GS01 0163
Analysis of Microarray Data

Keith Baggerly and Bradley Broom
Department of Bioinformatics and Computational Biology
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org
bmbroom@mdanderson.org

28 October 2010

Lecture 17: Classification with Microarrays II

- Genetic Algorithms
  - A simple example
  - Application to proteomics
  - Reliability
- Feature Selection
  - A tale of two studies
  - Gleason grade and Gleason score
  - Combining the data from two studies
  - Logistic Regression

Genetic Algorithms (GAs)

Yet one more method for classification.

Ironically (for this course), GAs do not involve specific genes. Rather, the key idea is to develop a good classifier through *evolution*. This process is assumed to mimic the way in which genes evolved and gained functionality.

GAs work by specifying an objective function (such as the fraction of samples correctly classified), and trying combinations of elements (logical chromosomes) to see how well they optimize the objective.

A Simple Example

Say we want to maximize $f(x) = x^2(1-x)^3$ over the interval $[0, 1]$. We can of course solve for this analytically; the maximum value of 0.03456 is attained when $x = 0.4$. But this is a toy problem.

We want to represent the number x in terms of binary pieces, as a "logical chromosome".

Say we start with a sequence of 30 zeros and ones:

00101 10101 01010 10011 11011 01011

Treating this as a binary integer divided by 2^{30} gives 0.177089.

Testing the Fit

Here, our logical chromosome has "fitness"

$$f(0.177089) = 0.01747504.$$

This is just one random string. Let's say we generate 100 such strings. Then we can compute the fitness for each string.

Assembling "Chromosomes"

```
n.strings <- 100;
n.units <- 30;
startgen <-
  matrix(round(runif(n.strings*n.units)),
         n.strings,n.units);
x.weights <- 2^(-1:-n.units);
x.vals <- startgen %*% x.weights;
x.fitness <- x.vals^2 * (1 - x.vals)^3;
```

a brute-force way to generate some binary strings.

For Example...

```
> startgen[1,]
 [1] 11011 11010 11000 00111 00001 00111
> startgen[2,]
 [1] 01101 01000 00101 00000 00101 00010
> startgen[3,]
 [1] 11010 00100 00100 11101 00100 11110

> x.vals[1:3]
[1] 0.8698798 0.4142152 0.8165561

> x.fitness[1:3]
[1] 0.001667067 0.034487868 0.004116060
```

How do we Evolve?

At each generation, we pick pairs of elements to “breed”, with the chance of being selected being linked to the overall fitness in some way.

Given a pair, we line them up, and let them “cross over” at some randomly chosen location. So

11011 11010 11000 00111 00001 00111
01101 01000 00101 00000 00101 00010

might produce

11011 11010 11000 00000 00101 00010

Other Tweaks

Crossover is thought to be the main evolutionary driver over several generations. However, there is typically also a small chance of something new getting introduced via mutation – with a small probability, a random element of the logical chromosome will be “flipped”. So

11011 11010 11000 00111 00001 00111

might produce

11011 11010 11000 10111 00001 00111

Note the 1 in the middle!

Trying it here

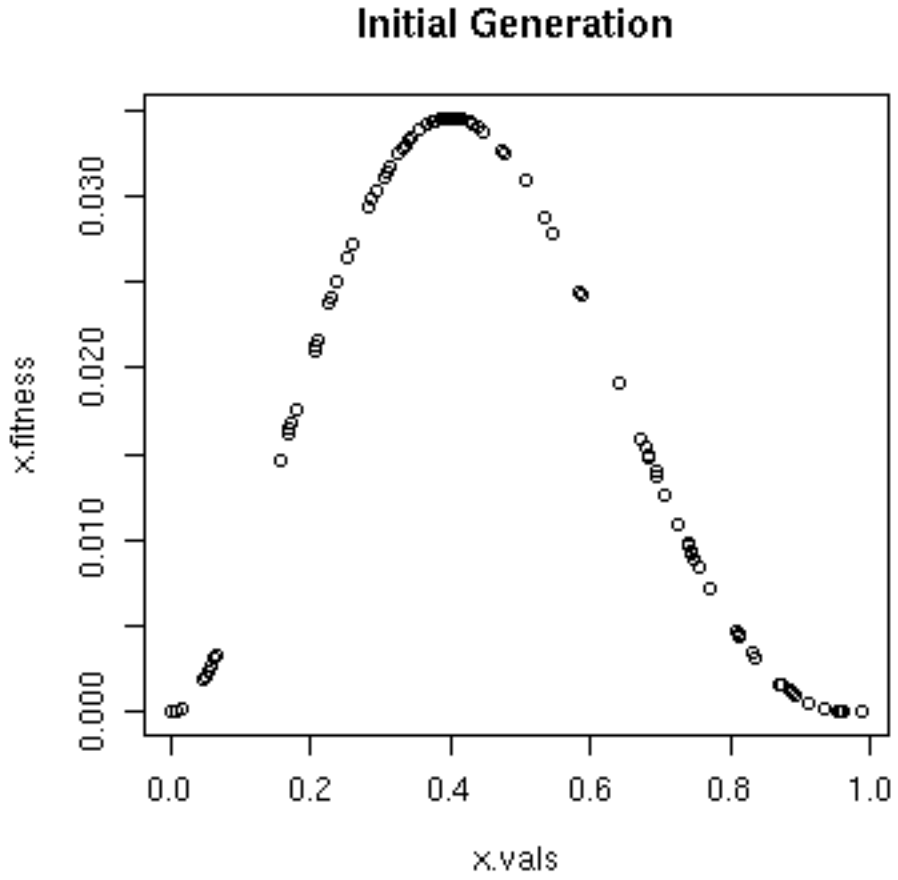
I chose a population of 100 strings of length 30.

The chance of being selected as a member of a pair for forming a next generation individual was proportional to

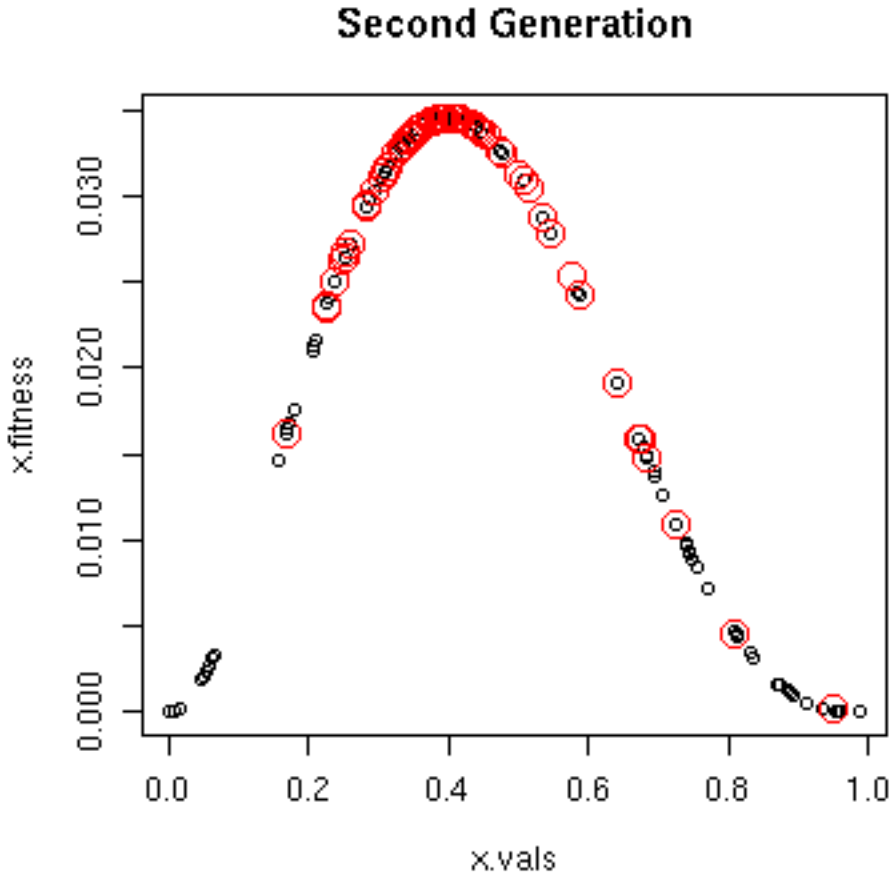
$$\left(\frac{fit(i) - \min(fit)}{\max(fit) - \min(fit)} \right)^2$$

I didn't bother with mutation, and I let things go for 10 generations.

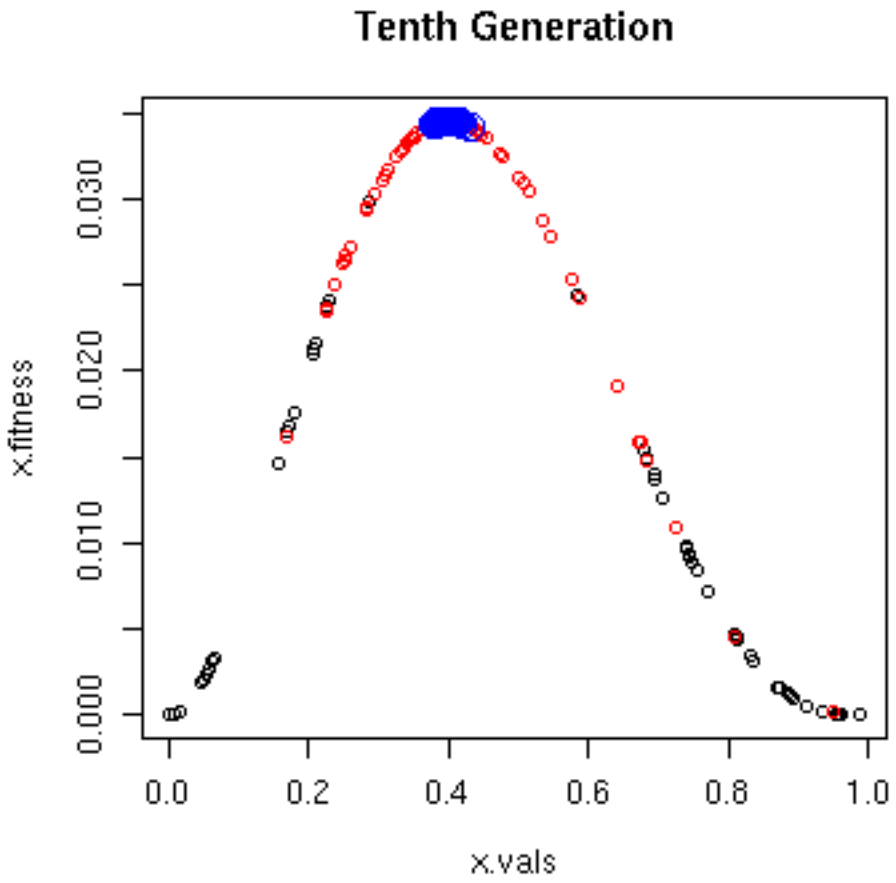
GA Fits



GA Fits



GA Fits



What Does This Gain Us?

A GA is a stochastic (random) search method. It tries lots of combinations of things, including some that might not occur to us. It has the potential of including some types of interactions that might not have otherwise been spotted.

The goal is to efficiently explore a high-dimensional space.

Coupled with Arrays

In the microarray context, the logical chromosome can be used to indicate the presence or absence of a gene.

Alternatively, if we want to work with just a small number of genes (say 5), the individual "chromosomes" can be lists of 5 index values.

For each chromosome, we can compute the overall cross-validated classification accuracy using LDA, or the distance between the group centers after standardizing.

An Application to Proteomics

Proteomic spectra yield peaks at specific m/z (loosely mass) values. Different mass peaks ideally correspond to different proteins.

We started with a 506 by 41 matrix of peak intensities (log transformed) – 506 m/z values, and 41 samples: 24 from patients with lung cancer, and 17 controls.

We then looked for good sets of 1 through 5 peaks.

What we Did

We used the square of the Mahalanobis distance as our fitness function:

$$MD = (\bar{x}_1 - \bar{x}_2)^T S^{-1} (\bar{x}_1 - \bar{x}_2)$$

This is the multivariate generalization of the two sample t-test. (As before, S is the covariance matrix.) It works with the distances between group centers, and is scale invariant.

Searching the Space

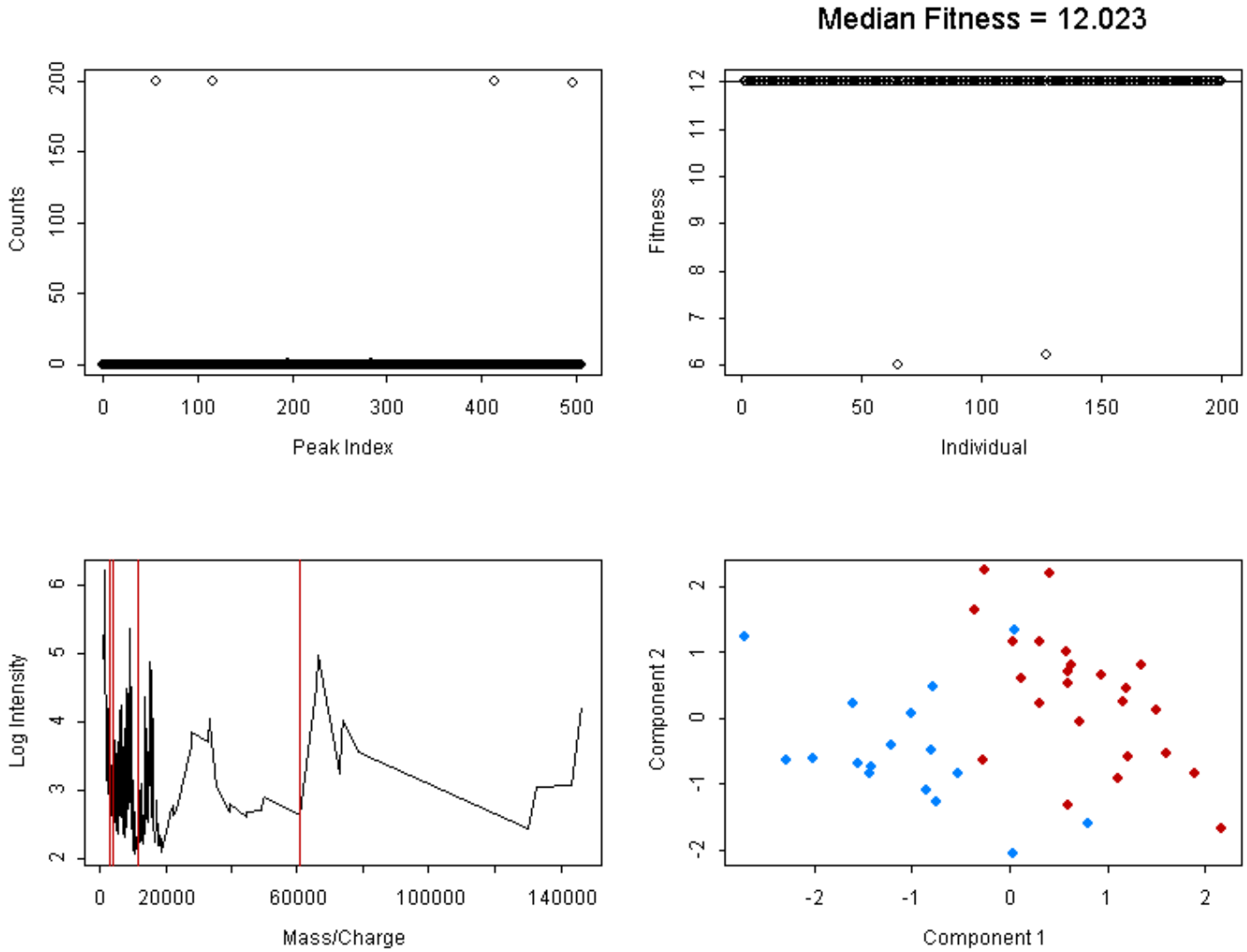
How we searched

- For 1 and 2 peaks, exhaustive search.
- For sets of 3, 4 and 5 peaks, use a Genetic Algorithm (GA).

Some GA details

- 200 logical chromosomes/run, 250 generations.
- 50 different random starts (3,4,5).
- Every run of the GA converged.

A Typical Solution: Best 4



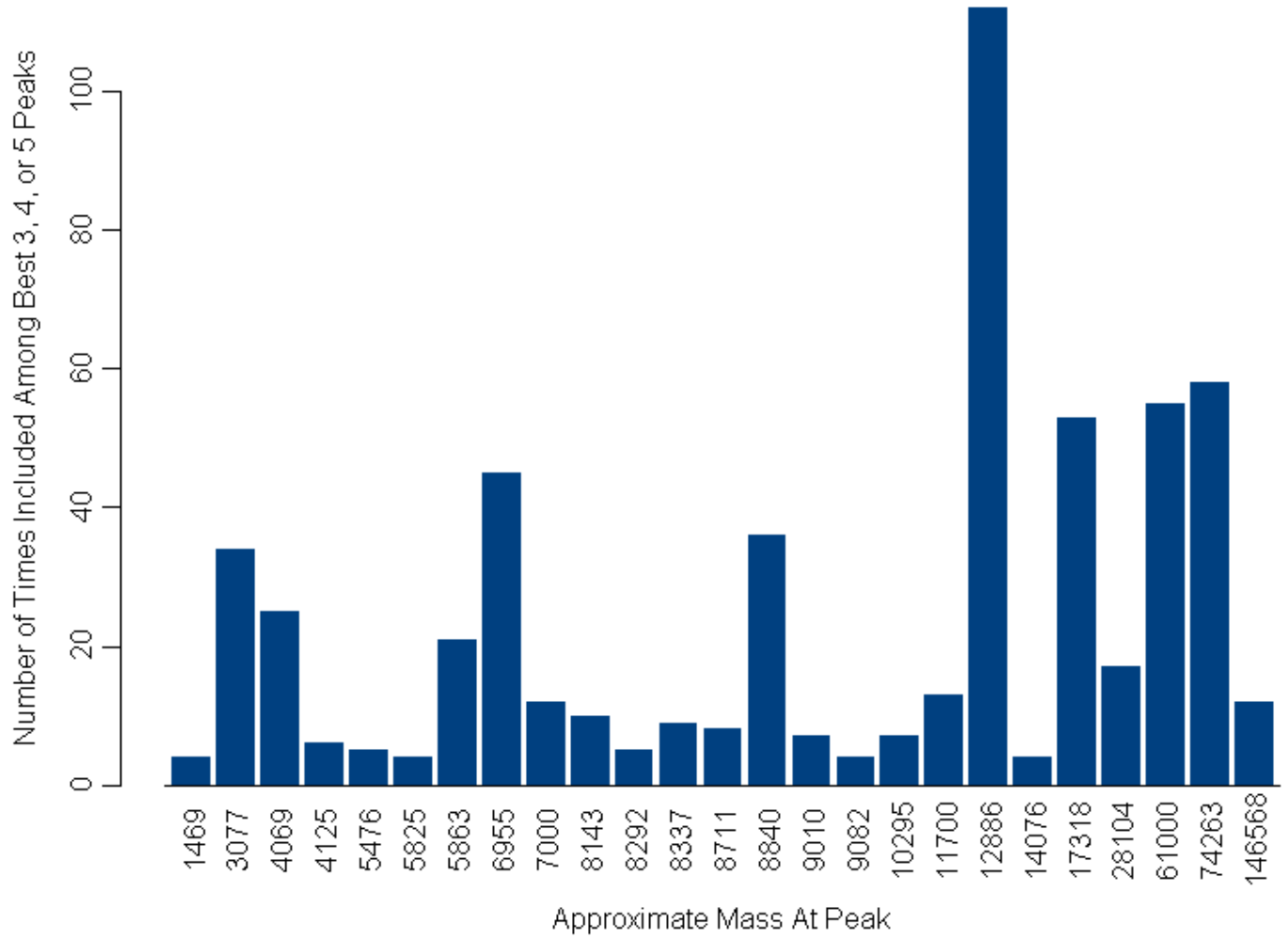
Find the Best Separators

Peaks	MD	P-Value	Wrong	LOOCV
12886	2.547	≤ 0.005	11	11
8840, 12886	5.679	≤ 0.01	5	6
3077, 12886 74263	9.019	≤ 0.01	3	4
5863, 8143 8840, 12886	12.585	≤ 0.01	3	3
4125, 7000 9010, 12886 74263	23.108	≤ 0.01	1	1

Remember the Randomness!

- GAs are a form of “directed random search”.
- Because of randomness in the algorithm, we can get different answers every time (and we do).
- Because there is no unique solution, we need to verify that the values we find are worth paying attention to.

How Often Did We Find the Best Peaks?



Survey the Results

There are 9 values that recur frequently, at masses of 3077, 4069, 5825, 6955, 8840, 12886, 17318, 61000, and 74263.

P-values are not from table lookups!

Simulating Significance

We weren't sure when a Mahalanobis distance would be "big" here, so we repeated the procedure with randomly generated data – noise matrices of size 506 by 41, and recomputing the MD values after evolving.

This is a slightly different use of simulation; earlier we used simulation to assess whether we were using cross-validation correctly, and here we're using it to assess whether the values we got were big.

In the end, the GA let us look for interactions, but many of our final recommendaions were based on univariate behavior.

Feature Selection

Most classification methods involve three steps:

1. Feature Selection
2. Model Construction
3. Model Validation

Our GA combines feature selection with model construction. CART also combines these steps. By contrast, most other methods are applied only after a separate feature selection step.

In many respects, the choice of classifier (LDA, KNN, SVM, etc) is less important than how the features are selected.

A tale of two studies

We start by reviewing two published microarray studies of prostate cancer.

Reference: Lapointe et al. Gene expression profiling identifies clinically relevant subtypes of prostate cancer. *Proc Natl Acad Sci USA*. 2004; **101**: 811–816.

- 62 samples of prostate cancer
- 41 matched samples of normal prostate
- 9 samples of lymph node metastases from prostate cancer

Lapointe's gene filtering

This paper uses two-color microarrays produced at Stanford. They were processed with local background subtraction, global (mean) normalization, then taking log ratios with the reference channel.

After preprocessing the data, they filtered the genes in two steps:

- Intensity filter (intensity > 1.5-fold above background in both experimental samples and reference samples in at least 75% of experiments)
- Variation filter (≥ 3-fold variation from the mean in at least two samples)

Lapointe's clustering

Next, they performed hierarchical clustering. No information was supplied on the distance measure nor on the linkage rule. No validation of clusters was performed.

Claimed result: Can distinguish normals from tumors with two exceptions. (The two tumors clustered with normals are both unusual.) They also claim to find three subclusters of the tumor population.

Lapointe's tumor subtypes

Continuing with their analysis, they looked at how clinical features matched the putative subtypes. One of the three subtypes (subtype III) contained 8 of the 9 lymph node metastases.

Interestingly, they pooled subtypes II and III before performing chi-squared tests to look at how clinical information matched the subtypes. Based on these chi-squared tests, higher grade tumors and advanced stage tumors were more likely to be clustered in combined group II–III than in group I.

Lapointe's study of differential expression

Next, they looked for genes that were differentially expressed between various subgroups of cancer. Differentially expressed genes were selected using Significance Analysis of Microarrays (SAM). They compared tumors based on:

- Gleason grade ($\leq 3+4$ vs. $\geq 4+3$), finding 41 genes
- Stage ($\leq T2$ vs. $\geq T3$), finding 11 genes
- Recurrence, finding 23 genes

A second study

Reference: Singh et al. Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell*. 2002; **1**: 203–209.

- 52 samples of prostate cancer
- 50 samples of apparently normal prostate

This study used Affymetrix U95Av2 oligonucleotide arrays. They were quantified using MAS4.0, and genes that varied less than 5-fold across the sample set were removed from consideration.

Singh's study of differential expression

Genes were ranked according to their differential expression between tumor and normal using a variant of the t-statistic (with a non-standard estimate of the pooled standard deviation). Significance was determined using a permutation test, and 317 genes were found to be up-regulated in prostate cancer, along with 139 genes that were down-regulated.

Singh's prediction of tumor status

Next, they built a classifier to predict tumor vs. normal using k-nearest neighbors. (The value of k is not specified in the paper.) Leave-one-out cross-validation was performed. (Feature selection by ranking genes was included in the cross-validation.) They constructed 4-gene and 16-gene models; the 4-gene model had a leave-one-out cross-validation accuracy of 90%.

Singh's investigation of clinical correlates

They looked for differential expression with respect to a number of clinical factors; the only one that appeared significant was Gleason grade, where they found 29 differentially expressed genes.

They also tried to predict recurrence and survival, with limited success.

Gleason grade and Gleason score

Biopsies of prostate cancer are graded by pathologists using a scale developed by Gleason. The grade is based on the appearance of the tumor specimen under a microscope. An individual tumor focus is graded on a scale from 1 to 5, with 1 being essentially normal in appearance. The severity of the tumor increases (and it appears less differentiated) as the grade increases.

Gleason grade and Gleason score, cont.

In general, prostate cancer appears in multiple distinct foci.

The two most abundant foci are graded separately, and the Gleason score is often summarized by adding the two grades.

A patient with prostate cancer can get a Gleason score of 7 represented as a $4 + 3$, which means that the largest tumor focus is grade 4 and the secondary tumor focus if grade 3.

This may well represent a worse tumor than a Gleason 7 that arises from a $3 + 4$.

Gleason score and prognosis

In general, patients with a Gleason score of 6 or lower have a good prognosis and a low risk of recurrence after surgery. Standard clinical practice for these patients is to "watch-and-wait".

By contrast, patients with a Gleason score of 8 or higher have a poor prognosis and a high risk of recurrence after surgery. These patients are usually treated witrh chemotherapy, which is often effective in preventing recurrence.

Gleason score and prognosis (hard)

Patients with a Gleason score of 7 have an intermediate risk, and it is unclear how best to proceed.

GOAL: Find a model that can predict whether a patient has good prognosis (defined as Gleason 6 or lower) or poor prognosis (Gleason 8 or higher), and then see what this model tells us about patients with Gleason 7 prostate cancer.

Combining the data from two studies

We get a much larger sample size if we combine the data:

Study	Low	Medium	High	Unknown	Total
Singh	19	29	4	0	52
Lapointe	24	22	15	1	62
Total	43	51	19	1	114

Re-processing

Both studies need to be pre-processed differently.

The Singh study used the old MAS4.0 algorithm, which gives inferior estimates of gene expression. We reprocessed the CEL files using the PM-only model in dChip version 1.3, which should do reasonably well with 52 cancer and 50 normal samples. After processing, we computed the log intensities.

The Lapointe study used global normalization. The M-vs-A plots suggest here that a loess normalization would be more appropriate, and so we applied loess before computing log ratios. (Is it safe to assume that most genes don't change here?)

Gene joining

A critical task when combing two studies from different platforms is to determine which genes are measured on both platforms.

Using the GenBank identifiers and the Affymetrix probe-set identifiers, we updated the annotations on both platforms to use UniGene build 170 (July 2004). Probes targeting the same UniGene cluster were assumed to be measuring the same gene on both platforms.

We found 8054 probe-sets on the Affymetrix U95A that matched with 11,596 clones on the Stanford glass microarrays, representing 6204 distinct UniGene clusters.

Everything is relative

At any rate, all microarray measurements are relative.

So, we can't directly combine glass array data (represented as log ratios to a reference sample) to Affymetrix data (represented as log intensities using different probes).

Although we're interested in comparing cancer subtypes, we have measurements of normal prostate on each platform. Thus, we can adjust the measurements on both platforms to be relative to the same thing.

Standardized gene expression

On each platform, we define

$$S_i = \frac{X_i - \mu_{normal}}{\sigma_{normal}}.$$

Here X_i represents the vector of gene measurements in sample i. We estimate μ and σ from the normal prostate samples on each platform, and we standardize the measurements so that the normals have mean 0 and sd 1. We refer to the S_is as standardized gene expression measurements.

Note: There are multiple probes for many UniGene clusters within each platform. After standardizing, we average these measurements and then re-standardize.

Catching our breath

At this point, we can combine the prostate cancer data from the two platforms. We have a data matrix of size 6204×62 from the Lapointe study that includes 62 cancer samples, and a data matrix of size 6204×52 from the Singh study that includes 52 cancer samples. We also know the Gleason score of all 114 cancer samples.

How do we build a model to predict Gleason score from gene expression?

Logistic Regression

Our goal is to predict “good” or “poor” prognosis in terms of gene expression, where “good” means Gleason 6 or lower and “poor” means Gleason 8 or higher. So, we actually have a binary outcome variable to predict. Numerically, we can code this outcome as a value of 0 (good) or 1 (poor).

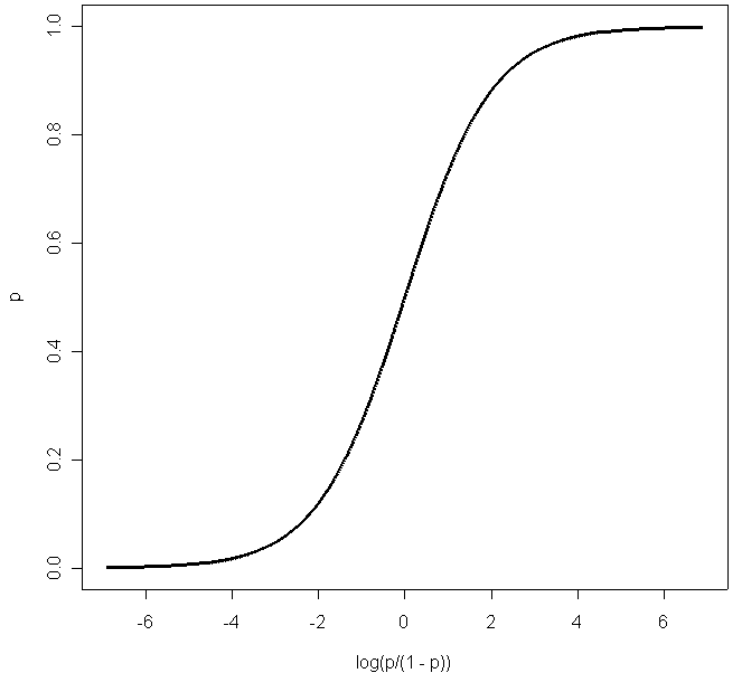
A favorite statistical tool for a wide variety of problems is to build a linear model. It’s hard, however, to get linear functions to restrict their output values to 0’s and 1’s.

So, we first generalize and think about the outcome as the probability of a poor outcome. That helps a little bit, since probabilities can be any continuous value between 0 and 1. But there is still the problem that linear functions will, sooner or later, extend outside of any fixed interval.

Logistic functions, or the logit transformation

Next, we transform the outcome so it spreads out across the entire range of real numbers:

$$y = log(\frac{p}{1-p}).$$



Almost there

Now given any set of covariates X_1, X_2, ..., X_n, we can build a predictive linear model of the form

$$y = log(\frac{p}{1-p}) = X_1\beta_1 + \cdots X_n\beta_j.$$

There is, of course, one minor technical glitch here. The value $p = 0$ corresponding to an observed "good" prognosis gets mapped to negative infinity, and the value $p = 1$ corresponding to an observed "poor" prognosis gets mapped to positive infinity. This difficulty prevents us from using the standard techniques to fit the model from the data. Instead, we have to resort to using iterative methods to find maximum likelihood estimates.

Logistic regression in R

Fortunately, we don't have to concern ourselves with the details of finding the maximum likelihood estimates, since the iterative procedure is already coded in an R function. Let's assume we have put together a data frame (`my.data`) containing three columns:

X The standardized expression levels of a single gene for all samples

G A binary indicator of Gleason score (0 = good, 1 = poor)

S A binary indicator of the study (0 = Lapointe, 1 = Singh)

Logistic regression in R

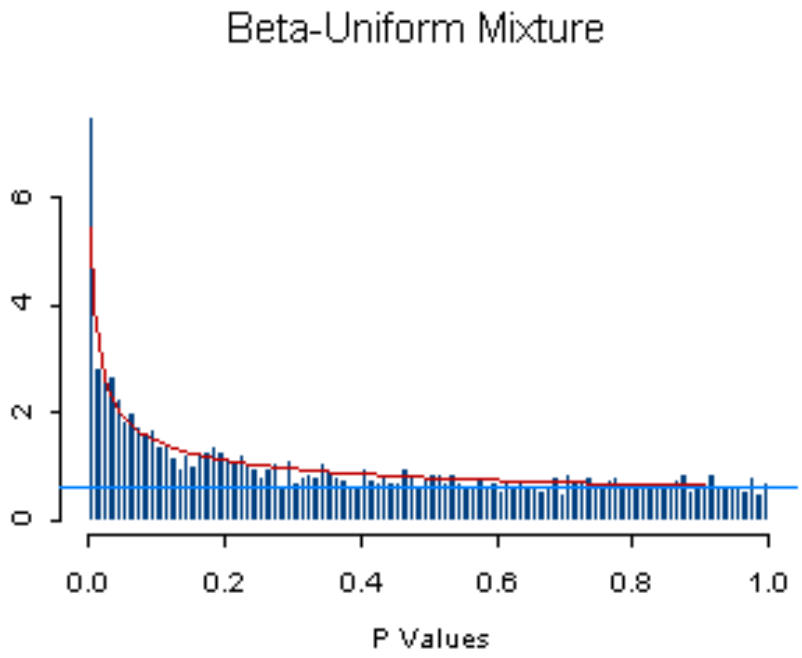
Then we can fit a logistic regression model that predicts Gleason score from gene expression, allowing for a study effect, by

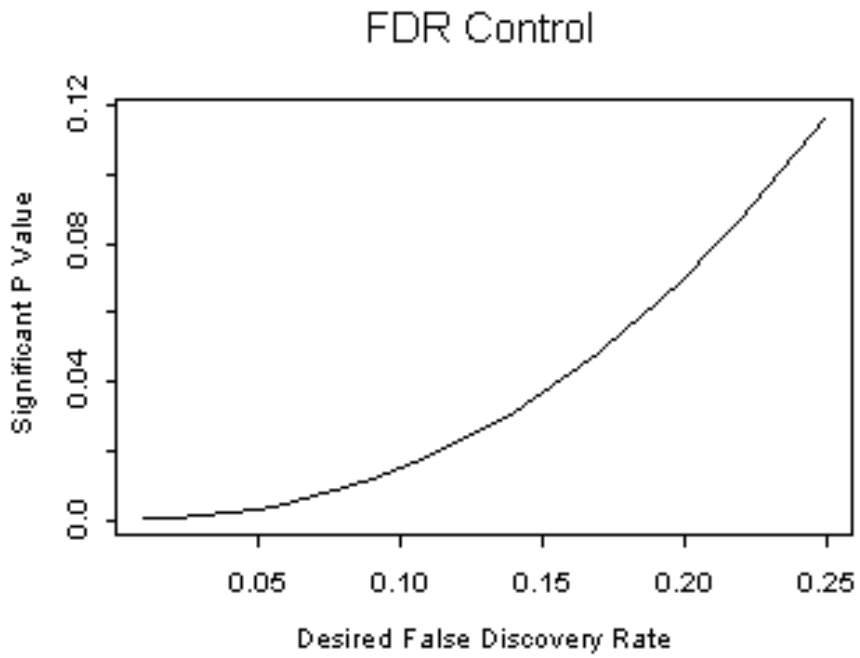
```
> logreg <- glm(G ~ S + X, data = my.data,
+   family = binomial)
```

A p-value associated with this model tells us how well it describes the data; separate p-values can be computed for the significance of the study effect or of the gene expression as a contributor to the prediction of the Gleason score. (Note: study had no effect on Gleason score.) We computed logistic regression models for each of the 6402 UniGene clusters measured on both studies, and computed a p-value for the significance of each model.

BUM and logistic regression

To account for multiple testing, we modeled the distribution of all the p-values using a beta-uniform mixture (BUM), the same method we used for differential expression.





Combining multiple genes in a single model

So, we now have some evidence that there are genes that make a nontrivial contribution to our ability to predict Gleason score. How do we put together a single model that combines information across genes?

First, we arbitrarily restricted ourselves to the 20 genes that had the most significant p-values from the logistic regression models using only one gene at a time. We put together a data frame (`top.twenty`) that combined the Gleason score, the study effect, and the standardized expression values of the 20 genes. We also constructed a logistic regression model that included all twenty genes:

```
> logreg <- glm(G ~ ., data = top.twenty,
+   family = binomial)
```

Akaike information criterion

Nested models using different explanatory variables can be compared using the Akaike Information Criterion:

$$AIC = -2 * (\text{max log likelihood}) + 2 * (\text{No. parameters}).$$

A smaller value of the AIC is better. The AIC uses the number of parameters as a penalty term. If two models explain the data equally well, then the model with fewer parameters (or fewer explanatory variables) is preferred.

R includes an automated procedure to discard genes that are not contributing to the model, based on the AIC:

```
> best.model <- step(logreg)
```

The seven-gene predictor

We ran this procedure on the combined data to predict Gleason good (6 or lower) or Gleason poor (8 or higher).

- We got a seven-gene model:

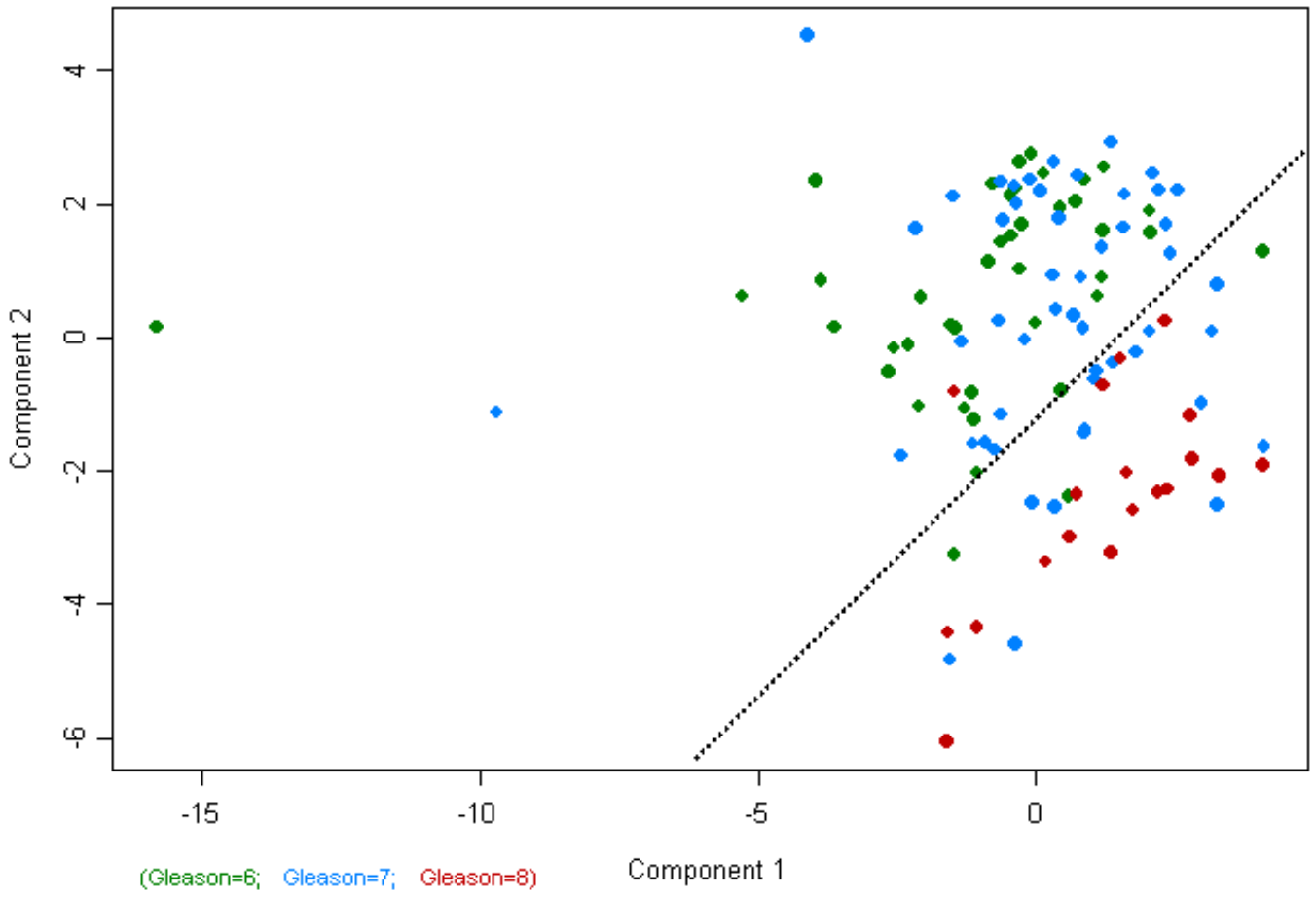
  LTBP2 latent transforming growth factor beta binding protein 2
  TIMP2 tissue inhibitor of metalloproteinase 2
  CDH11 cadherin 11, type 2, OB-cadherin (osteoblast)
  RAP140 retinoblastoma-associated protein 140
  ProSAPiP2 ProSAPiP2 protein
  CXCR4 chemokine (C-X-C motif) receptor 4
  SEPT6 Septin 6

The seven-gene predictor

Do we know anything about the seven?

Three of these genes (TIMP2, CDH11, CXCR4) have been previously reported to be related to prognosis in prostate cancer.

PCA based on seven genes



Is this reproducible?

We repeated the logistic regression procedure with stepwise forward selection, using leave-one-out cross-validation method to estimate the misclassification error rate.

Method	No. Features Selected	LOOCV Error Rate
Logistic Regression - Forward Stepwise	4–9	31%
Greedy - LDA	10–20	31%
Genetic Algorithm - LDA	10	28%
PCA - LDA	10–20	24%
Robust FS - LDA	5–11	31%

Robust Feature Selection

The high error rates suggest that we are overfitting the data. We didn't do any better using a "robust" method for feature selection similar to the proteomics example.

- Robust Feature Selection
  1. Combine feature selection and model construction on each leave-one-out sample.
  2. Count the number of times a feature is selected overall
  3. Only use the features that are selected many times for the final model.

This method is then evaluated using another layer of leave-one-out. Since it still did poorly, we looked at what happens when we use fewer features.

Less is more

Number of Features	Misclassification Rates (%) from Cross-Validation			
	Greedy-LDA	GA-LDA	PCA-LDA	Robust - LDA
1	33	32	27	19
2	24	26	27	15
3	31	19	23	21
4	26	24	22	24
5	29	21	19	29
6	29	31	19	29
7	26	24	19	31