Chapter 11: Shrinkage Methods

Stats 500, Fall 2015
Brian Thelen, University of Michigan
443 West Hall, bjthelen@umich.edu

Shrinkage Methods

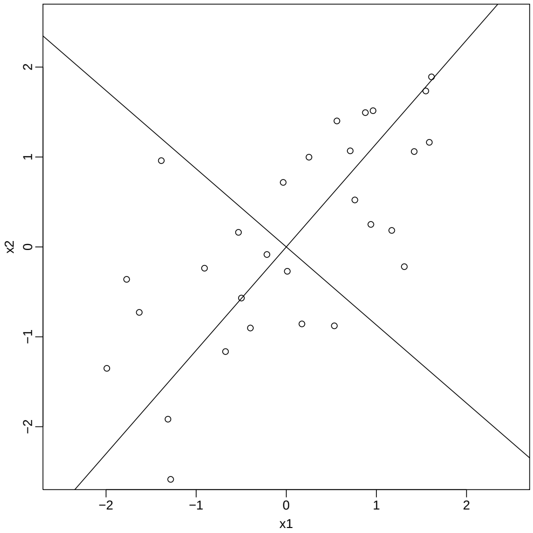
- Principal components regression
- Partial least squares
- Ridge regression

Principal Components (PC)

Main uses:

- **Reduce the dimensionality of the data**
- Find linear combinations of predictors that explain the most variation in the data
- Facilitate visualization
- In regression, makes predictors orthogonal to each other

PC Illustration



Definition of PCs

1. Center each variable by its mean, $x_j - \bar{x}_j \Rightarrow X_{n \times p}$.
2. Find u_1 such that $var(Xu_1)$ is maximized subject to $u_1^T u_1 = 1$.
3. For each $k = 2, ..p$, given $u_1, \dots, u_{k-1}$, find u_k such that $var(Xu_k)$ is maximized subject to $u_k^T u_k = 1$ and $u_k^T u_j = 0$ for all $j = 1, \dots, k-1$.
4. Vectors u_j are called the PC directions.
5. Vectors $z_j = Xu_j$ are called the principal components of X.

Remarks

- $z_j = Xu_j$ are **projections** of data points on the direction u_j.
- u_j are the **eigenvectors** of $X^T X$.
- $var(Xu_j) = \lambda_j$, the **eigenvalues** of $X^T X$.
- $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_p$
- Combine vectors into a matrix and get

$$diag(\lambda_1, \ldots, \lambda_p) = Z^T Z = U^T X^T X U$$

- Recommended: standardize each variable first.

Principal Components Regression

PCR replaces the regression model

$$y = \beta_0 + \beta_1 x_1 + \ldots \beta_p x_p$$

with

$$\mathbf{y} = \beta'_{\mathbf{0}} + \beta'_{\mathbf{1}}\mathbf{z_1} + \cdots + \beta'_{\mathbf{k}}\mathbf{z_k}$$

Note: PCs are centered, so $\hat{\beta}'_0 = \bar{y}$.

How do we pick the number of PCs?

- Typically most variation in X can be represented by a few principal components – **Dimension reduction** .
- Can choose k to explain certain **percent of variation:** pick first k so that

$$\sum_{i=1}^{k} \lambda_i \geq (1-\alpha) \sum_{i=1}^{p} \lambda_i$$

- Can look at the **scree plot** (sq. rooted eigenvalues in decreasing order) and look for a gap in eigenvalues
- More sophisticated methods for estimating **intrinsic dimension** of the data

Food Analyzer Example

- Response: fat content
- Predictors: 100 channel spectrum of absorbances
- Number of data points: $n = 215$
- Number of predictors: $p = 100$

Prediction Performance

Goal : build a model that predicts well on **future data** .

- Divide the data into two groups: **training data** and **test data** .
- Build the models using the training data and evaluate them on the test data.

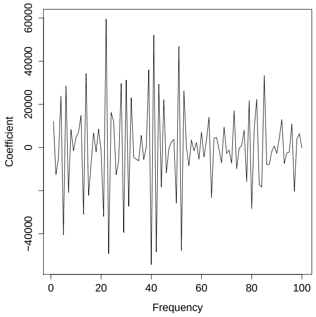
Food Analyzer Example Continued

```
> library(faraway)
> data(meatspec)
> dim(meatspec)
[1] 215 101
## Training and test data
> trainmeat = meatspec[1:172,]
> testmeat = meatspec[173:215,]

## Linear model
> modlm = lm(fat ~ ., tr)
> summary(modlm)$r.squared
[1] 0.9970196

## Root mean squared error
> rmse = function(x, y) { sqrt(mean( (x - y)^2 ))}
> rmse(modlm$fit, trainmeat$fat)
[1] 0.6903167
> ## Prediction
> rmse( predict(modlm, newdata=testmeat), testmeat$fat )
[1] 3.814000
```

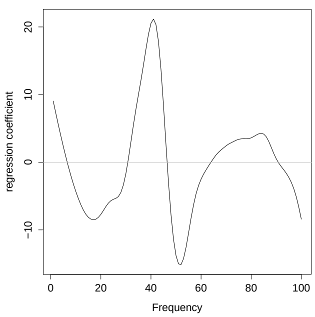
```
## AIC
> modlm2 = step(modlm)
> rmse( modlm$fit, tr$fat )
[1] 0.7095069
> rmse( predict(modlm2, newdata=testmeat), testmeat$fat )
[1] 3.590245

## Principal components regression
> library(stats)
> meatpca = prcomp(tr[,-101])
## Square root of the eigenvalues
> round(meatpca$sdev, 3)
  [1] 5.055 0.511 0.282 0.168 0.038 0.025 0.014
  [8] 0.011 0.005 0.003 0.002 0.002 0.001 0.001
 ... ...
> matplot(1:100, meatpca$rot[,1:3], type="l",
          xlab="Frequency", ylab="",lwd=3)
```

```
## Make a scree plot (to choose number of PCs k)
> plot(1:10, meatpca$sdev[1:10], type="l",
       xlab="PC number", ylab="SD of PC")
## Fit all PCRs at once and calculate test RMSE for each k
> library(pls)
> modpcr = pcr(fat ~ ., data=tr, ncomp=50)
> rmsmeat = NULL
>  for (k in 1:50) {
+    pv = predict(modprc, newdata=testmeat, ncomp=k)
+    rmsmeat[k] = rmse(pv, te$fat )  }
> plot(rmsmeat, xlab="PC number",   ylab="Test RMS")
# scree plot suggestion
> rmsmeat[5]
[1] 3.533628
> which.min(rmsmeat)
[1] 27
> rmsmeat[27]
[1] 1.854858
```

Comparison of Coefficients



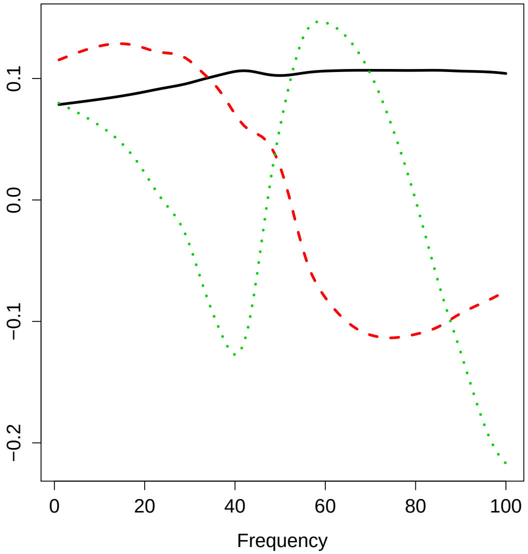
Coefficients from Linear Model



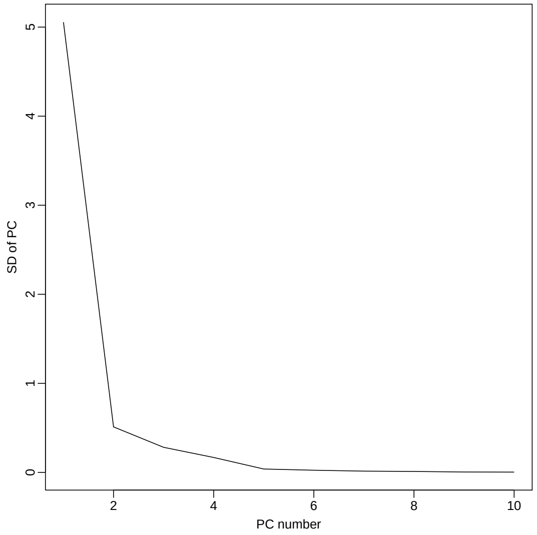
Effective coefficients from Pr. Component Regression (n=5)

```
plot(modlm$coef[-1],xlab="Frequency",ylab="Coefficient",
  + type="l")
coefplot(modpcr, ncomp=5, xlab="Frequency",main="")
```

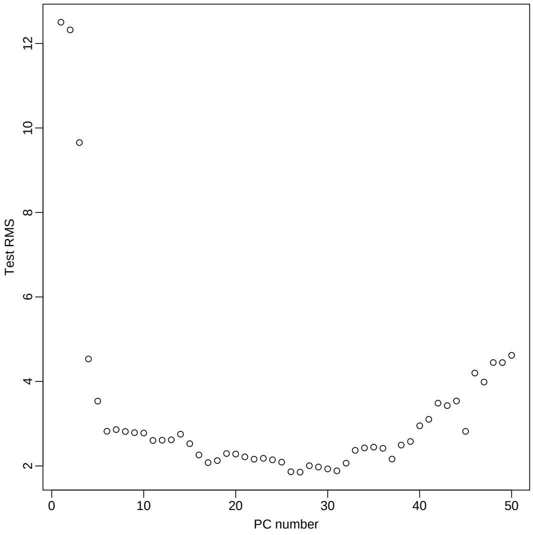
Food Analyzer: First 3 PCs



Food Analyzer: Scree plot



Food Analyzer: Test RMSE



Cross-Validation

- We cannot use the test data to pick k.
- Setting some of the training data aside for validation is possible, but reduces training sample size

A possible solution: **K-fold cross-validation** :

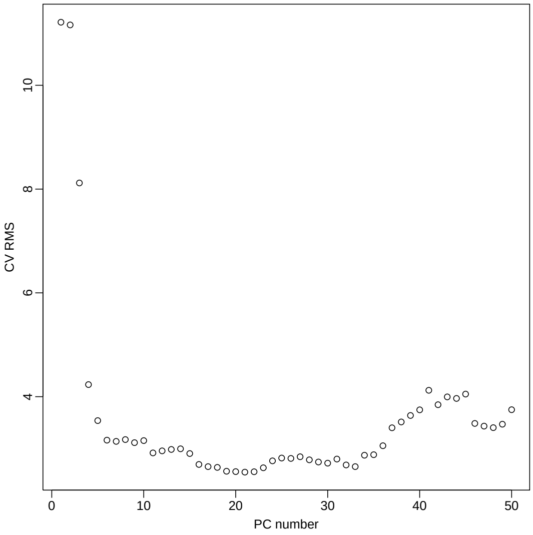
1. Divide the data into K parts
2. Use parts $2, \ldots, K$ as *training* data and part 1 as *test* data. Compute prediction errors on part 1.
3. Repeat for each part
4. Average prediction errors from all parts and pick k minimizing the average.

Food Analyzer Example: Cross-validation

```
> modpcr2 = pcr(fat ~ ., data=trainmeat, ncomp=50,
> +           validation="CV",  segments = 10)
> rmsCV= RMSEP(modpcr2, estimate='CV')
> which.min(rmsCV$val)
      19
# Another try
> modpcr2 = pcr(fat ~ ., data=trainmeat, ncomp=50,
> +           validation="CV", segments = 10)
> rmsCV= RMSEP(modpcr2, estimate='CV')
> which.min(rmsCV$val)
      21

## Plot the RMSE; k=0 is the model with intercept only
> plot(rmsCV$val, xlab="PC number", ylab="CV RMS")
## Get test error
> yfit = predict(modpcr2, newdata=testmeat, ncomp=21)
> rmse(testmeat$fat, yfit)
[1] 2.214545
```

CV tends to underestimate the real test RMSE – often close.

Food Analyzer Example (PCR)



Partial Least Squares

- **PCR** ignores y when building z's
- Partial least squares (**PLS**) chooses z's that are best at predicting y.
- PLS can handle multiple-output regression (vector-valued y)
- PLS does not solve a well-defined modelling problem
- Many algorithms for PLS exist
- Also need to select number of components
- No interpretation

Partial Least Squares Continued

Algorithm:

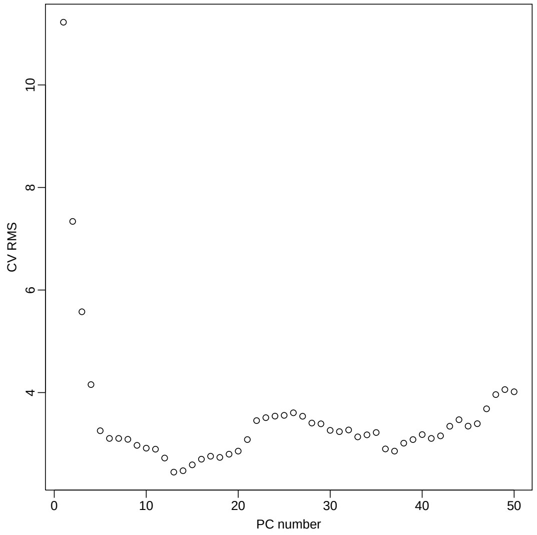
1. Center y, center and standardize each x_j
2. Regress y on each x_j *separately* to get α_j
3. Construct $z_1 = \sum \alpha_j x_j$, which is the first PLS component
4. Regress y on z_1 to get $\hat{\beta}_1$
5. Orthogonalize each x_j with respect to z_1
6. Continue until the final model is fit:

$$\hat{y} = \bar{y} + \hat{\beta}_1 z_1 + \cdots \hat{\beta}_k z_k$$

Food Analyzer Example

```
> ## Partial least squares
> modpls = plsr(fat ~ ., data=tr, ncomp=50, validation="CV")
> # plot RMSE estimated by CV
> pls_rmsCV = RMSEP(modpls, estimate='CV')
> plot(pls_rmsCV$val, xlab="PC number",ylab="CV RMS")
> which.min(pls_rmsCV$val)
[1] 14
> ## RMSE on the training data
> dim(modpls$fit)
[1] 172   1  50
> rmse(modpls$fit[,,14], tr$fat)
[1] 1.952796

> ## RMSE on the test data
> ypred.test = predict(plsg, newdata=testmeat)
> dim(ypred.test)
[1] 43  1 50
> rmse(ypred.test[,,14], testmeat$fat)
[1] 2.011180
```

Food Analyzer Example (PLS)



Ridge Regression

When to use this: the predictors are collinear and the usual least squares estimates are unstable. Center y, center and standardize each x_j. Consider

$$\min(y - X\beta)^T(y - X\beta) + \lambda\|\beta\|^2$$

β does not include the intercept β_0. Solution:

$$\hat{\beta} = (\mathbf{X^T X} + \lambda\mathbf{I})^{-\mathbf{1}}\mathbf{X^T y}$$

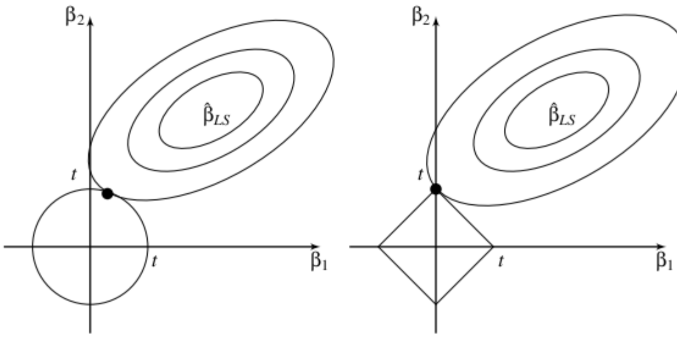
Remarks

- $\lambda = 0$ reduces to the usual least squares solution
- $\lambda \to \infty$ implies $\hat{\beta} \to 0$
- Alternative formulation: solve

$$\min(y - X\beta)^T(y - X\beta)$$

  subject to $\|\beta\|^2 \leq C$.
- Another shrinkage method is **Lasso** : replace $\|\beta\|^2 = \sum_i \beta_i^2$ with $\sum_i |\beta_i|$.
- Lasso shrinks many coefficients to exactly 0; ridge makes all coefficients smaller but does not set them to 0.

Picture: Ridge Regression and Lasso



Bias-Variance Trade-off

Why does ridge regression work? Is it biased?

$$E(\hat{\beta}) = (X^T X + \lambda I)^{-1}(X^T X)\beta$$

Goal: mean squared error

$$\begin{aligned}\mathbf{E}(\hat{\mathbf{y}} - \mathbf{E}(\mathbf{y}))^{\mathbf{2}} &= (E(\hat{y}) - E(y))^2 + E\left(\hat{y} - E(\hat{y})\right)^2 \\ &= \text{Bias}^{\mathbf{2}} + \text{Variance}\end{aligned}$$

Sometimes introducing a small bias leads to a large drop in variance.

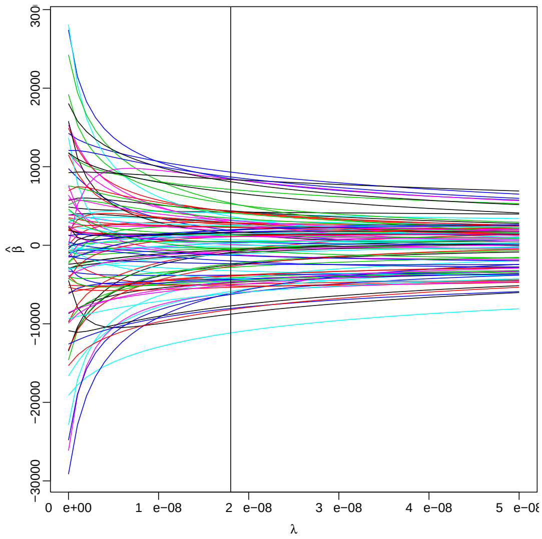
Food Analyzer: Ridge Regression

```
> library(MASS)
> ## The function will center the training data
> modridge = lm.ridge(fat ~ ., lambda=seq(0, 5e-8, 1e-9),
+ data = trainmeat)
> matplot(modridge$lambda, t(modridge$coef), type="l", lty=1,
+ xlab=expression(lambda),  ylab=expression(hat(beta)) )

> ## Select an appropriate lambda
> select(modridge)
modified HKB estimator is 1.058342e-08
modified L-W estimator is 0.7096864
smallest value of GCV  at 1.8e-08
> abline(v=1.8e-8)

> ## No fitted values returned - compute yourself
> yfit = modridge$ym + scale(tr[,-101], center=gridge$xm,
    scale=modridge$scales ) %*% modridge$coef[, 19]
> # RMSE on training data
> rmse(yfit, trainmeat$fat)
[1] 0.8045392
```

```
## No predict() function for ridge
## Predict for test data, but need to center with training means
> ypred =  modridge$ym + scale(testmeat[,-101], center=
+ modridge$xm, scale = modridge$scales ) %*% modridge$coef[,19]
> rmse(ypred, testmeat$fat)
[1] 4.096579
```

```
## One really bad prediction
> c( ypred[13], testmeat$fat[13] )
     185
11.18769 34.80000
> rmse( ypred[-13], testmeat$fat[-13] )
[1] 1.976548
```

Food Analyzer Example Continued



Summary

- Main reason to use shrinkage: too many predictors or collinearity
- Interpretation is usually lost
- Ridge is still a linear model in the original predictors but no inference
- Prediction is usually improved by shrinkage
- All require selecting a **tuning parameter** (number of components for PCR and PLS, λ for ridge) – need validation data or cross-validation.