Principal Components

- Suppose we have a data matrix, **X**, of *n* independent samples and *p* features. The sample covariance matrix of **X** is **S**.
- Principal components will find *p* linear combinations of the features which are orthogonal (independent) of each other and are ranked by variance, so the first principal component will have the largest variance and so on.
- If the features different dramatically in scale then it would be best to center and scale the raw data, e.g. $z_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j}$. The principal components from the centered and scaled data will be different then the unscaled data. There is no simple transformation.
- Thus, the first principal component is $Y_1 = a_{11}X_1 + \cdots + a_{p1}X_p = \boldsymbol{a}_1^T \boldsymbol{x}$
- The $Var(Y_1) = \boldsymbol{a}_1^T \boldsymbol{S} \boldsymbol{a}_1$

Principal Components

- We want to find $\mathbf{a}_1$ such that Var(Y_1) it has the largest variance of all normalized linear compounds that satisfies $\boldsymbol{a}_1^T\boldsymbol{a}_1 = 1$.
- Due to the constraint finding the maximum is a little more difficult but can be done with Lagrange multipliers.
- Skipping the details, the Lagrange multipliers results in p simultaneous equations, $(S - l_1\boldsymbol{I})\boldsymbol{a}_1 = 0$, where l_1 is the Lagrange multiplier.
- The only way for this to not have a trivial solution is if, $det(S - l_1\boldsymbol{I}) = \mathbf{0}$
- This means that l_1 is the characteristic root (eigenvalue) of **S** and $\boldsymbol{a}_1$ is its associated characteristic vector (eigenvector).

Principal Components

- If we pre-multiply $(S - l_1\boldsymbol{I})\boldsymbol{a}_1 = 0$ by $\boldsymbol{a}_1^T$ we get,
  $\boldsymbol{a}_1^T(S - l_1\boldsymbol{I})\boldsymbol{a}_1 = 0$
  $\boldsymbol{a}_1^T S\boldsymbol{a}_1 - \boldsymbol{a}_1^T l_1\boldsymbol{I}\boldsymbol{a}_1 = \boldsymbol{a}_1^T S\boldsymbol{a}_1 - l_1 = 0$
  $l_1 = \boldsymbol{a}_1^T\boldsymbol{S}\boldsymbol{a}_1 = Var(Y_1)$
- Since the first principal component should have the largest variance then l_1 should be the largest eigenvalue out of *p* possible eigenvalues.
- The second principal component satisfies, $\boldsymbol{a}_2^T\boldsymbol{a}_2 = 1$, and $\boldsymbol{a}_1^T\boldsymbol{a}_2 = 0$
- The second principal component is the second largest eigenvalue of **S** and so on.
- It is also the case that $l_1 + \cdots + l_p = tr\boldsymbol{S}$
- So, if we divide the variance of each principal component by the total variance it will equal the proportion of the total variance.

Principal Components, examples

- When $p>>n$, the model can be simplified by using the first, second and additional principal components as single features.
- However, since each principal component is a linear combination of all p features you haven't really removed potentially irrelevant features.

In these figures n=50, and p=45. In the left figure all 45 features contribute to the response, while in the right figure only 2 do.

The x-axis is the number of principal components used in the regression model.

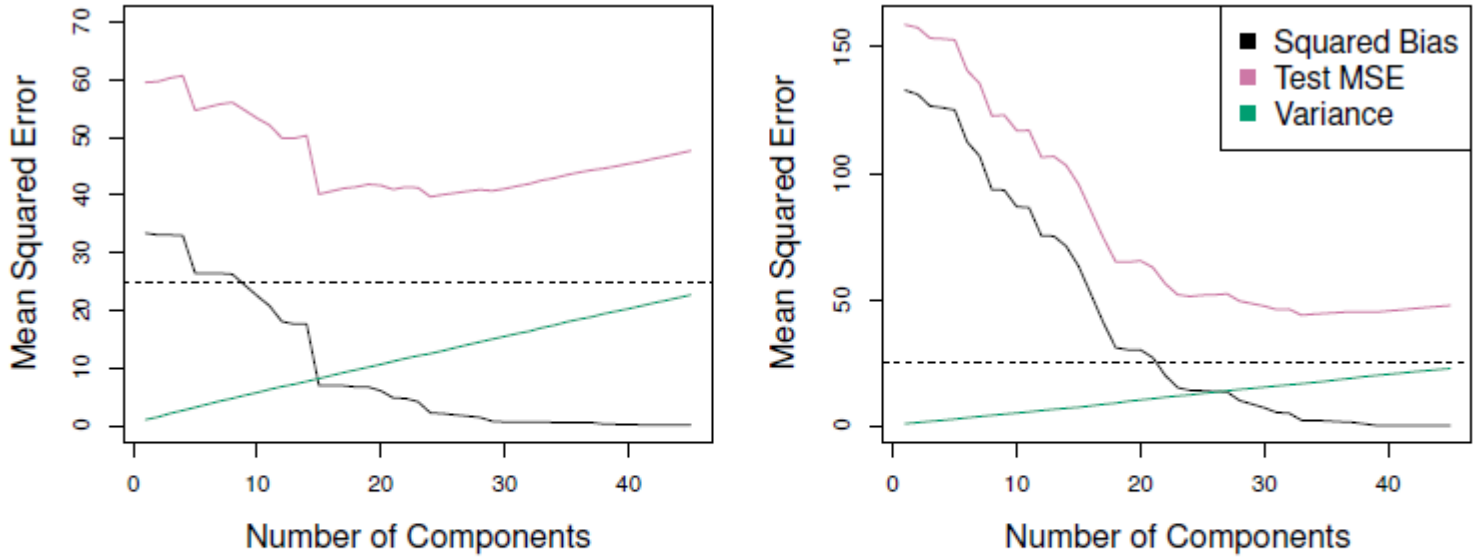


FIGURE 6.18. *PCR was applied to two simulated data sets. In each panel, the horizontal dashed line represents the irreducible error.* Left: *Simulated data from Figure 6.8.* Right: *Simulated data from Figure 6.9.*

Supervised Principal Components

- To get details see chapter 18 of "The elements of statistical learning" or Blair et at. 2007. *JASA* 101:119.
- This technique is designed for the $p>n$ case.
- We don't want to use all the features only those that are correlated with the outcomes, hence the supervision.
- The technique was originally designed for survival data. However, it can be used with normal regression problems.
- The program can be run with the "superpc" package written by Blair and Tibshirani.
- For a reasonably good tutorial go to http://statweb.stanford.edu/~tibs/superpc/tutorial.html

Supervised Principal Components

- Algorithm
  1. Compute the standardized univariate regression coefficients ($\frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}$, where v_j is *jth* diagonal of $(\mathbf{X}^T\mathbf{X})^{-1}$) for the outcome as a function of each feature.
  2. For each value of the threshold θ from the list $0 \leq \theta_1 < .. < \theta_K$:
  (a) Form a reduced data matrix consisting of only those features whose univariate coefficient exceeds θ in absolute value, and compute the first *m* (1-3) principal components of this matrix.
  (b) Use these principal components in a regression model to predict the outcome.
  3. Pick θ (and *m*) by cross-validation.

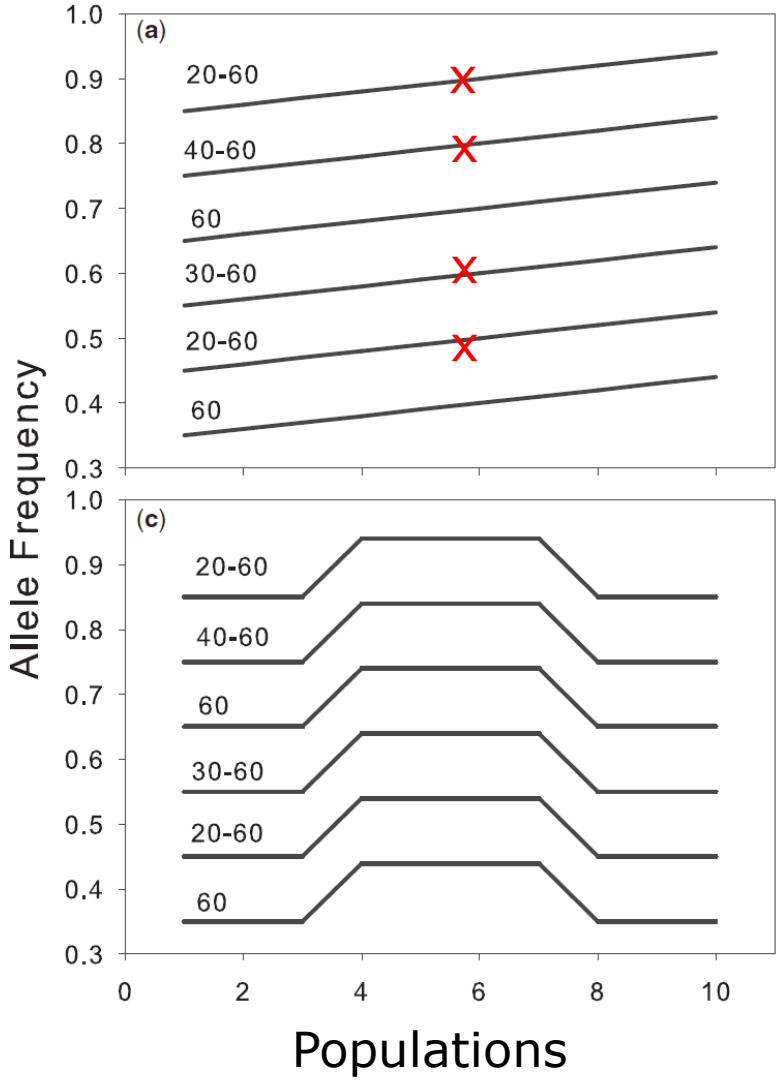
Example: Supervised Principal Components

- We use a simulated pooled genomic allele frequency database. In this database loci 1-30 have some effect on a phenotype, loci 31-40 show the same level of allele frequency differentiation as loci 1-30 but have NO effect on the phenotype and loci 41-2000 show random variation between populations and also do not affect the phenotype.
- There are a total of 40 populations (so n=40 and p=2000).
- Before doing this analysis, we remove loci by testing for allele frequency differences and using a false discovery rate of 5%.
- With this database the pre-filtering reduced p to 43.

Example: Supervised Principal Components

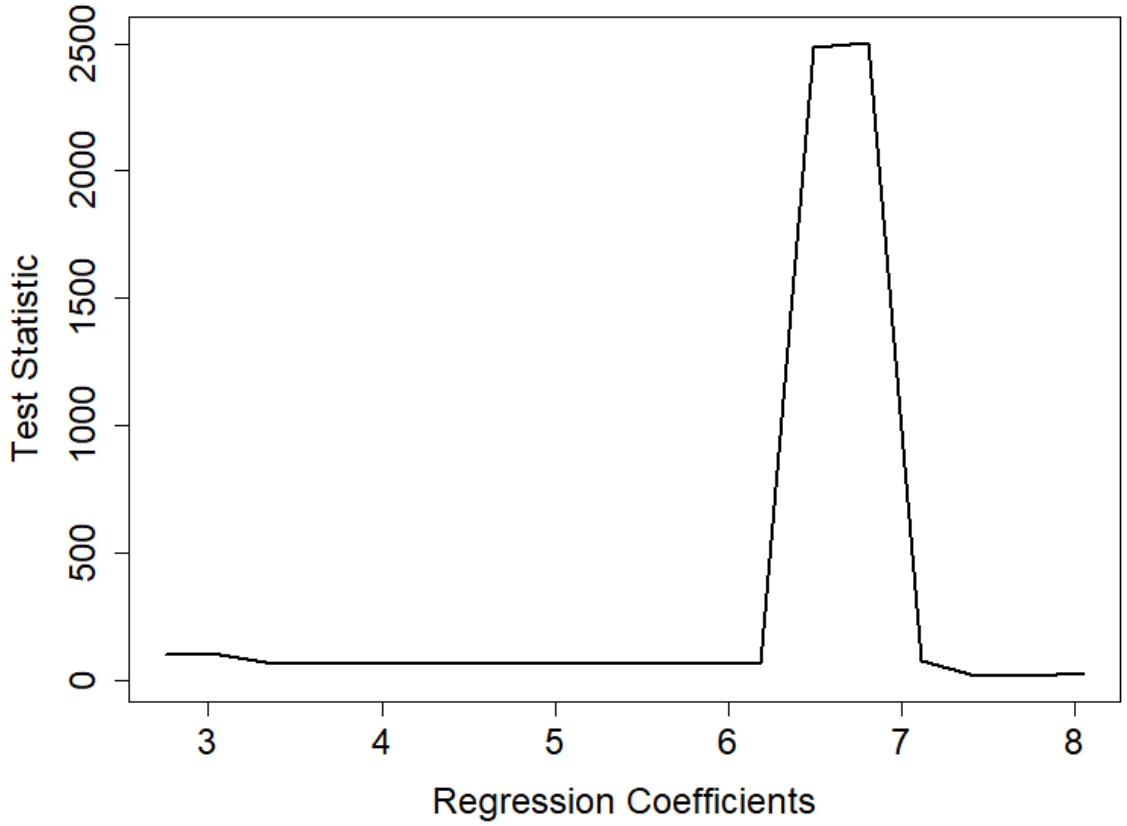
In this simulated database the allele frequency variation among 40 populations is shown in (a). Allele frequencies for 4 groups of 10 populations are marked with a red “x”.

These are the population mean allele frequencies. The database shows binomial sampling variation about these means.



Example: Supervised Principal Components

- Data was randomly divided into a training set of 32 populations and a test set of 8 populations.
- After training the data we choose the threshold value (θ_j) 6.49 from the graph below.
- Thus, only features with standardized regression coefficients greater than 6.49 will be included to compute PC's.
- Only the first principal component is shown.
- The test statistic is from a likelihood ratio test.



Example: Supervised Principal Components

- We can test the significance of the first three principal components.

```
sim401.train<- superpc.train(data.train, type="regression")
sim401.cv<-superpc.cv(sim401.train, data.train,n.components = 2) #From this determine threshold
sim401.fit<- superpc.predict(sim401.train, data.train, data.test, threshold=6.49, n.components=3,
prediction.type="continuous") #Predict test data response with threshold and 3 PC's

> superpc.fit.to.outcome(sim401.train, data.test, sim401.fit$v.pred)# Test fit of test data with
                                                                    each PC


Call:
lm(formula = data.test$y ~ ., data = temp.list)

Residuals:
         1          2          3          4          5          6          7          8
 4.291e-03 -2.465e-05 -8.364e-03  6.712e-03 -3.558e-05  4.458e-03 -3.632e-03 -3.405e-03

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.699444   0.003362 208.056  3.2e-09 ***
score.1      0.988445   0.013528  73.065  2.1e-07 ***
score.2      0.811663   0.198431   4.090    0.015 *
score.3     -0.295224   0.168617  -1.751    0.155
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.006672 on 4 degrees of freedom
Multiple R-squared:  0.9993,  Adjusted R-squared:  0.9987
F-statistic:  1849 on 3 and 4 DF,  p-value: 9.736e-07
```

Example: Supervised Principal Components

Results

Importance-score	Raw-score	Name
0.843	8.101	feature10
0.83	8.55	feature16
0.83	7.151	feature27
0.825	7.83	feature26
0.817	**6.858**	**feature36**
0.815	8.604	feature17
0.813	7.961	feature23
0.813	9.309	feature11
0.813	**7.83**	**feature32**
0.809	9.299	feature30
0.805	8.409	feature3
0.802	8.067	feature1
0.792	7.873	feature5
0.79	8.469	feature21
0.783	9.207	feature7
0.779	8.302	feature4
0.777	**7.335**	**feature38**
0.775	**7.479**	**feature34**
0.77	**8.463**	**feature37**
0.769	**8.01**	**feature31**
0.767	8.651	feature18
0.766	8.854	feature14
0.757	8.746	feature28
0.755	8.664	feature15
0.75	8.315	feature8
0.744	6.826	feature9
0.743	**7.393**	**feature33**
0.743	**7.613**	**feature40**
0.742	8.598	feature2
0.738	7.757	feature24
0.734	8.419	feature22
0.731	**7.03**	**feature35**
0.723	8.508	feature12
0.72	7.723	feature29
0.718	7.553	feature19
0.714	7.698	feature6
0.7	7.192	feature25
0.683	7.109	feature20
0.657	7.377	feature13

The input list included all these plus #'s 39, 453, and 1560
Thus, supervised principal components was only able to eliminate three loci and it included almost all of the non-causative loci (in bold). These results were based on only the first principal component.

FLAM (fused Lasso additive model)

- Apply FLAM to the same artificial database yielding the following sparse list.

FLAM 50% criteria	Frequency/100
feature1	73
feature2	100
feature4	98
feature6	90
feature7	89
feature8	85
feature11	95
feature12	71
feature14	98
feature16	80
feature17	70
feature21	90
feature22	79
feature26	94
feature27	72
feature28	92
feature30	97
feature32	**76**

Partial Least Squares

- This technique can be used for dimension reduction like principal component regression.
- Up to *p* new directions are created which are linear functions of the original features.
- Unlike principal components the new directions are based on ***X*** and ***y*** not just ***X*** like principal components.
- In principal components we chose each one to maximize the variance of the first , then second and so on. Partial least squares chooses directions (e.g. vectors that are linear combinations of the features) that have a high variance and high correlation with the outcomes (***y***).
- Partial least squares software can be found in the "pls" R-package.
- The detailed algorithm is on page 81 of the Elements book.

Partial Least Squares Algorithm

- Preliminaries
- Inner product, $< x, y > = x^T y = \sum_{i=1}^{p} x_i y_i$
- Linear regression with a single feature and no intercept yields a least squares estimate of, $\hat{\beta} = \frac{<x,y>}{<x,x>}$.
- If the feature has been centered and scaled then,

$$x^T x = \sum x_i^2 = var(x) = 1$$

Partial Least Squares Algorithm

- We seek *M* new orthogonal coordinates, $Z_1, Z_2, .., Z_M, M < p$ that are related to the original coordinates as, $Z_m = \sum_{j=1}^{p} \varphi_{jm} X_j$.
- 1. Center and scale the features. Start by computing $\hat{\varphi}_{1j} = < x_j, y >$ for each j.
- 2. From 1. construct the first direction, $z_1 = \sum_j \hat{\varphi}_{1j} x_j$. This is the first coordinate and its inputs are weighted by the strength of their univariate effect on y.
- 3. The features then have to be orthogonalized relative to z_1 and the process is then repeated.

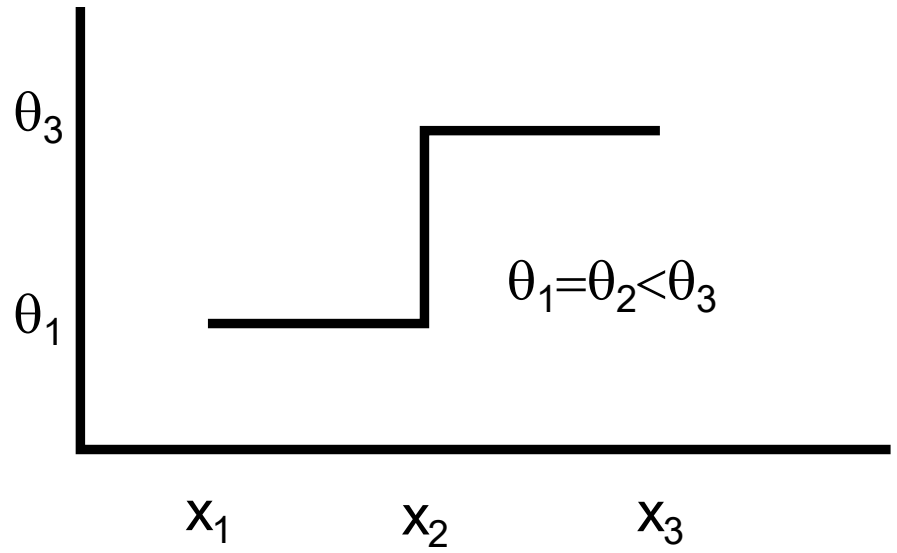
FLAM

- Fussed Lasso Additive Model
- Assume *n*-responses, $p_1, p_2, \ldots, p_n$. At one feature (j) assume the ordered values are x_{j1}, …, x_{jn}.
- The regression model is $E\left[p_i \middle| x_{ji}\right] = \theta_i$
- The D matrix inside of the l_1 norm encourages adjacent parameters to be zero, e.g. $\left|\hat{\theta}_{i-1} - \hat{\theta}_i\right| = 0$. Knots (jumps) will only appear if the reduce the sum of squares.

$$\underset{\theta_j \in \mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\left\|\mathbf{P} - \theta_j\right\|_2^2 + \lambda\left\|\mathbf{D}\theta_j\right\|_1$$

where

$$\mathbf{D} = \begin{pmatrix} 1 & -1 & 0 & \ldots .0 \\ 0 & 1 & -1 & \ldots .0 \\ . & . & . & . \\ 0 & 0 \ldots . & 1 & -1 \end{pmatrix}$$



FLAM

- The full minimization problem has a "group lasso" although it uses an l_2 norm that encourages discarding whole loci.
- $\boldsymbol{P}_j$ is a permutation matrix which orders the $\boldsymbol{x}_j$ from least to greatest.
- The tuning parameters λ and α have to be chosen from a grid. Luckily there is a finite value for λ above which the results are completely sparse. α ranges from 0 to 1.
- If the outcomes, *y*, vary widely in magnitude consider a transformation like log(*y*). Since the test MSE determines the model parameters, very small *y* may have a minor effect on the final model.

$$\underset{\theta_0 \in \mathbb{R}, \boldsymbol{\theta}_j \in \mathbb{R}^n, 1 \le j \le p}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{y} - \sum_{j=1}^{p} \boldsymbol{\theta}_j - \theta_0 \mathbf{1} \right\|_2^2 + \alpha\lambda \sum_{j=1}^{p} \|\boldsymbol{D}\boldsymbol{P}_j\boldsymbol{\theta}_j\|_1 + (1-\alpha)\lambda \sum_{j=1}^{p} \|\boldsymbol{\theta}_j\|_2$$

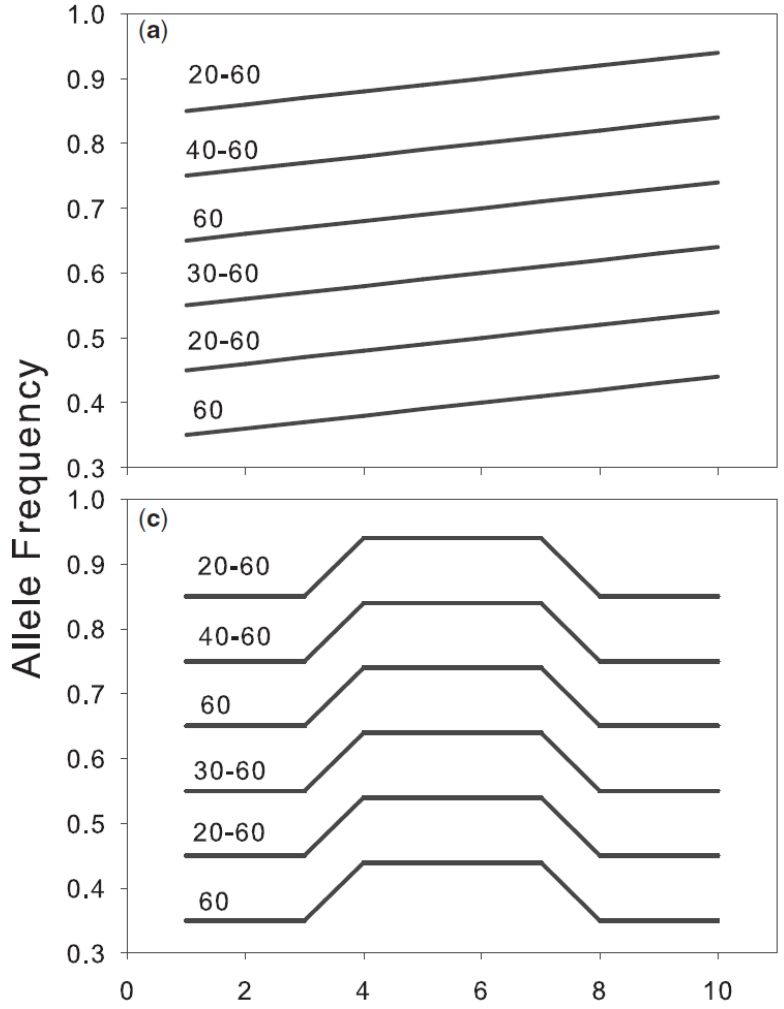
FLAM minimization details

- There are three numerical methods for finding the minimum of the FLAM objective function.
- One method, “BCD” or block coordinate descent will reveal several equivalent minima but simply permuting the order of the columns of the feature matrix.
- If this permutation is done, say 100 times, then features can be ranked by how often they are found in the sparse set.
- The number of causative SNPs can be improved by taking all that appear in the sparse list at least 50% as often as the most common SNP.
- This is referred to as the 50% rule.

FLAM: Example

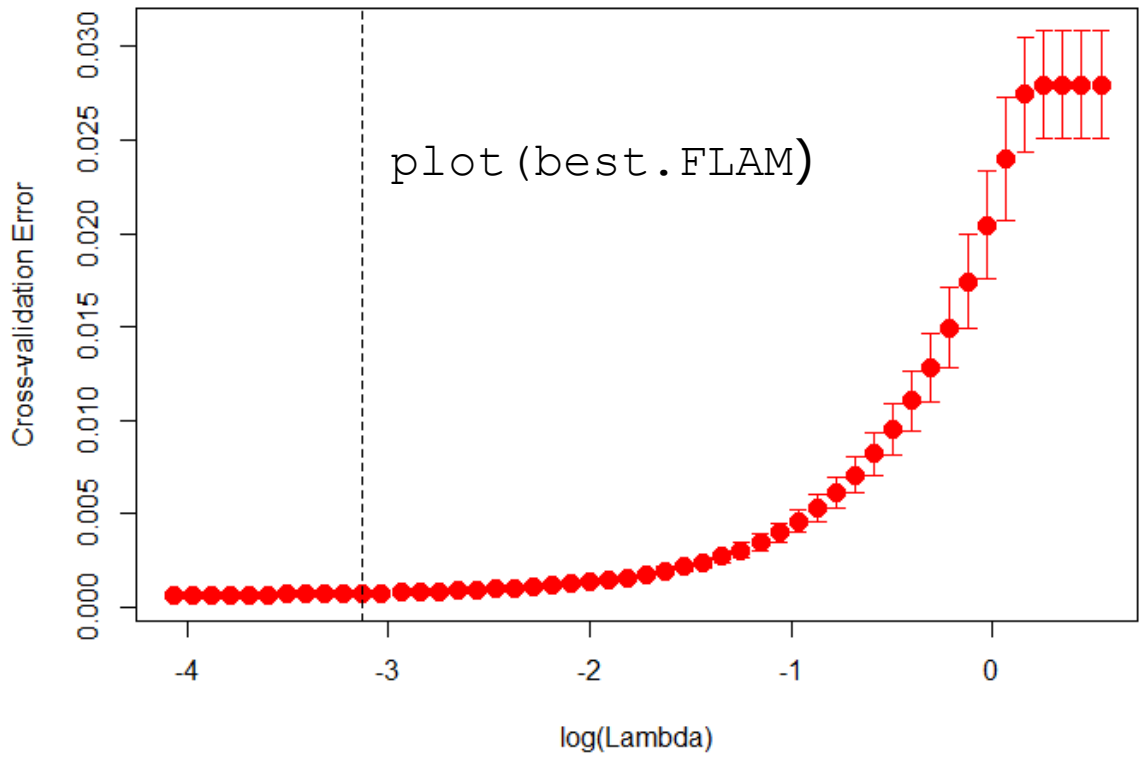
- One simulated database with 40 populations.
- The patterns are shown in panel (c) below.
- FLAM is in the "flam" package.
- After initial filtering, FLAM was run 100 times on permuted genetic databases.
- From the sparse list using the 50% rule, FLAM was run on just those features after the best α was determined.
- The final results was,

```
best.FLAM<-
flamCV(sparse.gen,pheno.data,alpha=0.4,n.fold=5,seed=1,met
hod="BCD")
```

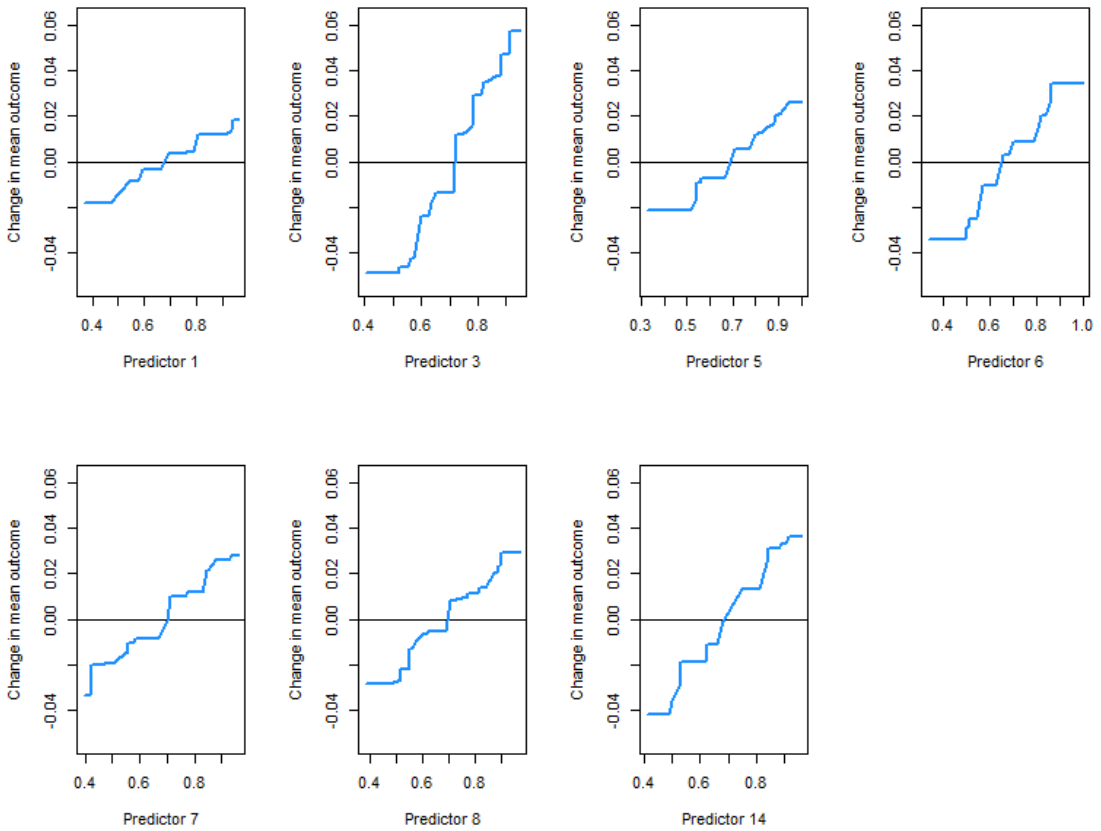


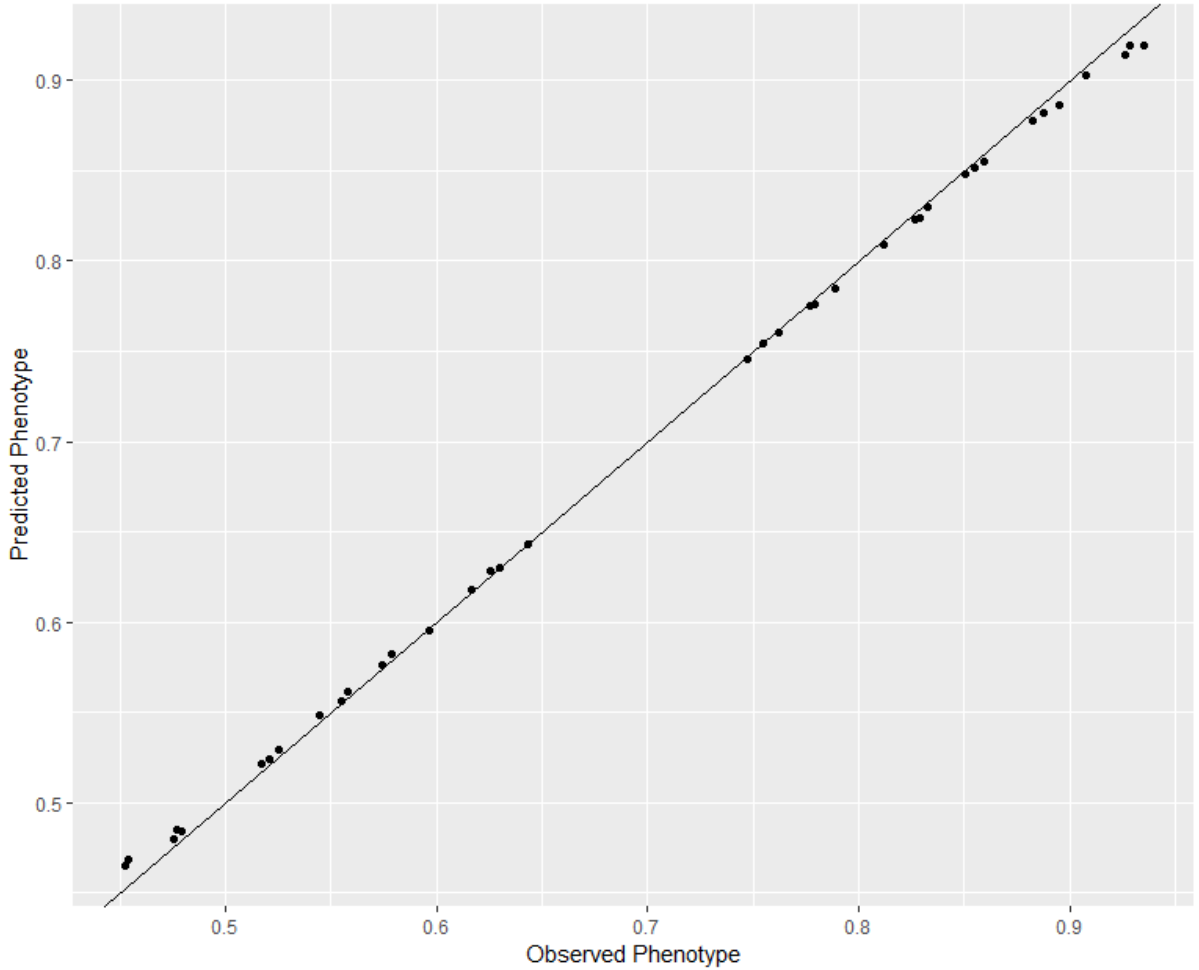
FLAM: Example

```
> summary(best.FLAM)
Call:
flamCV(x = sparse.gen, y = pheno.data, alpha = 0.4,
method = "BCD",
    n.fold = 5, seed = 1)

FLAM was fit using the tuning parameters:

lambda: 1.713 1.559 1.419 1.292 1.176 1.071 0.975 0.887
0.808 0.735 0.669 0.609 0.555 0.505 0.46 0.418 0.381
0.347 0.316 0.287 0.261 0.238 0.217 0.197 0.18 0.163
0.149 0.135 0.123 0.112 0.102 0.093 0.085 0.077 0.07
0.064 0.058 0.053 0.048 0.044 0.04 0.036 0.033 0.03
0.027 0.025 0.023 0.021 0.019 0.017

alpha: 0.4

Cross-validation with K=5 folds was used to choose
lambda.
Lambda was chosen to be the largest value with CV error
within one standard error of the minimum CV error.
Best.FLAM$index.cv is the index corresponding to the
best model.

The chosen lambda was 0.044.
This corresponds to 7 predictors having non-sparse fits.
The predictors with non-sparse fits:
1 3 5 6 7 8 14
```



FLAM: Example






```
plot(best.FLAM$flam.out,best.FLAM$index.cv)
5= gene 7
6= gene 8
7= gene 10
8= gene 13
14= gene 27
```

```
best.predict<- cbind(pheno.data,best.FLAM$flam.out$y.hat.mat[best.FLAM$index.cv,])
best.predict<- as.data.frame(best.predict)
colnames(best.predict)<- c("Observed","Predicted")
library(ggplot2)
ggplot(best.predict,aes(Observed,Predicted))+geom_point()
+ylab("Predicted Phenotype")+xlab("Observed Phenotype")+geom_abline()
```

Feature Assessment and Multiple Testing

- Problem: determine if there are significant differences in the mean feature value between two groups.
- If $p>>n$ then this involves multiple hypothesis tests.
- If the type-I error (the chance of rejecting the null hypothesis when true) is 5% then we expect to have many type-I errors when p is very large.
- A family wise error rate (FWER) controls the type-I error on a collection of hypothesis tests.
- If we do a total of M hypothesis tests with a type-I error rate of α, then the chance that any of the M tests results in a type-I error is, $(1-(1-\alpha)^M)$=FWER. If there is positive dependence between the tests then FWER will be smaller.
- See chapter 13 of James et al. (2022).

False Discovery Rate

TABLE 18.5. *Possible outcomes from M hypothesis tests. Note that V is the number of false-positive tests; the type-I error rate is $\mathrm{E}(V)/M_0$. The type-II error rate is $\mathrm{E}(T)/M_1$, and the power is $1 - \mathrm{E}(T)/M_1$.*

	Called Not Significant	Called Significant	Total
H_0 True	U	V	M_0
H_0 False	T	S	M_1
Total	$M - R$	R	M

- From table 18.5 we can estimate the false discovery rate (FDR) as E(*V*/*R*)

Methods to control family-wise error rates

- **Bonferroni inequality**. Suppose we want a FWER of α_{FWER}. Then we want α_{FWER} to equal the chance that we falsely reject just one of m different hypothesis tests each carried out with a type-I error of $\tilde{\alpha}$. Then no matter what the relationship is among the m tests, $\alpha_{FWER} \leq \sum_{i=1}^{m} \tilde{\alpha} = m\tilde{\alpha}$.
- Thus, if we want a FWER of 0.05, then each individual test, $\tilde{\alpha}$, must be set to 0.05/m.

Holm's step down procedure

- This method is less conservative and thus should have greater power.
- Algorithm:
  1. Fix the false discovery rate at α and let $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(M)}$ denote the ordered *p*-values.
  2. Define $L = min\left\{j\text{: } p_{(j)} > \frac{\alpha}{M+1-j}\right\}$
  3. Reject all hypotheses for which $p_j \leq p_{(L)}$, the Holm rejection threshold.

Comparison of Bonferroni and Holm's step-down

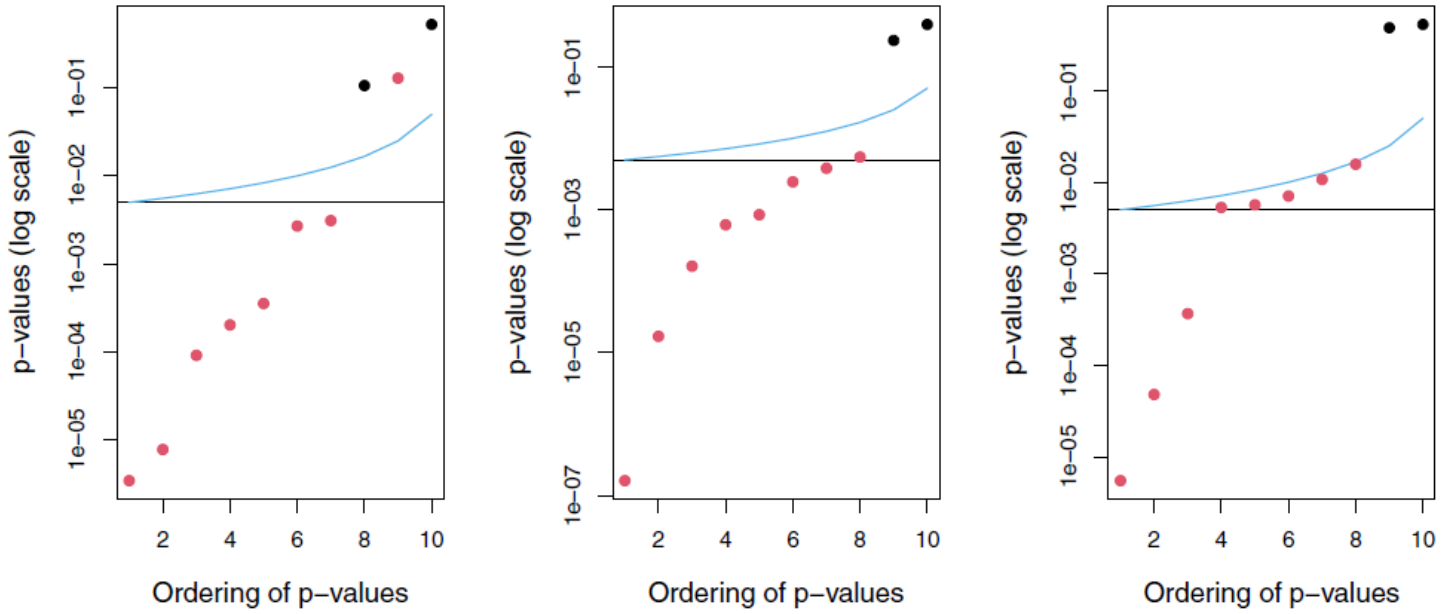


FIGURE 13.3. *Each panel displays, for a separate simulation, the sorted p-values for tests of* $m = 10$ *null hypotheses. The p-values corresponding to the* $m_0 = 2$ *true null hypotheses are displayed in black, and the rest are in red. When controlling the FWER at level* 0.05*, the Bonferroni procedure rejects all null hypotheses that fall below the black line, and the Holm procedure rejects all null hypotheses that fall below the blue line. The region between the blue and black lines indicates null hypotheses that are rejected using the Holm procedure but not using the Bonferroni procedure. In the center panel, the Holm procedure rejects one more null hypothesis than the Bonferroni procedure. In the right-hand panel, it rejects five more null hypotheses.*

By rejecting the hypotheses between the black and blue line, the Holm step-down method is displaying greater power than the Bonferroni method.

Benjamini and Hockberg Method

- See, 1995, J. Royal Stat. Soc. Series B 85: 289-300.
- Algorithm:
  1. Fix the false discovery rate at α and let $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(M)}$ denote the ordered *p*-values.
  2. Define $L = max\left\{j\colon p_{(j)} < \alpha \cdot \frac{j}{M}\right\}$
  3. Reject all hypotheses for which $p_j \leq p_{(L)}$, the BH rejection threshold.

Benjamini and Hockberg Method

The BH threshold is 0.00012. The Bonferroni with α=0.15 is 0.000012, an order of magnitude smaller.

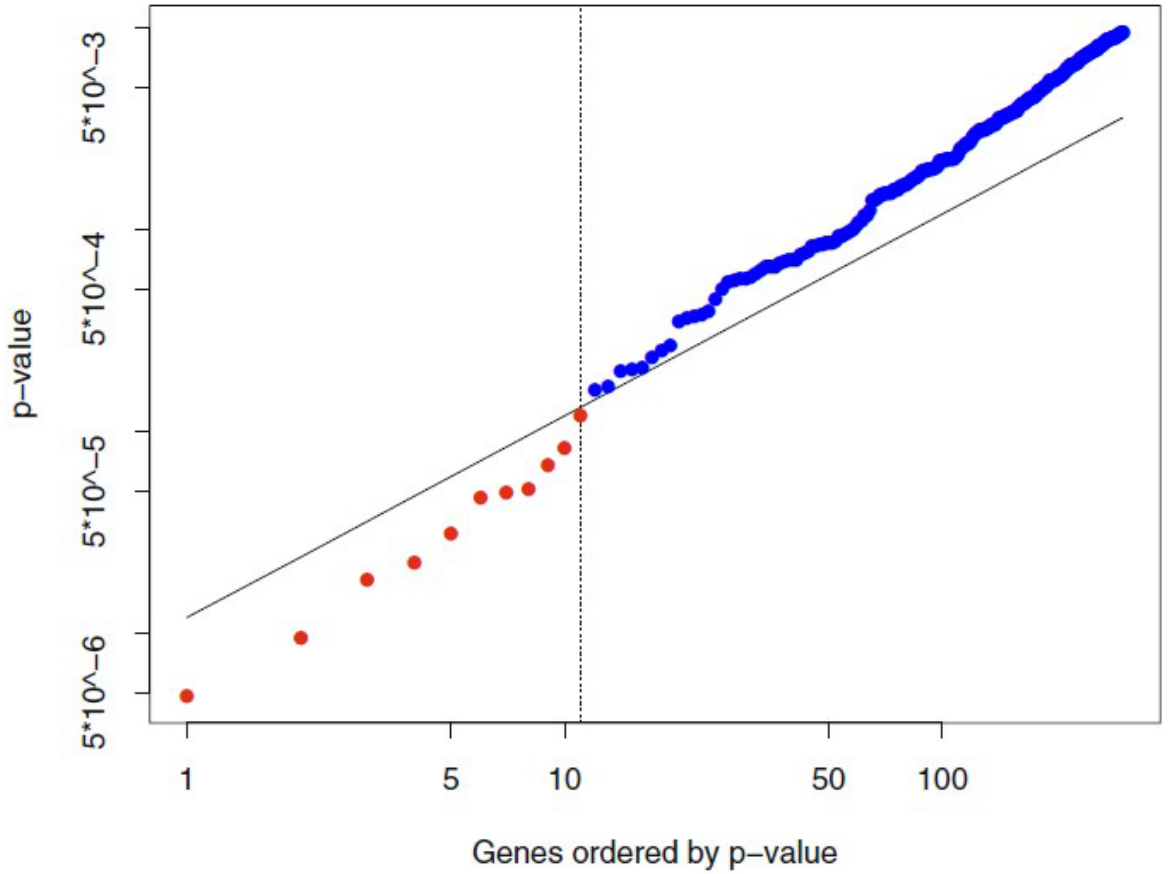


FIGURE 18.19. *Microarray example continued. Shown is a plot of the ordered p-values $p_{(j)}$ and the line $0.15 \cdot (j/12,625)$, for the Benjamini–Hochberg method. The largest j for which the p-value $p_{(j)}$ falls below the line, gives the BH threshold. Here this occurs at $j = 11$, indicated by the vertical line. Thus the BH method calls significant the 11 genes (in red) with smallest p-values.*

Feature Assessment and Multiple Testing

To test each feature for significant differences first calculate a *t*-statistic, $t_j = \frac{\bar{x}_{2j} - \bar{x}_{1j}}{se_j}$, where in general, $\bar{x}_{lj} = \sum_{i \in C_l} x_{ij}/N_l$, where C_l are the indices of group *l* with sample size N_l.

The standard error is calculated as,

$$se_j = \hat{\sigma}_j \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}$$

$$\hat{\sigma}_j{}^2 = \frac{1}{N_1 + N_2 - 2}\left(\sum_{i \in C_1}(x_{ij} - \bar{x}_{1j})^2 + \sum_{i \in C_2}(x_{ij} - \bar{x}_{2j})^2\right)$$

We can approximate the distribution using a *t*-distribution or make a permutation distribution.

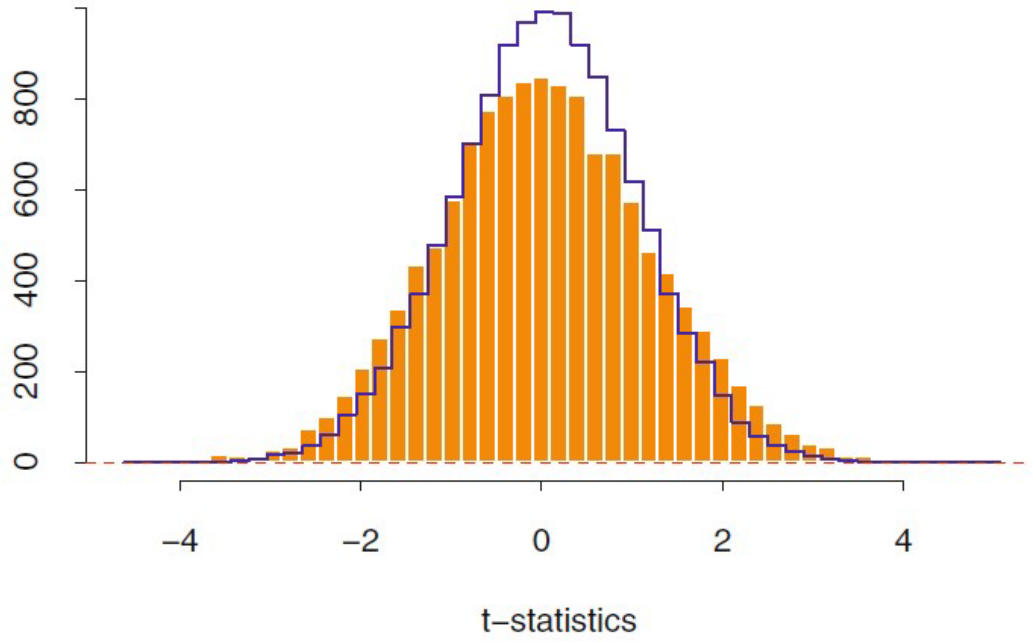


FIGURE 18.18. *Radiation sensitivity microarray example. A histogram of the* 12,625 *t-statistics comparing the radiation-sensitive versus insensitive groups. Overlaid in blue is the histogram of the t-statistics from* 1000 *permutations of the sample labels.*

Permutation Distribution

- Here we permute the labels of the features many times and for each permutation compute the *t*-statistics.
- In theory we could look at all possible permutations. So, the number of different ways to sample labels for group 1 are $K = \binom{N_1 + N_2}{N_1}$. Thus, for permutation *k*, the *t*-statistic is t_j^k, then the *p*-value for feature-*j* is $p_j = \frac{1}{K}\sum_{k=1}^{K} I\left(\left|t_j^k\right| > \left|t_j\right|\right)$. If the features are very similar then the calculation for p_j can be summed over all features to get a better average.
- The Bonferroni method can give a FWER of $\leq \alpha$ by simply dividing the individual error rate, α, by the number of tests. However, this can be overly conservation for large numbers of tests.

Plug-in estimate of false discovery rate

- Algorithm:
  1. Create *K* permutations of the data, producing *t*-statistics t_j^k for features j=1, …,M and permutations k=1,…, K.
  2. For a range of values of the cut-point C, let
  $R_{obs} = \sum_{j=1}^{M} I(|t_j| > C), \widehat{E(V)} = \frac{1}{K}\sum_{j=1}^{M}\sum_{k=1}^{K} I(|t_j^k| > C)$
  recall that R is the total number of observed significant tests and V is the number of falsely declared significant tests.
  3. Estimate the FDR by $\widehat{FDR} = \widehat{E(V)}/R_{obs}$
- For the microarray data the BH threshold was for t=4.101. If we use as the cut-point, R_{obs}=11, and $\widehat{E(V)} = 1.518$, thus $\widehat{FDR} = 0.14$
- Thus, to make the FDR=0.15, by the plug in we would need to decrease the cut-point to < 4.101
- The plug in method rejects a greater number of hypotheses while controlling the same error rate, which leads to greater power (Storey, 2002, J. Roy Soc. Stat. B 64: 479.