Multicollinearity

- In many real applications, the model input variables are not independent of one another
- Like scaling, if they are closely related to one another the matrix inverse $A^T A$ may be ill-conditioned
- This is similar to dividing by a very small number
- This can cause very large model coefficients and ultimately unstable predictions
- This problem occurs if two or more inputs have a linear relationship to one another:
  $$x_i \approx \sum_{j \neq i} \alpha_j x_j$$
  for some coefficients $\alpha_j$
- Generally, this problem is called **multicollinearity**

Multicollinearity Continued

- For example, suppose our statistical model is
  $$y = 3x_1 + 2x_2 + \varepsilon$$
- If $x_1 = 2x_2$ (perfectly correlated), then this statistical model has many equivalent representations
  $$y = 3x_1 + 2x_2 + \varepsilon$$
  $$y = 4x_1 + \varepsilon$$
  $$y = 2x_1 + 4x_2 + \varepsilon$$
- The data cannot tell us which one of these models is correct
- There are a number of measures that can be taken to reduce this effect
- We will discuss four of them
Singular Value Decomposition

\[ A_{n \times p} = U_{n \times n} \Sigma_{n \times p} V_{p \times p}^T \]

- The matrix \( U \) can be written as

\[ U = \begin{bmatrix} U_+ & U_- \end{bmatrix}_{n \times p} \]

- This enables us to decompose the \( A \) matrix slightly differently

\[ A_{n \times p} = U_{n \times n} \Sigma_{n \times p} V_{p \times p}^T = U_+ \Sigma_+ V_+^T \]

- The elements along the diagonal of \( \Sigma_+ \) are called the singular values of \( A \)

- They are nonnegative

- Usually they are ordered such that

\[ \sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_p \geq 0 \]

Example 1: Multicollinearity

\[ \text{N} = 20; \]
\[ x1 = \text{rand}(\text{N},1); \]
\[ x2 = 5 \times x1; \]
\[ \text{om} = [-1 2 3]'; % True process coefficients \]
\[ A = [\text{ones}(\text{N},1) \ x1 \ x2]; % Statistical model \]
\[ y = A \times \text{om} + 0.1 \times \text{randn}(\text{N},1); % Statistical model \]
\[ b = y; \]
\[ w = \text{inv}(A' \times A) \times A' \times b \ % Regression model coefficients \]

This returns

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 3.801412e-018.

\[ w = \begin{bmatrix} -1.0088 & 3.1875 & -1.0408 \end{bmatrix} \]

Singular Value Decomposition & PCA

\[ A_{n \times p} = U_+ \Sigma_+ V_+^T \]

- The \( \Sigma \) matrix can be decomposed as a product of three different matrices

- \( U \) and \( V \) are unitary matrices

\[ \Sigma^T U = I_{n \times n} = U U^T \]
\[ V^T V = I_{p \times p} = V V^T \]

- \( \Sigma \) is a diagonal matrix

\[ \Sigma_{n \times p} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p \end{bmatrix} = \begin{bmatrix} \Sigma_+ \\ 0 \end{bmatrix} \]

- The \( V \) matrix can be written in terms of its column vectors

\[ V_{p \times p} = \begin{bmatrix} v_1 & v_2 & \cdots & v_p \end{bmatrix} \]

- The square of the singular values \( \sigma_i^2 \) represents the 2nd moment of the data along projections of \( A \) onto the vectors \( v_i \)

- The input vectors are rotated to the directions that maximize the estimated second moment of the projected data

\[ v_i = \text{argmax}_{v_1} \| A v_1 \|^2 = (A v_1)^T (A v_1) = v_1 A^T A v_1 \]

- Locating these vectors and their projected variances is called principal components analysis

Example 1: Multicollinearity

\[ N = 20; \]
\[ x1 = \text{rand}(N,1); \]
\[ x2 = 5 \times x1; \]
\[ \text{om} = [-1 2 3]'; % True process coefficients \]
\[ A = [\text{ones}(N,1) \ x1 \ x2]; % Statistical model \]
\[ y = A \times \text{om} + 0.1 \times \text{randn}(N,1); % Statistical model \]
\[ b = y; \]
\[ w = \text{inv}(A' \times A) \times A' \times b \ % Regression model coefficients \]

This returns

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 3.801412e-018.

\[ w = \begin{bmatrix} -1.0088 & 3.1875 & -1.0408 \end{bmatrix} \]
Example 2: PCA without Centering

\[ p_1 = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad p_2 = \begin{bmatrix} V(1:2,1) \times S(1,1)/15 \end{bmatrix} \]

\[ h = \text{DrawArrow}(p_1(1),p_2(1),p_1(2),p_2(2)); \]

\[ \text{set}(h,'\text{HeadStyle}','\text{plain}'); \]

\[ p_1 = \begin{bmatrix} 0 & 0 \end{bmatrix} ; p_2 = \begin{bmatrix} V(1:2,2) \times S(2,2)/15 \end{bmatrix} ; \]

\[ h = \text{DrawArrow}(p_1(1),p_2(1),p_1(2),p_2(2)); \]

\[ \text{set}(h,'\text{HeadStyle}','\text{plain}'); \]

\[ \text{hold off}; \]

\[ \text{xlabel}('x_1'); \]

\[ \text{ylabel}('x_2'); \]

\[ \text{title('Principal Components Analysis Without Centering');} \]

\[ \text{AxisSet}(8); \]

\[ \text{print -depscPCAUncentered.eps;} \]

\[ x_1c = \text{mean}(x_1); \]

\[ x_2c = \text{mean}(x_2); \]

\[ xc = [x_1c \ x_2c]'; \]

\[ A = [x_1-x_1c \ x_2-x_2c]; \]

\[ [U,S,V] = \text{svd}(A); \]

\[ \text{figure} ; \]

\[ \text{FigureSet}(1,5,5); \]

\[ \text{ax} = \text{axes('Position',}[0.1 0.1 0.8 0.8]); \]

\[ h = \text{plot}(x_1,x_2,'.r'); \]

\[ \text{set}(h,'\text{MarkerSize}',6); \]

\[ \text{hold on}; \]

\[ \text{xlim([-0.10 1.00])}; \]

\[ \text{ylim([-0.10 1.00])}; \]

\[ \text{AxisLines}; \]

\[ \text{hold off}; \]

\[ \text{axis('square')}; \]

\[ \text{xlabel('x_1');} \]

\[ \text{ylabel('x_2');} \]

\[ \text{title('Principal Components Analysis With Centering')}; \]

\[ \text{AxisSet}(8); \]

\[ \text{print -depscPCACentered.eps;} \]

Example 2: MATLAB Code

```matlab
function[] = PCACentering();
%clear;
rand('state',0);
randn('state',11);
NP = 100; % Number of points
x1 = 0.08*randn(NP,1); % Input 1
x2 = -x1 + 0.04*randn(NP,1); % Input 2
x1 = x1 + 0.5;
x2 = x2 + 0.5;
A = [x1 x2 ones(NP,1)];
[U,S,V] = svd(A); % Singular Value Decomposition
V(:,1) = V(:,1); % Function in my collection
figure;
FigureSet(1,5,5);
ax = axes('Position',}[0.1 0.1 0.8 0.8]);
hold on;
axis([-0.10 1.00]);
ylim([-0.10 1.00]);
AxisLines;
```

J. McNames Portland State University ECE 4/557 Regularization Ver. 1.27 11
**Principal Components Analysis**

- In general, finding the directions of maximum variance is more useful than finding the directions that maximize the second moment.
- This can be achieved by subtracting the average from all of the input vectors:
  \[ A' = \begin{bmatrix} x'_1 & x'_2 & \ldots & x'_{p-1} \end{bmatrix} \]
  where \( x'_i = x_i - \bar{x}_i \)
- If \( A' \) is decomposed as \( A' = U \Sigma V^T \), then
  \[ \sigma_1^2 = \text{var}(Av_1) \quad \sigma_2^2 = \text{var}(Av_2) \quad \ldots \]
- Note that the column of ones is omitted from \( A' \)
- The vectors \( v_i \) now represent the directions of maximum variance.

**Example 3: PCA With Centering**

**PCA & SVD**

\[ A = U \Sigma V^T \quad A^T A = V^T \Lambda V \]

- Often PCA is calculated using eigenvalues and eigenvectors instead of singular value decomposition.
- It can be shown that
  \[ A^T A = V^T \Lambda V \]
  where
  \[ \Lambda = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_p \end{bmatrix} \]
  This is the same \( V \) matrix as computed using SVD on \( A \)
- The eigenvalues are related to the singular values by \( \lambda_i = \sigma_i^2 \)

**PCA & SVD Summary**

\[ A = U \Sigma V^T \quad A^T A = V^T \Lambda V \]

\[ A = \sum_{i=1}^{p} u_i \sigma_i v_i^T = \sum_{i=1}^{p} \sigma_i \langle u_i v_i^T \rangle \]

- \( A \) can be expressed as a sum of \( p \) rank-1 matrices.
- PCA is useful for compression.
- If most of the variance is captured by the first few principal components, then we can omit the other components with minimal loss of information.
- Just truncate the sum to get an approximation of \( A \)
  \[ A \approx \sum_{i=1}^{\rho} \sigma_i \langle u_i v_i^T \rangle \]
  for some \( \rho < p \)
Let $A$ represent the centered data matrix $w = (A^T A)^{-1} A^T y$

$w = [(U \Sigma V^T)^T (U \Sigma V^T)]^{-1} (U \Sigma V^T)^T y$

$w = [V \Sigma^T U^T U \Sigma V^T]^T y$

$w = [V \Sigma^T U^T U \Sigma V^T]^{-1} V \Sigma^T U^T y$

$w = [V \Sigma^T U^T U \Sigma V^T]^{-1} V \Sigma U^T y$

$w = [V \Sigma^T U^T U \Sigma V^T]^{-1} V \Sigma^2 U^T y$

$w = (V^T)^{-1} \Sigma^2 V^{-1} V \Sigma^2 U^T y$

$w = V \Sigma^2 U^T y$

$w = V \Sigma U^T y$

We can also use PCA to reduce the number of inputs to a smaller set of features.

If we use PCA, the idea is to project the input vectors onto a linear subspace that explains most of the variance of the input vectors.

Practically, this is implemented by performing PCA on the centered version of the $A$ matrix (i.e. $A'$)

The input vectors are then compressed to a set of features $u$ by multiplying each input vector by the $\rho$ columns of $V$.

Recall that $\sigma^2_i$ represents the variance in the direction of $v_i$.

If some of the inputs are nearly collinear, there will be very little variance in some directions.

This will result in some $\sigma_i$ being very small.
Example 4: Principal Components Regression

Consider the following (MATLAB) model

\[ N = 500; \]
\[ P = 4; \]
\[ x1 = \text{randn}(N,1); \]
\[ x2 = \text{randn}(N,1); \]
\[ x3 = x1 + 0.1*\text{randn}(N,1); \]
\[ x4 = x2 + 0.001*\text{randn}(N,1); \]
\[ A = [x1 \ x2 \ x3 \ x4]; \]
\[ A = A - \text{ones}(N,1)*\text{mean}(A); \quad \text{Center: Make the inputs zero mean} \]
\[ w = [2 -2 0 0]'; \]
\[ y = A*w + \text{randn}(N,1); \]

Principal Components Regression (PCR)

- If some \( \sigma_i \) are very small, the weights \( w \) may be very large
- Which can cause \( \hat{y} \) to be very large
- To fix this problem, the sum is often truncated

\[
\hat{y} = x^T \left( \sum_{i=1}^{\rho} \frac{u_i^T y}{\sigma_i} v_i \right)
\]

where \( \rho < p \)
- This is the magic in MATLAB’s \text{pinv} function
- Practically, this has the effect of producing weight vectors with smaller norms (think vector length)
- Sometimes this is called principal components regression

Principal Components Regression Comments

- This method of estimating the weight vector lacks a lot of the properties of the least squares solution
  - Is not Unbiased
  - Does not minimize the sum of the errors squared
  - Is not the maximum likelihood solution
- However, it does reduce the variance of the model outputs
- Can have smaller prediction error
- Is a good idea if the inputs are collinear
- Converting the \( p - 1 \) inputs to a smaller set of \( \rho \) features using PCA is equivalent to principal components regression

MATLAB generated the following solutions.

Least Squares.
\[
\begin{array}{cccc}
2.0364 & -41.3566 & -0.0485 & 39.4472 \\
-41.3566 & -0.9541 & -0.9548 & -0.2042 \\
-0.0485 & -0.0418 & 0.9935 & 1.1310 \\
39.4472 & -0.9545 & -0.9548 & -0.2042 \\
\end{array}
\]

Least Squares After PCA Dimension Reduction.
\[
\begin{array}{cccc}
2.0364 & 2.0304 & 0.9825 & 1.1177 \\
-41.3566 & -0.9541 & -0.9548 & -0.2042 \\
-0.0485 & -0.0418 & 0.9935 & 1.1310 \\
39.4472 & -0.9545 & -0.9548 & -0.2042 \\
\end{array}
\]

Principal Components Regression.
\[
\begin{array}{cccc}
2.0364 & 2.0304 & 0.9825 & 1.1177 \\
-41.3566 & -0.9541 & -0.9548 & -0.2042 \\
-0.0485 & -0.0418 & 0.9935 & 1.1310 \\
39.4472 & -0.9545 & -0.9548 & -0.2042 \\
\end{array}
\]
Example 4: MATLAB Code

```matlab
function [] = Regularization();
clear all;

%-----------------------------
% True Weights
%-----------------------------
N = 500; P = 4;
s1 = randn(N,1);
s2 = randn(N,1);
s3 = s1 + 0.1*randn(N,1);
s4 = s2 + 0.001*randn(N,1);
A = [s1 s2 s3 s4];
A = A - mean(A);% Center: Make the inputs zero mean
w = [2 2 0 0]';
y = A*w + randn(N,1);
fprintf('Process.
');
disp(w)

%-----------------------------
% Least Squares
%-----------------------------
wl = inv(A'*A)*A'*y;% Least squares solution
fprintf('Least Squares.
');
disp(wl)

%-----------------------------
% PCA Dimension Reduction
%-----------------------------
[U,S,V] = svd(A,0);
V4 = V(:,1:4); A4 = A*V4;
V3 = V(:,1:3); A3 = A*V3;
V2 = V(:,1:2); A2 = A*V2;
V1 = V(:,1:1); A1 = A*V1;

wdr4 = V4*inv(A4'*A4)*A4'*y;% PCA Dim Reduction
wdr3 = V3*inv(A3'*A3)*A3'*y;% PCA Dim Reduction
wdr2 = V2*inv(A2'*A2)*A2'*y;% PCA Dim Reduction
wdr1 = V1*inv(A1'*A1)*A1'*y;% PCA Dim Reduction
fprintf('Least Squares After PCA Dimension Reduction.
');
disp([wdr4 wdr3 wdr2 wdr1])

%-----------------------------
% Principal Components Regression
%-----------------------------
SI = inv(S);
for c = 1:size(SI,2),% Orthogonalize other vectors to z
  x = A*V(:,c);
  A(:,c) = x - pinv(A*x'*x)*x'*x;
end;

wpc4 = V*SI*U'*y;% Principal Components Regression rho = 4
wpc3 = V*SI*U'*y;% Principal Components Regression rho = 3
wpc2 = V*SI*U'*y;% Principal Components Regression rho = 2
wpc1 = V*SI*U'*y;% Principal Components Regression rho = 1
fprintf('Principal Components Regression.
');
disp([wpc4 wpc3 wpc2 wpc1])
```

```matlab
%-----------------------------
% PCA Dimension Reduction
%-----------------------------
wi = inv(SI);
for c = 1:size(wi,2),
  x = A*V(:,c);
  A(:,c) = x - pinv(A*x'*x)*x'*x;
end;

wrr1 = inv(A'*A + 0.0000*eye(P,P))*A'*y;% Ridge regression solution
wrr2 = inv(A'*A + 0.0001*eye(P,P))*A'*y;% Ridge regression solution
wrr3 = inv(A'*A + 0.0100*eye(P,P))*A'*y;% Ridge regression solution
wrr4 = inv(A'*A + 1.0000*eye(P,P))*A'*y;% Ridge regression solution
wrr5 = inv(A'*A + 100.0000*eye(P,P))*A'*y;% Ridge regression solution
wrr6 = inv(A'*A + 10000.0000*eye(P,P))*A'*y;% Ridge regression solution
fprintf('RidgeRegression.
');
disp([wrr1 wrr2 wrr3 wrr4 wrr5 wrr6])
```

```matlab
%-----------------------------
% Partial Least Squares
%-----------------------------
An = zeros(size(A));% Normalized data matrix
for c = 1:size(A,2)
  x = A(:,c);
  x = (x - mean(x))/sqrt((x'*x));
  An(:,c) = x;
  end;
y = sqrt((N-1))*std(y);
y = (y-mean(y))/y;
Vpls = zeros(4,size(A,2));
rho = zeros(size(A,2),1);
rho2 = zeros(size(A,2),1);
yh = zeros(size(y));
for r = 1:4
  for c = 1:size(A,2),
    fprintf('Partial Least Squares.
');
    disp(Vpls)
  end;
end;
end;
```

```matlab
%-----------------------------
% Partial Least Squares
%-----------------------------
for c = 1:size(A,2),
  theta = inv(A'*A)*A'*y;% Partial Least Squares
  z = An(:,c);
  theta = theta - pinv(A*theta'*z)*z'*theta;
  end;
end;
wrr4 = V4*inv(A4'*A4)*A4'*y;% PCA Dim Reduction
wrr3 = V3*inv(A3'*A3)*A3'*y;% PCA Dim Reduction
wrr2 = V2*inv(A2'*A2)*A2'*y;% PCA Dim Reduction
wrr1 = V1*inv(A1'*A1)*A1'*y;% PCA Dim Reduction
fprintf('Least Squares After PCA Dimension Reduction.
');
disp([wrr4 wrr3 wrr2 wrr1])
```

```matlab
%-----------------------------
% PCA Dimension Reduction
%-----------------------------
SI = inv(S);
for c = 1:size(SI,2),% Orthogonalize other vectors to z
  x = A*V(:,c);
  A(:,c) = x - pinv(A*x'*x)*x'*x;
end;

wpc4 = V*SI*U'*y;% Principal Components Regression rho = 4
wpc3 = V*SI*U'*y;% Principal Components Regression rho = 3
wpc2 = V*SI*U'*y;% Principal Components Regression rho = 2
wpc1 = V*SI*U'*y;% Principal Components Regression rho = 1
fprintf('Principal Components Regression.
');
disp([wpc4 wpc3 wpc2 wpc1])
```
Stepwise Regression

- Stepwise regression tries to identify the best subset of input variables that will minimize the prediction error.
- This is different than principal components regression because it does not transform the input vectors to a smaller set of features.
- This is an iterative algorithm.
- Not guaranteed to converge to the best subset.
- An exhaustive search through all possible subsets is usually not viable.
- Total number of possible subsets assuming the bias term is always used:
  \[ n_s = \sum_{i=0}^{p-1} \binom{p-1}{i} \]

Ridge Regression

- Ridge regression is another method for handling ill-conditioned matrix inverses.
- Consider a modified error function:
  \[ RE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{i=1}^{p} w_i^2 \]
- This biases the least squares solution by penalizing large coefficients.
- The model coefficients that minimize the RE are given by:
  \[ w = (A^T A + \alpha I)^{-1} A^T y \]
- For large \( \alpha \), the inverse becomes similar to the identity matrix.
- For small \( \alpha \), the solution becomes similar to the least squares coefficients.

Example 5: Ridge Regression

Using the same model as in the previous example:

- `wrr1 = inv(A'*A + 0.0000*eye(P,P))*A'*y;` % Ridge regression solution
- `wrr2 = inv(A'*A + 0.0001*eye(P,P))*A'*y;` % Ridge regression solution
- `wrr3 = inv(A'*A + 0.0100*eye(P,P))*A'*y;` % Ridge regression solution
- `wrr4 = inv(A'*A + 1.0000*eye(P,P))*A'*y;` % Ridge regression solution
- `wrr5 = inv(A'*A + 100.0000*eye(P,P))*A'*y;` % Ridge regression solution
- `wrr6 = inv(A'*A + 10000.0000*eye(P,P))*A'*y;` % Ridge regression solution

MATLAB generated the following coefficients:

```
Ridge Regression.
2.0364 2.0347 2.0261 1.7173 0.9204 0.0927
-41.3566 -30.3511 -2.0053 -0.9642 -0.8689 -0.0876
-0.0485 -0.0466 -0.0376 0.2656 0.8824 0.0932
39.4472 28.4419 0.0966 -0.9428 -0.8687 -0.0876
```

Stepwise Regression Algorithm

1. Start with no variables in the model: \( \hat{y} = \bar{y} = w_0 \)
2. Search through all of the variables not in the model and find the variable that most decreases the ASE. For this variable, call it \( x_i \), calculate the following statistic:
   \[ F^*_i = \frac{\text{SSE}_{i-1} - \text{SSE}_i}{\text{SSE}_i} \]
   If \( F^*_i \) is greater than some threshold \( t_a \), add the variable \( x_i \) to the model.
3. For each of the variables in the model, calculate the \( F^*_i \) statistic above. If the smallest value for this statistic is less than a threshold \( t_d \), drop the variable from the model.
4. Loop to 2 until there are no variables added or removed.
Example 6: Stepwise Regression Comments

- Typical thresholds are $t_a = 4.0$ and $t_d = 3.9$
- The algorithm will only work if $t_d < t_a$
- Each test is performing a statistical test to determine whether $\omega_i = 0$ or not
- There are a number of variations on this algorithm
  - **Forward Selection**: Same as stepwise regression, but Step 3 is omitted
  - **Backward Elimination**: Begins with all variables in the model and then drops variables using stepwise regression with Step 2 omitted
- Sometimes the model size $p$ is specified, rather than thresholds
- The function is posted on the class web site

Stepwise Regression

Consider the following model:

$$
x_6 = \text{randn}(N,1);
x_5 = \text{randn}(N,1);
x_4 = x_6 + 0.5*\text{randn}(N,1);
x_3 = x_5 + 0.5*\text{randn}(N,1);
x_2 = x_4 + x_3 + 0.5*\text{randn}(N,1);
x_1 = x_6 + x_4 + 0.5*\text{randn}(N,1);
y = 5*x_6 + 5*x_5 + \text{randn}(N,1);
y = y - \text{mean}(y);
$$

Stepwise regression generated the following sequence of steps:

Example 6: Stepwise Regression

<table>
<thead>
<tr>
<th>Step 1 - Add</th>
<th>Step 2 - Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added variable: 2</td>
<td>Removed variable: None</td>
</tr>
<tr>
<td>F Statistic: 346.70</td>
<td>F Statistic: 109.61</td>
</tr>
<tr>
<td>Add threshold: 4.00</td>
<td>Drop threshold: 3.90</td>
</tr>
<tr>
<td>SSE Reduction: 0.69</td>
<td>SSE Increase: 0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 1 - Drop</th>
<th>Step 2 - Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Removed variable: None</td>
<td>Added variable: 5</td>
</tr>
<tr>
<td>F Statistic: 346.01</td>
<td>F Statistic: 356.25</td>
</tr>
<tr>
<td>Drop threshold: 3.90</td>
<td>Add threshold: 4.00</td>
</tr>
<tr>
<td>SSE Increase: 0.00</td>
<td>SSE Reduction: 0.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2 - Add</th>
<th>Step 3 - Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added variable: 6</td>
<td>Removed variable: 2</td>
</tr>
<tr>
<td>F Statistic: 33.67</td>
<td>F Statistic: 0.18</td>
</tr>
<tr>
<td>Add threshold: 4.00</td>
<td>Drop threshold: 3.90</td>
</tr>
<tr>
<td>SSE Reduction: 0.22</td>
<td>SSE Increase: -0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3 - Add</th>
<th>Step 4 - Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added variable: None</td>
<td>Removed variable: None</td>
</tr>
<tr>
<td>F Statistic: 3.01</td>
<td>F Statistic: 12698.91</td>
</tr>
<tr>
<td>Add threshold: 4.00</td>
<td>Drop threshold: 3.90</td>
</tr>
<tr>
<td>SSE Reduction: 0.01</td>
<td>SSE Increase: 0.00</td>
</tr>
</tbody>
</table>

Least Squares:

-0.0309
0.0266
-0.1855
-0.0628
5.1929
5.1362

Stepwise Regression (Inputs, Weights):

6.0000 5.0161
5.0000 5.0249
Example 6: MATLAB Code

```matlab
function [] = StepwiseRegression();
N = 500; % Indices of variables in the model
P = 6; % % Index of variable dropped from model
% Is there a change in the model? 1 = yes, 0 = no
% Center: Make the inputs zero mean
T = 4.0; % % Drop threshold
Ta = 4.0; % Add threshold
Td = 3.9; % Drop threshold
xi = []; % Indices of variables in the model
Ab = []; % Model output
SSE = sum((y-yh).^2); % Best SSE so far with xi inputs
ASE = SSE/(N-length(xi)); % Average squared error

% Initialize best F statistic
Fb = 0;

while chg,
    chg = 0; % Initialize flag as no change this loop
    %--------
    % Add variable loop
    %--------
    % Initialize best F statistic
    Fb = 0;
    for cnt = 1:P,
        if isempty(xi) & any(xi==cnt), % If variable is already in model, skip
            continue;
        end;
        xin = [xi;cnt]; % New (test) xi
        An = A(:,xin); % New A matrix (with extra input)
        yhb = An*inv(An'*An)*An'*y; % Model output
        SSE = SSE/(N-length(xi)); % New error sum of squares
        ASE = SSE/(N-length(xi)); % New averaged squared error
        if Fb>Ta,% If Fb is greater than add threshold,
            xi = xin; % Add the variable
            Ab = A(:,xi); % Best A matrix so far
            yhb = Ab*inv(Ab'*Ab)*Ab'*y; % Model output
            ASEb = SSE/(N-length(xi)); % Average squared error
            fprintf('Step %d - Add
',cnt);
            fprintf('------------------------
');
            fprintf('Added variable: %7d
',xi(length(xi)));
            fprintf('F Statistic : %7.2f
',Fb);
            fprintf('Add threshold : %7.2f
',Ta);
            fprintf('SSE Reduction : %7.2f
',(SSE-SSEb)/SSE);
            SSE = SSEb; ASE = ASEb; chg = 1;% Set the change flag
        else% Else, leave xi alone
            fprintf('Step %d - Drop
',cnt);
            fprintf('------------------------
');
            fprintf('Removed variable:None
');
            fprintf('F Statistic : %7.2f
',Fb);
            fprintf('Drop threshold : %7.2f
',Td);
            fprintf('SSE Increase : %7.2f
',(SSE-SSEb)/SSE);
            SSE = SSEb; ASE = ASEb; chg = 1;% Set the change flag
        end;
    end;
    chg = 1;% Is there a change in the model? 1 = yes, 0 = no
    %--------
    % Drop variable loop
    %--------
    % Initialize best F statistic
    Fb = 0;
    for cnt = 1:length(xi),
        xin = [xi(1:cnt-1);xi(cnt+1:length(xi))]; % New (test) xi
        An = A(:,xin); % New A matrix (with extra input)
        yhb = An*inv(An'*An)*An'*y; % Model output
        SSE = SSE/(N-length(xi)); % New error sum of squares
        ASE = SSE/(N-length(xi)); % New averaged squared error
        if Fb<Td,% If Fb is less than drop threshold,
            xi = xin; % Add the variable
            Ab = A(:,xi); % Best A matrix so far
            yhb = Ab*inv(Ab'*Ab)*Ab'*y; % Model output
            ASEb = SSE/(N-length(xi)); % Average squared error
            fprintf('Step %d - Drop
',cnt);
            fprintf('------------------------
');
            fprintf('Removed variable: %7d
',xi(cnt));
            fprintf('F Statistic : %7.2f
',Fb);
            fprintf('Drop threshold : %7.2f
',Td);
            fprintf('SSE Increase : %7.2f
',(SSE-SSEb)/SSE);
            SSE = SSEb; ASE = ASEb; chg = 1;% Set the change flag
        else% Else, leave xi alone
            fprintf('Step %d - Drop
',cnt);
            fprintf('------------------------
');
            fprintf('Removed variable:None
');
            fprintf('F Statistic : %7.2f
',Fb);
            fprintf('Drop threshold : %7.2f
',Td);
            fprintf('SSE Increase : %7.2f
',(SSE-SSEb)/SSE);
            SSE = SSEb; ASE = ASEb; chg = 1;% Set the change flag
        end;
    end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
```
Prediction Error Estimates & Cross-Validation Error

- For the least squares solution we saw that ASE is an unbiased, minimum variance estimate of the prediction error.
- Prediction error is defined as the expected error observed on new input vectors not in the data set:
  \[ E[(y - \hat{y})^2] \]
- ASE is not unbiased when the solution is biased.
- An alternative measure that is usually very accurate is the leave-one-out cross-validation error.
- Also called the PRESS (PRediction Error Sum of Squares) statistic.
- Definition:
  \[
  \text{CVE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2
  \]
  where \( \hat{y}_i \) is a linear model constructed with the \( i \)th point omitted from the data set.

Cross-Validation Error

- It appears that calculating the CVE would require building \( n \) different models with each data set containing \( n - 1 \) points.
- This would be computationally prohibitive.
- Although each model would be different, they would be similar since the data sets only differ by a single point.
- It turns out there is a very efficient way of calculating the CVE:
  \[
  \hat{y} = Aw = A(A^T A)^{-1} A^T y = H y
  \]
  where \( H = A(A^T A)^{-1} A^T \) is the hat matrix.
- The leave-one-out error can be computed with:
  \[
  d_i = y_i - \hat{y}_i = \frac{e_i}{1 - h_{ii}}
  \]
  where \( e_i = y_i - \hat{y}_i \) and \( h_{ii} \) is the \( i \)th element on the diagonal of the hat matrix.
Cross-Validation Error Continued

- This dramatically reduces computation required
- Can be used with the three biased methods discussed so far

Regularization Summary

- Principal Components Regression
  \[ \hat{y} = Aw = A(V\tilde{\Sigma}_+^{-1}U_+^T)y \]
  \[ H = AV\tilde{\Sigma}_+U_+^T = U_+\Sigma_+\tilde{\Sigma}_+^{-1}U_+^T \]
  Note that if \( \tilde{\Sigma}_+ = \Sigma_+ \), \( H = U_+U_+^T \neq UU^T = I \)

- Ridge Regression
  \[ \hat{y} = Aw = A(A^TA + \alpha I)^{-1}A^Ty \]
  \[ H = A(A^TA + \lambda I)^{-1}A^T \]

- Stepwise Regression
  \[ \hat{y} = A_R w_R = A_R(A_R^TA_R)^{-1}A_R^Ty \]
  \[ H = A_R(A_R^TA_R)^{-1}A_R^T \]