Department of Mathematics and Computer Science
University of Southern Denmark, Odense

February 4, 2013
Marco Chiarandini

DM825 - Introduction to Machine Learning

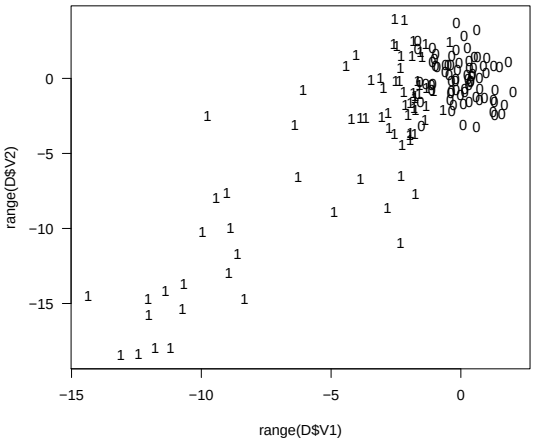
Sheet 3 – Solutions, Spring 2013 [pdf format]

Exercise Classification. The course website contains a data set of (x_n, y_n) pairs, where the x_n are 2-dimensional vectors and y_n is a binary label.

(a) Plot the data, using 0's and X's for the two classes. The plots in the following parts should be plotted on top of this plot.

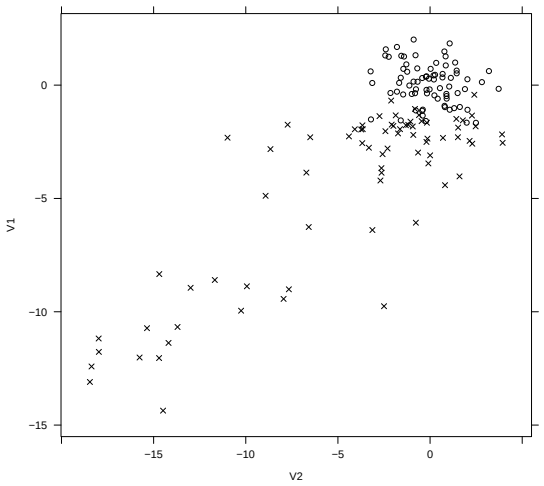
Solution

```
> D <- read.table("classification.dat")
> #plot(min(D$V1):max(D$V1),min(D$V1):max(D$V1),type="n")
> plot(range(D$V1),range(D$V2),type="n")
> text(D$V1,D$V2,D$V3)
```



Alternatively, with the lattice package (always explore the possibilities of the new functions you encounter via `example(f)`).

```
> require(lattice)
> print(
      xyplot(V1~V2,groups=V3,
             data=D,pch=c(1,4))
      )
> #print(
> #      xyplot(V1~V2,groups=V3,data=D,
> #             panel=function(x,y,subscripts,groups)
> #             ltext(x=x,y=y,
> #                   labels=groups[subscripts],#col=groups[subscripts],
> #                   cex=1)
> #             )
> #      )
```


0
−5
−10
−15
V1
−15
−10
−5
0
V2

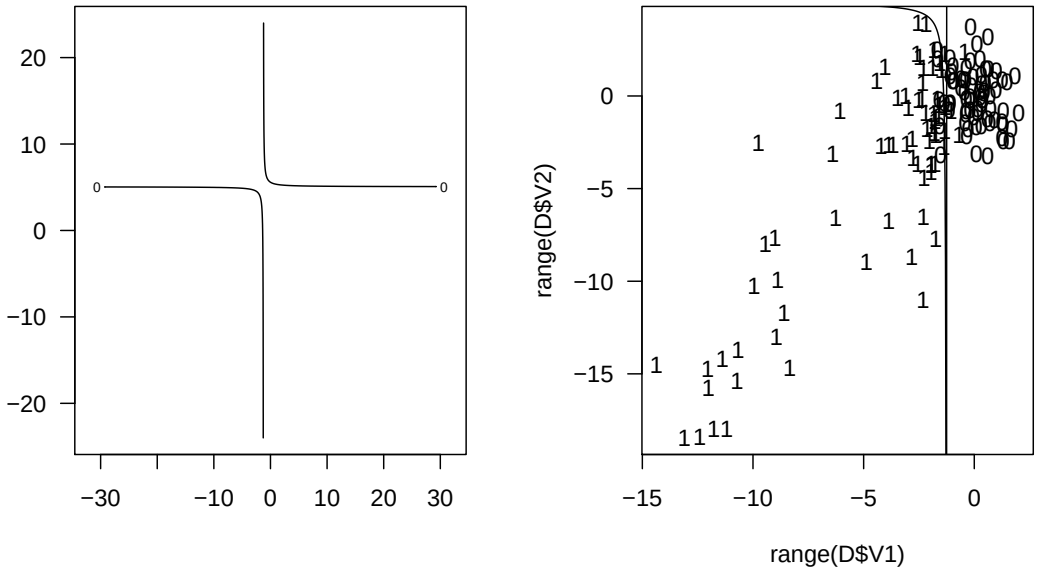
(b) Write a program to fit a logistic regression model using stochastic gradient ascent.

Solution We fit $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2$ with a batch implementation of the stochastic gradient.

```
> x1x2 <- D[,1]*D[,2]
> X <- cbind(1,D[,1],D[,2],x1x2)
> w.0 <- as.matrix(runif(4, min = -1, max = 1))
> w.t <- w.0
> i = 1
> alpha=0.2
> gradient <- 1000
> while (abs(gradient) > 0.1) {
   gradient <- 0
   for (i in 1:150) {
     ##j <- (i - 1)%%150 + 1
     xv <- t(as.matrix(X[i,]))
     yv <- xv %*% w.t
     hv <- 1/(1+exp(-yv))
     gradient <- gradient + drop(D[i,3] - hv) * t(xv)
   }
   w.t <- w.t + alpha * gradient
   i = i + 1
 }
```

We plot the curve $\frac{1}{1+e^{-\theta x}} = 0.5$ which reduces to $\vec{\theta}\vec{x} = 0$.

```
> x1 <- seq(-32,32,.2)
> x2 <- seq(-24,24,.2)
> f.gradient <- function(x,y) {
   apply(as.matrix(cbind(x,y)),1,
         function(l) c(1,l[1],l[2],l[1]*l[2]) %*% w.t)
 }
> zs <- outer(x1,x2,FUN=f.gradient)
> par(mfrow=c(1,2))
> contour(x1,x2,zs,levels=0)
> plot(range(D$V1),range(D$V2),type="n")
> text(D$V1,D$V2,D$V3)
> matlines(x <- seq(-15,2,.02),(-w.t[2]*x-w.t[1])/(w.t[3]+w.t[4]*x),lwd=1)
```



In the plot on the right there is a line mistakenly connecting two points at different sides of the discontinuity (ie, from $-\infty$ to ∞).

(c) Compare this outcome with the result attained using the `glm` function in R (check example in `predict.glm`).

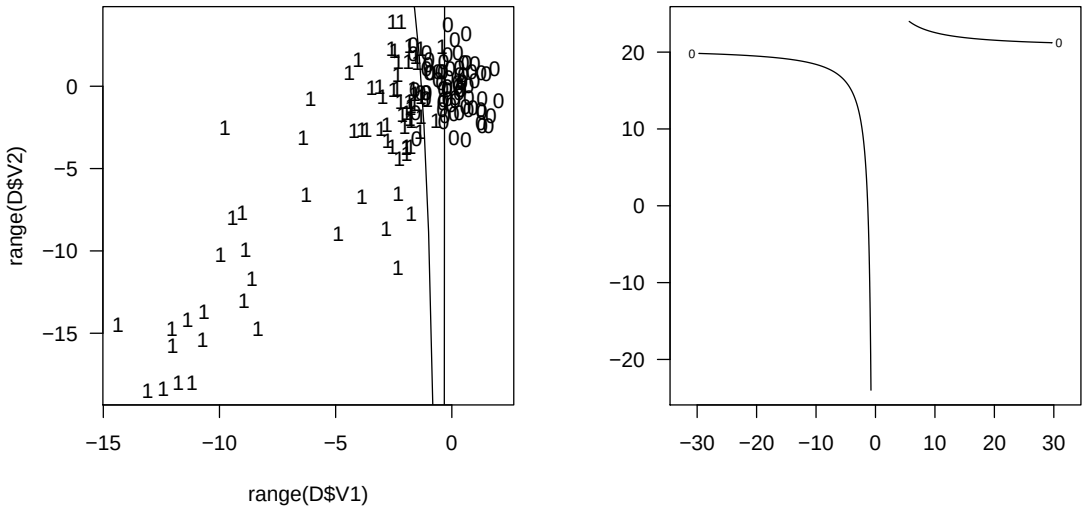
Solution

```
> reslogit <- glm(V3 ~ V1*V2, data=D, family=binomial(link="logit"))
> summary(reslogit)
>
> # components of the resulting object reslogit:
> # reslogit$coefficients: estimated regression coefficients
> # reslogit$fitted.values: estimated success probabilities
> # reslogit$residuals: residuals
> # reslogit$linear.predictors: the linear predictor b0+b1*x1+b2*x2
> #
```

The line that corresponds to $p = 0.5$:

```
> theta <- reslogit$coefficients
> par(mfrow=c(1,2))
> plot(range(D$V1),range(D$V2),type="n")
> text(D$V1,D$V2,D$V3)
> matlines(x <- seq(-15,2,.2),(-theta[2]*x-theta[1])/(theta[3]+theta[4]*x),lwd=1)
> x1 <- seq(-32,32,.2)
> x2 <- seq(-24,24,.2)
> f.glm <- function(x,y) {
   apply(as.matrix(cbind(x,y)),1,
         function(l) theta%*%c(1,l[1],l[2],l[1]*l[2]))
```

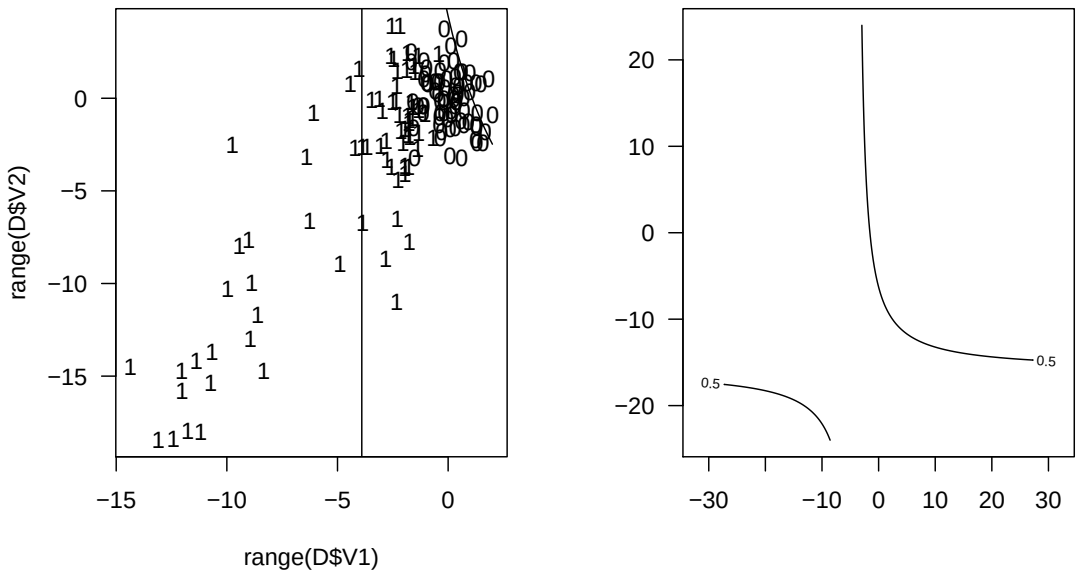
```
 }
> zs <- outer(x1,x2,FUN=f.glm)
> contour(x1,x2,zs,levels=0)
```



(d) Fit a linear regression to the problem, treating the class labels as real values 0 and 1. (You can solve the linear regression in any way you like, including solving the normal equations, using the LMS algorithm, or calling the built-in `lm` routine in R). Plot the line where the linear regression function is equal to 0.5.

Solution

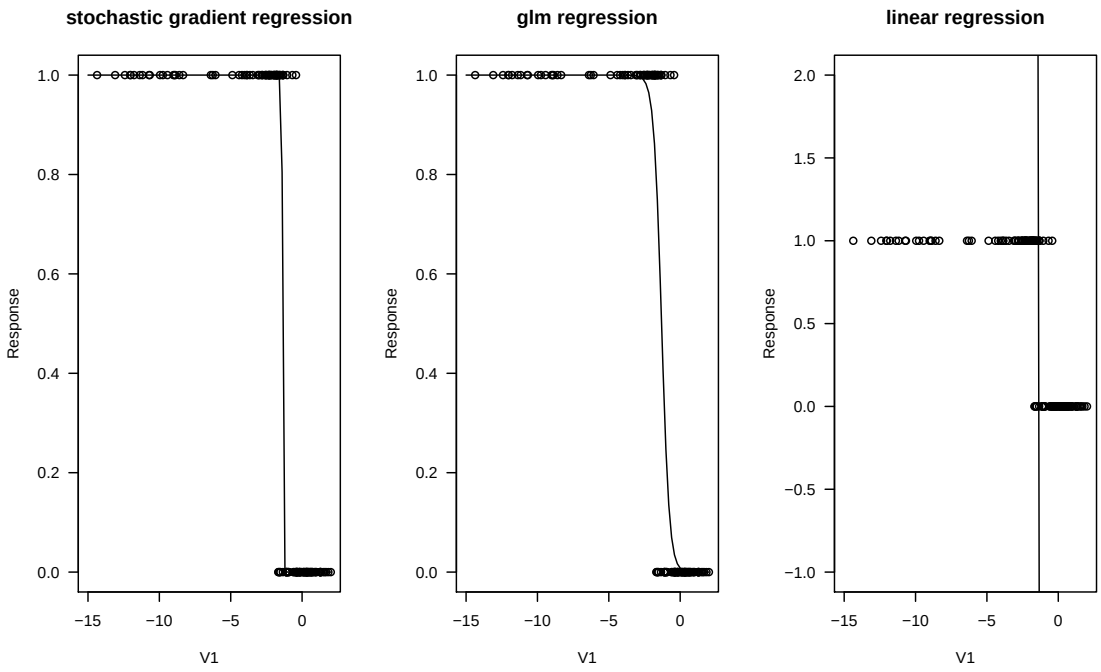
```
> reslm <- lm(V3 ~ V1*V2, data=D)
> summary(reslm)
> theta.lm <- reslm$coefficients

> par(mfrow=c(1,2))
> plot(range(D$V1),range(D$V2),type="n")
> text(D$V1,D$V2,D$V3)
> matlines(x <- seq(-15,2,.2),(-theta.lm[2]*x-theta.lm[1])/(theta.lm[3]+theta.lm[4]*x)
> x1 <- seq(-32,32,.2)
> x2 <- seq(-24,24,.2)
> f.linear <- function(x,y) {
   apply(as.matrix(cbind(x,y)),1,
         function(l) theta.lm%*%c(1,l[1],l[2],l[1]*l[2]))
 }
> zs <- outer(x1,x2,FUN=f.linear)
> contour(x1,x2,zs,levels=0.5)
```



Let's plot at a fixed value 0.8 for V2

```
> x1 <- seq(-15,2,.2)
> x2 <- -0.8
> ## from the stochastic gradient
> lp <- w.t[1] + w.t[2]*x1 + w.t[3]*x2 + w.t[4]*x1*x2
```

```
> pr <- exp(lp)/(1+exp(lp))
> par(mfrow=c(1,3))
> plot(D$V1,D$V3,xlim=c(-15,2),ylim=c(0,1),xlab="V1",
      main="stochastic gradient regression",ylab="Response")
> lines(x1,pr,lty=1)
> ## from the glm regression
> lp <- reslogit$coefficients[1]+reslogit$coefficients[2]*x1
>       +reslogit$coefficients[3]*x2+reslogit$coefficients[3]*x2*x1
> pr <- exp(lp)/(1+exp(lp))
> plot(D$V1,D$V3,xlim=c(-15,2),ylim=c(0,1),main="glm regression",
      xlab="V1",ylab="Response")
> lines(x1,pr,lty=1)
> ## from the linear model
> pr <- w.t[1] + w.t[2]*x1 + w.t[3]*x2 + w.t[4]*x1*x2
> plot(D$V1,D$V3,xlim=c(-15,2),ylim=c(-1,2),xlab="V1",
      main="linear regression",ylab="Response")
> lines(x1,pr,lty=1)
>
```



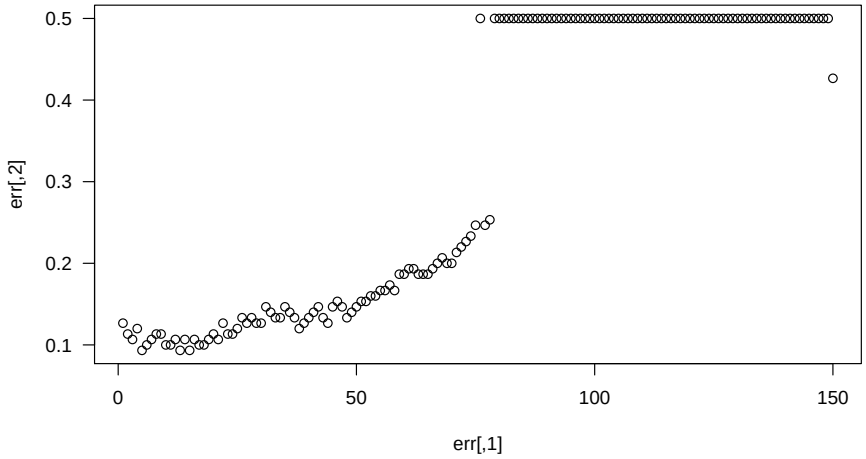
(e) Implement a k-nearest neighbor classifier. Plot the training and predicted points for $k = 3$. Further, show graphically the behavior of the loss function as k increases from $k = 1$ to the size of the training set.

Solution A k-nearest neighbor classifier can be found in the package `class` or in the package `FNN`. Be aware of the difference between classification and regression!

```
> # we read the test data
> T <- read.table("classification.test")
> names(T) <- c("X1","X2","class")
> # confusion matrix
> cm <- function(actual, predicted)
   {
     t <- table(predicted,actual)
     t[apply(t,2,function(c) order(-c)[1]),]
   }
> library(class)
> T$knn.predictions <- knn(D[,c(1,2)],T[,c(1,2)],D[,3],k=3,prob=FALSE)
> (m <- cm(T$class,T$knn.predictions))

          actual
predicted  0  1
        0 63  7
        1  9 71
```

We take $L(\vec{G}, \hat{G}(\vec{x})) = \sum_{i=1}^{m} I(G^i \neq \hat{G}^i(\vec{x})$ as definition of the loss function.

```
> err <- matrix(nrow=150,ncol=2)
> for (k in 1:nrow(err))
   {
     knn.predictions <- knn(D[,c(1,2)],T[,c(1,2)],D[,3],k=k,prob=FALSE)
     m <- cm(T$class,knn.predictions)
     err[k,] <- c(k,1-sum(diag(m))/sum(m))
   }
> plot(err)
```



(f) The data set is a separate data set generated from the same source. Test your fits from parts (b), (c), and (d) and with the k-nearest neighbor on these data and compare the results.

Solution

```
> (m <- cm(T$class,T$knn.predictions))

          actual
predicted  0  1
        0 63  7
        1  9 71

> (err.knn <- 1-sum(diag(m))/sum(m))

[1] 0.107

> h.logit <- function(thetav,xv) {
   1/(1+exp(-xv%*%thetav))
 }
> T$grd.prd <- apply(T[,c(1,2)],1,
                            function(xv) round(h.logit(w.t, c(1,xv[1],xv[2],xv[1]*xv[2
                            )
> (m <- cm(T$class,T$grd.prd))

          actual
predicted  0  1
        0 66  8
        1  6 70

> (err.grd <- 1-sum(diag(m))/sum(m))

[1] 0.0933
```

```
> T$glm.prd <- apply(T[,c(1,2)],1,
                             function(xv) round(h.logit(theta, c(1,xv[1],xv[2],xv[1]*xv
                             )
> (m <- cm(T$class,T$glm.prd))

          actual
predicted  0  1
        0 65  8
        1  7 70

> (err.glm <- 1-sum(diag(m))/sum(m))

[1] 0.1

> h.lin <- function(thetav,xv) {
   xv%*%thetav
 }
> T$lin.prd <- apply(T[,c(1,2)],1,
                             function(xv) h.lin(theta.lm, c(1,xv[1],xv[2],xv[1]*xv[2]))
                             )
> T$lin.prd <- ifelse(T$lin.prd>0.5,1,0)
> (m <- cm(T$class,T$lin.prd))

          actual
predicted  0  1
        0 66 12
        1  6 66

> (err.lin <- 1-sum(diag(m))/sum(m))

[1] 0.12
```