Class Prediction from Omics Data

BST 226
Statistical Methods for Bioinformatics
David M. Rocke

Class prediction from omics data

- One common use of omics data is to try to develop predictions for classes of patients, such as
  - cancer/normal
  - type of tumor
  - grading or staging of tumors
  - many other disease/healthy or diagnosis of disease type

Two-class prediction

- Linear regression
- Logistic regression
- Linear or quadratic discriminant analysis
- Partial least squares
- Fuzzy neural nets estimated by genetic algorithms and other buzzwords
- Many such methods require fewer variables than cases, so dimension reduction is needed

Dimension Reduction

- Suppose we have 20,000 variables and wish to predict whether a patient has ovarian cancer or not and suppose we have 50 cases and 50 controls
- We can only use a number of predictors much smaller than 50
- How do we do this?

- Two distinct ways are selection of genes and selection of “supergenes” as linear combinations
- We can choose the genes with the most significant t-tests or other individual gene criteria
- We can use forward stepwise logistic regression, which adds the most significant gene, then the most significant addition, and so on, or other ways of picking the best subset of genes

Supergenes are linear combinations of genes. If g_1, g_2, g_3, …, g_p are the expression measurements for the p genes in an array, and a_1, a_2, a_3, …, a_p are a set of coefficients then $g_1 a_1 + g_2 a_2 + g_3 a_3 + \ldots + g_p a_p$ is a supergene. Methods for construction of supergenes include PCA and PLS

Choosing Subsets of Genes

- Suppose we have 50 cases and 50 controls and an array of 20,000 gene expression values for each of the 100 observations
- In general, any arbitrary set of 100 genes will be able to predict perfectly in the data if a logistic regression is fit to the 100 genes
- Most of these will predict poorly in future samples

- This is a mathematical fact
- A statistical fact is that even if there is no association at all between any gene and the disease, often a few genes will produce apparently excellent results, that will not generalize at all
- We must somehow account for this, and cross validation is the usual way

```
source(“spuriousprediction.r”)
y <- rep(0:1,each=50)
x <- matrix(rnorm(100*20000),ncol=100)
ts <- vector("numeric",20000)
for (i in 1:20000)
{
  ts[i] <- (t.test(x[i,] ~ y)$statistic)^2
}
ind <- order(ts,decreasing=T)
```

```
ind <- order(ts,decreasing=T)
> source("spuriousprediction2.r")
sp.glm <- glm(y ~ x[ind[1],],binomial)
print(summary(sp.glm))
yp <- predict.glm(sp.glm,type="response")
yp[yp < 0.5] <- 0
yp[yp >= 0.5] <- 1
print("Number of Misclassifications out of 100")
print(sum(y != yp))

sp.glm <- glm(y ~ x[ind[1],],binomial)
yp <- predict.glm(sp.glm,type="response")
yp[yp < 0.5] <- 0
yp[yp >= 0.5] <- 1
print("Number of variables/Misclassifications out of 100")
print(c(1,sum(y != yp)))

sp.glm <- glm(y ~ x[ind[1],]+x[ind[2],],binomial)
yp <- predict.glm(sp.glm,type="response")
yp[yp < 0.5] <- 0
yp[yp >= 0.5] <- 1
print("Number of variables/Misclassifications out of 100")
print(c(2,sum(y != yp)))
```

```
> source("spuriousprediction2.r")

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.96156  -1.07483   0.08347   0.99583   1.68009

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.03078    0.22122  -0.139 0.889342
x[ind[1], ] -1.15034    0.30385  -3.786 0.000153 ***

    Null deviance: 138.63  on 99  degrees of freedom
Residual deviance: 119.00  on 98  degrees of freedom
AIC: 123.00

Number of Fisher Scoring iterations: 4

[1] "Number of Misclassifications out of 100"
[1] 36
```

```
[1] "Number of variables/Misclassifications out of 100"
[1]  1 36
[1] "Number of variables/Misclassifications out of 100"
[1]  2 32
[1] "Number of variables/Misclassifications out of 100"
[1]  3 27
[1] "Number of variables/Misclassifications out of 100"
[1]  4 19
[1] "Number of variables/Misclassifications out of 100"
[1]  5 17
[1] "Number of variables/Misclassifications out of 100"
[1]  6 21
[1] "Number of variables/Misclassifications out of 100"
[1]  7 16
[1] "Number of variables/Misclassifications out of 100"
[1] 20  0
Warning messages:
1: Algorithm did not converge in: glm.fit(x = X, y = Y,
  weights = weights, start = start, etastart = etastart,
2: fitted probabilities numerically 0 or 1 occurred in:
glm.fit(x = X, y = Y, weights = weights, start = start,
etastart = etastart,
```

Now with the first 20 variables instead of the 20/20000 with the Biggest t-scores:

```
[1] "Number of variables/Misclassifications out of 100"
[1] 20 26

Call:
glm(formula = y ~ x[1, ] + x[2, ] + x[3, ] + x[4, ] + x[5, ] +
    x[6, ] + x[7, ] + x[8, ] + x[9, ] + x[10, ] + x[11, ] + x[12,
    ] + x[13, ] + x[14, ] + x[15, ] + x[16, ] + x[17, ] + x[18,
    ] + x[19, ] + x[20, ], family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.20702  -0.89041   0.01297   0.92103   1.90446
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.06041    0.24533   0.246   0.8055
x[1, ]      -0.43297    0.30242  -1.432   0.1522
x[2, ]       0.60087    0.28979   2.074   0.0381 *
x[3, ]       0.11777    0.23215   0.507   0.6119
x[4, ]       0.22212    0.24727   0.898   0.3690
x[5, ]      -0.15468    0.26043  -0.594   0.5526
x[6, ]       0.31370    0.24938   1.258   0.2084
x[7, ]      -0.43456    0.30462  -1.427   0.1537
x[8, ]      -0.41751    0.29113  -1.434   0.1515
x[9, ]      -0.45591    0.29228  -1.560   0.1188
x[10, ]      0.50699    0.28279   1.793   0.0730 .
x[11, ]     -0.54391    0.27250  -1.996   0.0459 *
x[12, ]      0.38480    0.26215   1.468   0.1422
x[13, ]     -0.04257    0.24281  -0.175   0.8608
x[14, ]      0.13996    0.25947   0.539   0.5896
x[15, ]      0.41957    0.23650   1.774   0.0761 .
x[16, ]     -0.20779    0.29312  -0.709   0.4784
x[17, ]      0.57632    0.30106   1.914   0.0556 .
x[18, ]      0.02833    0.27818   0.102   0.9189
x[19, ]      0.25862    0.25417   1.018   0.3089
x[20, ]      0.45244    0.23562   1.920   0.0548 .
```

```
Null deviance: 138.63  on 99  degrees of freedom
Residual deviance: 112.35  on 79  degrees of freedom
--------------------------------------------------
(138.63 - 112.35) = 26.28 ~ chisq(20) p ~ .32
--------------------------------------------------
```

	Df	Deviance	Resid. Df	Resid. Dev	P(>|Chi|)
NULL			99	138.629	
x[1,]	1	0.467	98	138.163	0.494
x[2,]	1	1.376	97	136.787	0.241
x[3,]	1	0.217	96	136.570	0.641
x[4,]	1	0.135	95	136.435	0.713
x[5,]	1	0.962	94	135.473	0.327
x[6,]	1	0.603	93	134.870	0.437
x[7,]	1	1.622	92	133.248	0.203
x[8,]	1	0.575	91	132.672	0.448
x[9,]	1	0.574	90	132.099	0.449
x[10,]	1	1.509	89	130.589	0.219
x[11,]	1	2.262	88	128.327	0.133
x[12,]	1	1.557	87	126.771	0.212
x[13,]	1	0.006	86	126.764	0.937
x[14,]	1	0.598	85	126.166	0.439
x[15,]	1	2.902	84	123.264	0.088
x[16,]	1	0.328	83	122.936	0.567
x[17,]	1	5.015	82	117.921	0.025
x[18,]	1	0.011	81	117.909	0.916
x[19,]	1	1.704	80	116.205	0.192
x[20,]	1	3.855	79	112.350	0.050

Consequences of many variables

- If there is no effect of any variable on the classification, it is still the case that the number of cases correctly classified increases in the sample that was used to derive the classifier as the number of variables increases
- But the statistical significance is usually not there

- If the variables used are selected from many, the apparent statistical significance and the apparent success in classification is greatly inflated, causing end-stage delusionary behavior in the investigator
- This problem can be improved using cross validation or other resampling methods

Overfitting

- When we fit a statistical model to data, we adjust the parameters so that the fit is as good as possible and the errors are as small as possible
- Once we have done so, the model may fit well, but we don't have an unbiased estimate of how well it fits if we use the same data to assess as to fit

Training and Test Data

- One way to approach this problem is to fit the model on one dataset (say half the data) and assess the fit on another
- This avoids bias but is inefficient, since we can only use perhaps half the data for fitting
- We can get more by doing this twice in which each half serves as the training set once and the test set once
- This is two-fold cross validation

- It may be more efficient to use 5- 10-, or 20-fold cross validation depending on the size of the data set
- Leave-out-one cross validation is also popular, especially with small data sets
- With 10-fold CV, one can divide the set into 10 parts, pick random subsets of size 1/10, or repeatedly divide the data

```
ind <- order(ts,decreasing=T)

n.tot <- 0
n.wrong <- 0

for (i in 1:100)
{
  test.set.list <- sample(100,10)
  test.seti <- rep(F,100)
  test.seti[test.set.list] <- T
  train.seti <- !test.seti
  y1 <- y[train.seti]
  x1 <- x[ind[1],train.seti]
  sp.glm <- glm( y1 ~ x1,binomial)
  yp <- predict.glm(sp.glm,data.frame(x1=x[ind[1],test.seti]),type="response")
  yp[yp < 0.5] <- 0
  yp[yp >= 0.5] <- 1
  n.tot <- n.tot+10
  n.wrong <- n.wrong+sum(y[test.seti] != yp)
}
```

```
print("Number of variables/Misclassifications out of 1000")
print(c(1,n.wrong,n.tot,100*n.wrong/n.tot))

> source("spuriousprediction3.r")
[1] "Number of variables/Misclassifications out of 1000"
[1]    1.0  363.0 1000.0   36.3


Cf. missclass within the 100 for this variable was 36
It should have been about 50 since the predictors are random
Cross validation does not solve the problem if the whole data
Set was used to find the variable(s)
```

Stepwise Logistic Regression

- Another way to select variables is stepwise
- This can be better than individual variable selection, which may choose many highly correlated predictors that are redundent
- A generic function step() can be used for many kinds of predictor functions in R

Using step()

- step(glm.model) is sufficient
- It uses steps either backward (using drop1) or forward (using add1) until a model is reached that cannot be improved
- Criterion is AIC = Akaiki Information Criterion, which tries to account for the effect of extra variables, more so than MSE or R^2

- You may also specify a scope in the form of a list(lower=model1, upper =model2)
- For expression arrays, with thousands of variables one should start with y ~ 1 and use scope =list(lower=y~1, upper=**)

```
for (i in 1:100)
{
  assign(paste("x",i,sep=""),x[ind[i],])
}
fchar <- "y~x1"
for (i in 2:100)
{
  fchar <- paste(fchar,"+x",i,sep="")
}
form <- as.formula(fchar)
step(glm(y ~ 1),list(lower=(y~1),upper=form))
```

assign creates a variable with a name and a value

paste makes a character string by pasting together parts

The first loop creates variables x1 to x100

The second loop creates a formula of the form

```
y~x1+x2+x3+…+x100
```

```
Step:  AIC= -288.12
 y ~ x29 + x13 + x60 + x17 + x47 + x3 + x50 + x30 + x26 + x16 +
    x78 + x9 + x37 + x89 + x52 + x6 + x46 + x75 + x83 + x62 +
    x28 + x14 + x98 + x22 + x8 + x56 + x81 + x53 + x65 + x5 +
    x23 + x27 + x44 + x99 + x90 + x92 + x93 + x71 + x70 + x40 +
    x10 + x77 + x20 + x15 + x4 + x33 + x61 + x25 + x68 + x35 +
    x67 + x55 + x96 + x19 + x87 + x39 + x42 + x64 + x100 + x94 +
    x18 + x63 + x2 + x11 + x86 + x7 + x12 + x57 + x24 + x80 +
    x31 + x32 + x21 + x51 + x49 + x72 + x58 + x41 + x69 + x36
```

Given that there is no variable here actually related to the Response, this cannot be said to have done very well. Partly The problem is that we started with the 100 accidentally highest t-scores

Conclusions

- Predicting an outcome from a set of variables many times the size of the number of observations is hazardous
- Cross validation or something similar is the only way to have any chance of integrity.
- Nothing can be done to the data before cross validation that uses both the class labels and the predictors.
- So we can eliminate variables all of whose values are too small
- But we cannot choose variables that predict well from the whole data set.