Lab 11: Penalized Logistic Regression

June 7, 2003

In this lab we introduce you to penalized logistic regression algorithms that are available in R. The examples in this section will rely on the data reported in Golub et al (1999). The basic comparisons here are between ALL to AML.

```
> library(Biobase)

Welcome to Bioconductor
         To view some introductory material -- look at our vignettes
         Simply type: openVignette()
         to see the available vignettes
         To read a vignette see the openVignette help page for details
Creating a new generic function for "summary" in package
Biobase

> library(annotate)
> library(golubEsets)
> library(genefilter)
> library(Design)

Design library by Frank E Harrell Jr

Type library(help='Design'), ?Overview, or ?Design.Overview')
to see overall documentation.

Loading required package: Hmisc
Hmisc library by Frank E Harrell Jr

Type library(help='Hmisc'), ?Overview, or ?Hmisc.Overview')
to see overall documentation.

Hmisc redefines [.factor to drop unused levels of factor variables
when subscripting. To prevent this behaviour, issue the command
```

```
options(drop.unused.levels=F).


Attaching package 'Hmisc':


        The following object(s) are masked from package:Design :

         .R. .SV4. under.unix


        The following object(s) are masked from package:methods :

         show


        The following object(s) are masked from package:base :

         %in% [.factor [.terms interaction


Attaching package 'Design':


        The following object(s) are masked _by_ package:Hmisc :

         .R. .SV4. under.unix
```

```
> library(Hmisc)
```

We will follow similar steps as in Lab4 to prefilter the data, i.e. we transform the data in much the same way that **?** did. The sequence of commands needed are given below.

```
> data(golubTrain)
> data(golubTest)
> LS <- exprs(golubTrain)
> cl <- golubTrain$ALL.AML
> TS <- exprs(golubTest)
> clts <- golubTest$ALL.AML
> LS[LS < 100] <- 100
> TS[TS < 100] <- 100
> LS[LS > 16000] <- 16000
```

```
> TS[TS > 16000] <- 16000
> mmfilt <- function(r = 5, d = 500, na.rm = TRUE) {
+     function(x) {
+         minval <- min(x, na.rm = na.rm)
+         maxval <- max(x, na.rm = na.rm)
+         (maxval/minval > r) && (maxval - minval > d)
+     }
+ }
> mmfun <- mmfilt()
> ffun <- filterfun(mmfun)
> good <- genefilter(cbind(LS, TS), ffun)
> sum(good)

[1] 3571

> LSsub <- log10(LS[good, ])
> TSsub <- log10(TS[good, ])
> LTsub <- cbind(LSsub, TSsub)
```

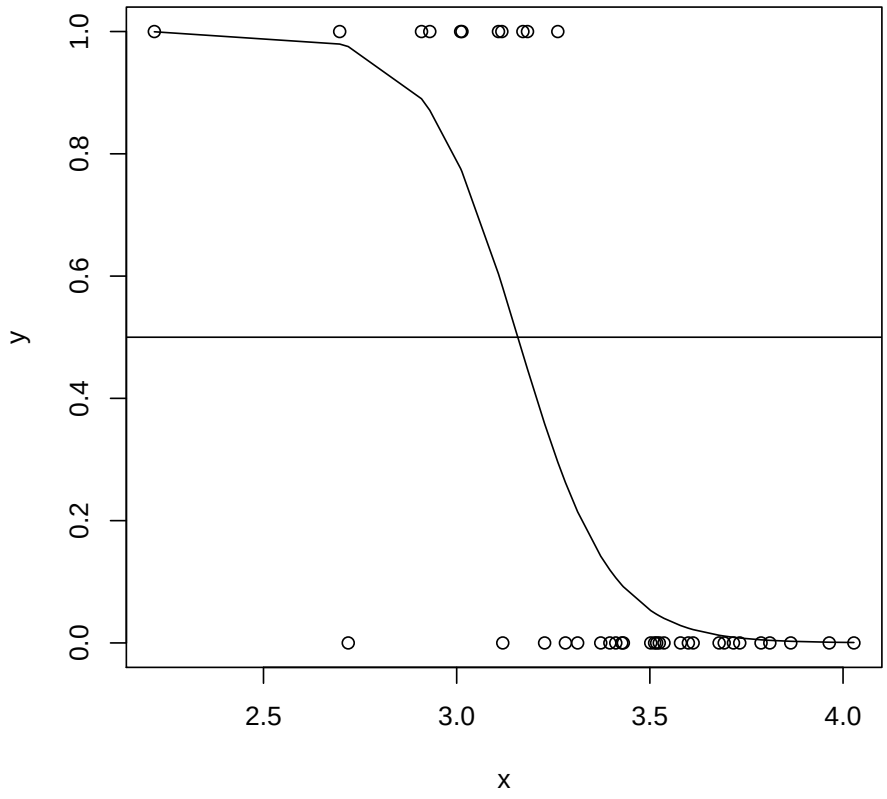
Once we have subset the data to the appropriate cases we next perform some specific filtering to remove any genes that show little variation across samples. We explicitly remove the non-informative ones before applying logistic regression, using either t-test statistics or using an ANOVA like F-test on the learning sample (beware of bias selection).

```
> gf <- gapFilter(900, 1500, 0.1)
> ff <- filterfun(gf)
> xx <- 10^LSsub
> good <- genefilter(xx, gf)
> sum(good)

[1] 280

> LS2 <- LSsub[good, ]
> TS2 <- TSsub[good, ]
```

To illustrate how to fit a simple logistic regression model we select one gene form the list and plot the resulting fit in what follows. Notice the lack of discrimination power.

```
> y = as.integer(cl) - 1
> x = LS2[99, ]
> aux = sort(x)
> fit = glm(y ~ x, family = "binomial")
> auxy = fit$coef[1] + fit$coef[2] * sort(x)
```

```
> auxy = plogis(auxy)
> plot(x, y)
> lines(sort(x), auxy)
> abline(0.5, 0)
```

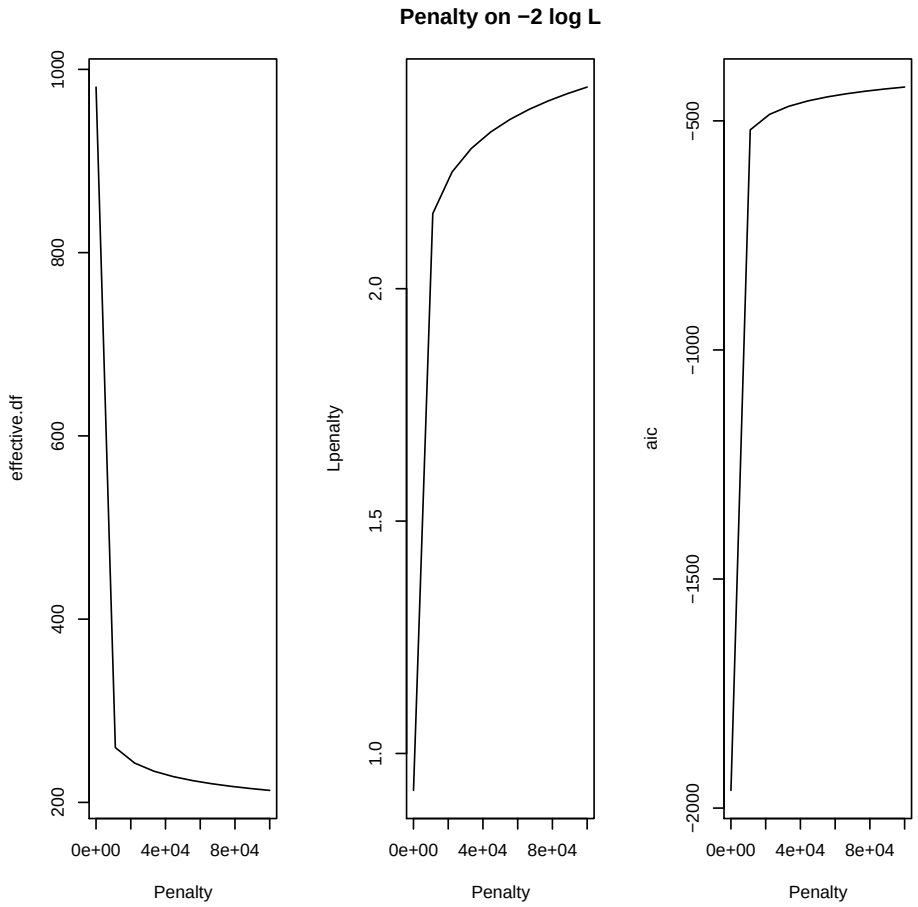


We will now perform a penalized logistic regression fit on the training sample usig the appropriate procedures in the *Design*. The penalty parameter is chosen by cross validation and the next commands may take some time for their execution.

```
> X = t(LS2)
> Y = as.integer(cl) - 1
> n = nrow(X)
> p = ncol(X)
> X.val = t(TS2)
> Y.val = as.integer(clts) - 1
> pm <- diag(diag(var(X)))
> Penalty <- seq(10, 10^5, length = 10)
> reps <- length(Penalty)
```

```
> effective.df <- effective.df2 <- aic <- aic2 <- deviance.val <- Lpenalty <- single(
> n.t <- round(n^0.75)
> ncv <- 10
> deviance <- matrix(NA, nrow = reps, ncol = length(ncv))
> for (i in 1:reps) {
+     pen <- log10(Penalty[i])
+     cat(format(pen), "")
+     f.full <- lrm.fit(X, Y, penalty.matrix = pen * pm)
+     Lpenalty[i] <- pen * t(f.full$coef[-1]) %*% pm %*% f.full$coef[-1]
+     f.full.nopenalty <- lrm.fit(X, Y, penalty.matrix = 0.05 *
+         pm, maxit = 1)
+     info.matrix.unpenalized <- solve(f.full.nopenalty$var)
+     effective.df[i] <- sum(diag(info.matrix.unpenalized %*%
+         f.full$var)) - 1
+     lrchisq <- f.full.nopenalty$stats["Model L.R."]
+     aic[i] <- lrchisq - 2 * effective.df[i]
+     pred <- plogis(f.full$linear.predictors)
+     score.matrix <- cbind(1, X) * (Y - pred)
+     sum.u.uprime <- t(score.matrix) %*% score.matrix
+     effective.df2[i] <- sum(diag(f.full$var %*% sum.u.uprime))
+     aic2[i] <- lrchisq - 2 * effective.df2[i]
+     dev <- 0
+     for (j in 1:max(ncv)) {
+         s <- sample(1:n, n.t)
+         cof <- lrm.fit(X[s, ], Y[s], penalty.matrix = pen *
+             pm)$coef
+         pred <- cof[1] + (X[-s, ] %*% cof[-1])
+         dev <- dev - 2 * sum(Y[-s] * pred + log(1 - plogis(pred)))
+         for (k in 1:length(ncv)) if (j == ncv[k])
+             deviance[i, k] <- dev/j
+     }
+     pred.val <- f.full$coef[1] + (X.val %*% f.full$coef[-1])
+     prob.val <- plogis(pred.val)
+     deviance.val[i] <- -2 * sum(Y.val * pred.val + log(1 - prob.val))
+ }

1 4.046105 4.346939 4.522966 4.647872 4.744762 4.82393 4.890868 4.948853 5

> par(mfrow = c(1, 3))
> plot(Penalty, effective.df, type = "l")
> lines(Penalty, effective.df2, lty = 2)
> plot(Penalty, Lpenalty, type = "l")
> title("Penalty on -2 log L")
```

```
> plot(Penalty, aic, type = "l")
> lines(Penalty, aic2, lty = 2)
```



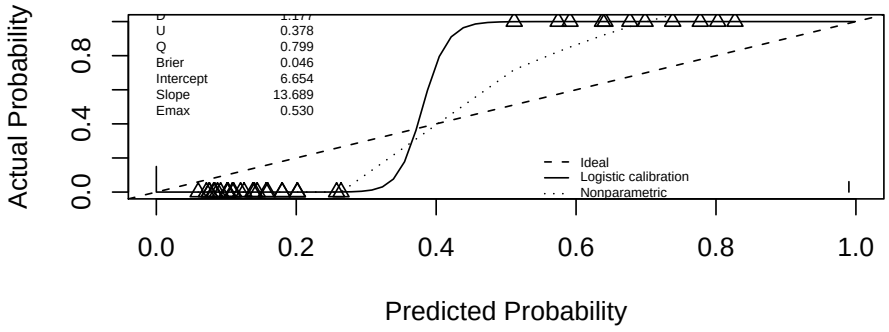
We can now exploit the previous result and adjust a penalized regression model with the appropriate amount of penalization. The resulting fit is then used to predict the cells in the test sample.

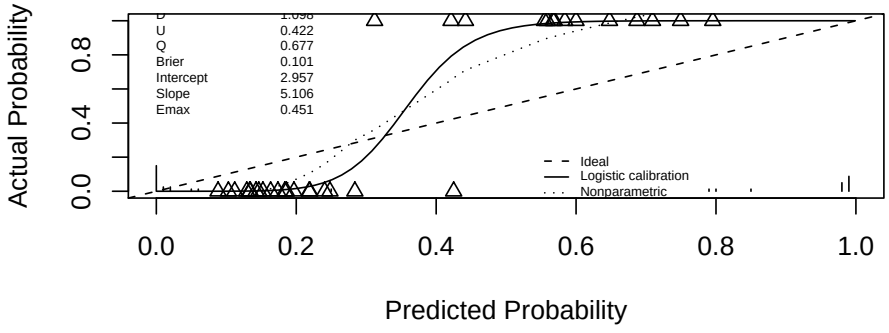
```
> par(mfrow = c(2, 1))
> pen = 200
> f.full <- lrm.fit(X, Y, penalty.matrix = pen * pm)
> predLS <- plogis(f.full$linear.predictors)
> yLS = as.integer(cl) - 1
> yTS = as.integer(clts) - 1
> val.prob(predLS, yLS, m = 1, cex = 0.5)

         Dxy      C (ROC)           R2            D     D:Chi-sq          D:p
1.000000e+00 1.000000e+00 9.999606e-01 1.176952e+00 4.572417e+01 1.361322e-11
           U     U:Chi-sq          U:p            Q        Brier    Intercept
```

```
3.781328e-01 1.636904e+01 2.789376e-04 7.988191e-01 4.646823e-02 6.654375e+00
       Slope         Emax         Eavg
1.368930e+01 5.303418e-01 1.734935e-01
```

```
> pred.val <- f.full$coef[1] + (X.val %*% f.full$coef[-1])
> predTS <- plogis(pred.val)
> val.prob(predTS, yTS, m = 1, cex = 0.5)
```

```
         Dxy      C (ROC)           R2            D     D:Chi-sq          D:p
9.857143e-01 9.928571e-01 9.112906e-01 1.098289e+00 3.834181e+01 5.937671e-10
           U     U:Chi-sq          U:p            Q        Brier    Intercept
4.215417e-01 1.633242e+01 2.840928e-04 6.767468e-01 1.005683e-01 2.957099e+00
       Slope         Emax         Eavg
5.106450e+00 4.506439e-01 2.040048e-01
```





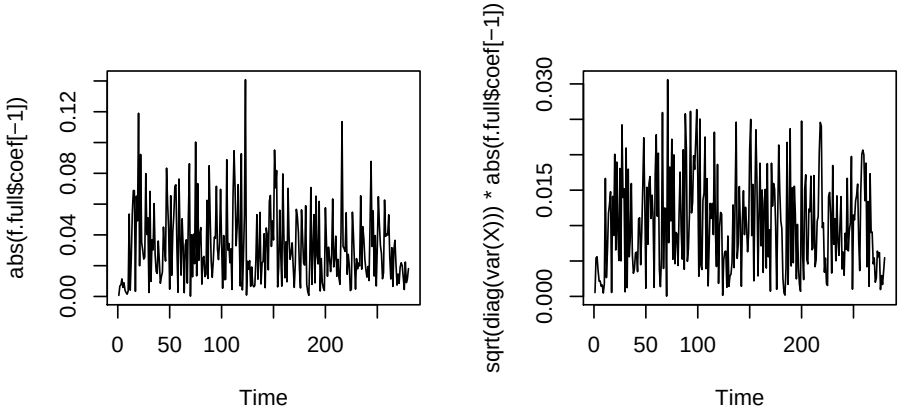
We can summarize the results from the following concordance table.

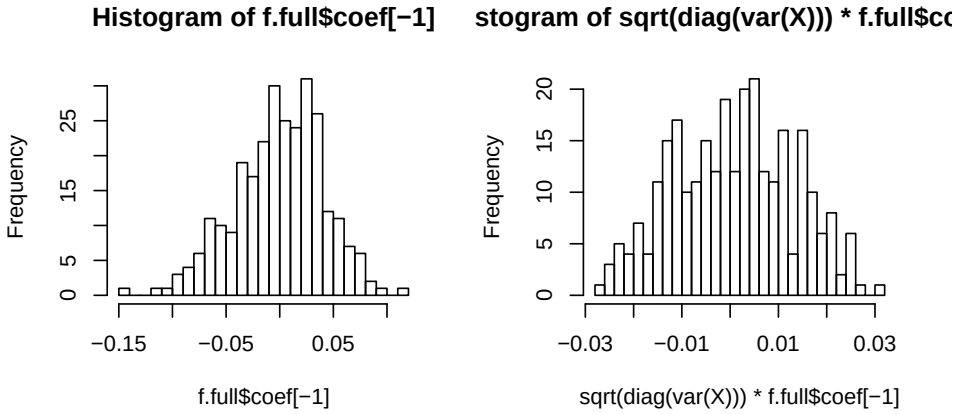
```
> con <- function(x, y) {
+     tab <- table(x, y)
```

```
+     print(tab)
+     diag(tab) <- 0
+     cat("error rate = ", round(100 * sum(tab)/length(x), 2),
+         "%\n")
+     invisible()
+ }
> predProbLS = (predLS > 0.5)
> predProbTS = (predTS > 0.5)
> con(as.integer(predProbLS), yLS)

   y
x   0  1
  0 27  0
  1  0 11
error rate =  0 %

> con(as.integer(predProbTS), yTS)

   y
x   0  1
  0 20  3
  1  0 11
error rate =  8.82 %
```

Finally to get a feeling on how the procedue penalized the coeffcients we display them in several plots.

```
> par(mfrow = c(2, 2))
> plot.ts(abs(f.full$coef[-1]))
> plot.ts(sqrt(diag(var(X))) * abs(f.full$coef[-1]))
> hist(f.full$coef[-1], breaks = 30)
> hist(sqrt(diag(var(X))) * f.full$coef[-1], breaks = 30)
```


abs(f.full$coef[−1])
0.00
0.04
0.08
0.12
0
50
100
200
Time
sqrt(diag(var(X))) * abs(f.full$coef[−1])
0.000
0.015
0.030
0
50
100
200
Time


Histogram of f.full$coef[−1]
stogram of sqrt(diag(var(X))) * f.full$cc
Frequency
0
5
15
25
−0.15
−0.05
0.05
f.full$coef[−1]
Frequency
0
5
10
15
20
−0.03
−0.01
0.01
0.03
sqrt(diag(var(X))) * f.full$coef[−1]