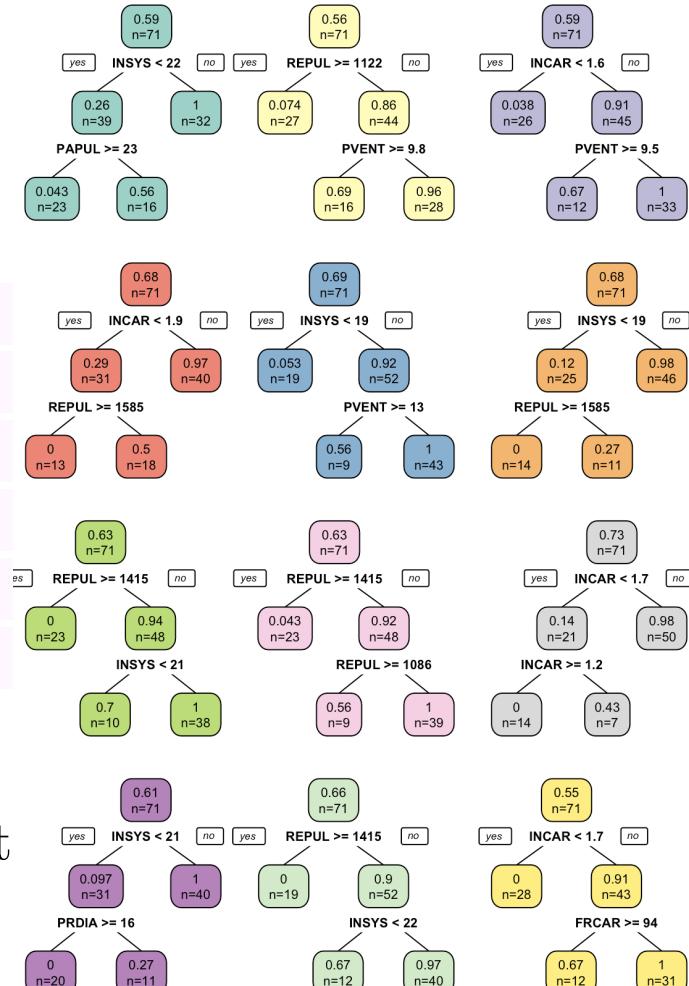
Classification : Bagging Trees

Random forests are obtained using

```

1 L_tree = list()
2 for(s in 1:1000){
3   idx = sample(1:n, size=n, replace=TRUE)
4   L_tree[[s]] = rpart(as.factor(PRONO) ~.,
5     myocarde[idx,])
6 }
```

Actually, random forest are slightly different : at each node, variables are randomly selected.



Boosting & Adaboost

Classification problem, $y_i \in \{\bullet, \circ\}$, consider a model at stage $k - 1$,

if $m_{k-1}(\mathbf{x}_i) \neq y_i$, increase the weight given to observation i

Boosting : weak learner

A weak model is a model slightly better than a pure random one (head/tails)

Boosting & Adaboost

Adaboost Algorithm

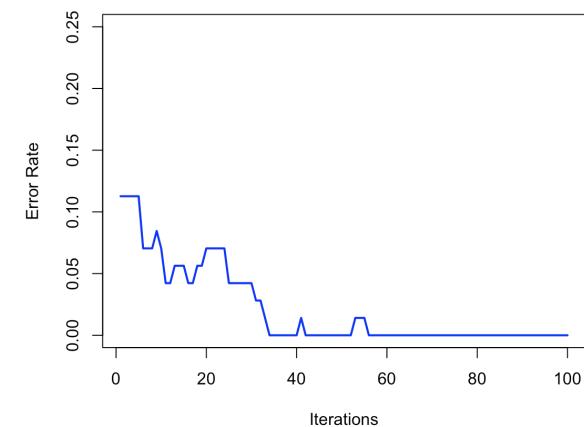
1. Set weights $\omega_i = 1/n$, $i = 1, \dots, n$
- 2 . For $k = 1, \dots$
 - (i) fit model on (y_i, \mathbf{x}_i) with weights ω_i , get $h_k(\mathbf{x})$
 - (ii) compute the error rate $\bar{\varepsilon}_k = \sum_{i=1}^n \tilde{\omega}_i \mathbf{1}_{y_i \neq h_k(\mathbf{x}_i)}$
 - (iii) compute $\alpha_k = \log \frac{1 - \bar{\varepsilon}_k}{\bar{\varepsilon}_k}$
 - (iv) reevaluate the weights $\omega_i = \omega_i \cdot e^{\alpha_k \mathbf{1}_{y_i \neq h_k(\mathbf{x}_i)}}$
3. The final model is $h_\kappa^\star(\mathbf{x}) = \sum_{k=1}^\kappa \alpha_k h_k(\mathbf{x})$

The error rate should not be too small ($\bar{\varepsilon}_k \leq 50\%$) to insure $\alpha_k > 0$

```

1 n_iter = 100
2 y = (myocarde[,"PRONO"]==1)*2-1
3 x = myocarde[,1:7]
4 error = rep(0,n_iter)
5 f = rep(0,length(y))
6 w = rep(1,length(y)) #
7 alpha = 1
8 library(rpart)
9 for(i in 1:n_iter){
10   w = exp(-alpha*y*f) *w
11   w = w/sum(w)
12   rfit = rpart(y~., x, w, method="class")
13   g = -1 + 2*(predict(rfit,x)[,2]>.5)
14   e = sum(w*(y*g<0))
15   alpha = .5*log ((1-e) / e )
16   alpha = 0.1*alpha
17   f = f + alpha*g
18   error[i] = mean(1*f*y<0)
19 }
20 plot(seq(1,n_iter),error)

```



Classification : Boosting (adabost)

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

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

```

1 for(i in 1:n_iter){
2   w_train = w_train*exp(-alpha*y_train*f_
3     train)
4   w_train = w_train/sum(w_train)
5   rfit = rpart(y_train~., x_train, w_train,
6     method="class")
7   g_train = -1 + 2*(predict(rfit,x_train)
8     [,2]>.5)
9   g_test = -1 + 2*(predict(rfit,x_test)
10    [,2]>.5)
11   e_train = sum(w_train*(y_train*g_train<0))
12   alpha = .5*log( (1-e_train) / e_train )
13   alpha = 0.1*alpha
14   f_train = f_train + alpha*g_train
15   f_test = f_test + alpha*g_test
16   train_error[i] = mean(1*f_train*y_train<0)
17   test_error[i] = mean(1*f_test*y_test<0)}

```

