Harrison Lam

Variable Selection and the Bias-Variance Tradeoff

In this project, several variable selection methods and the bias-variance tradeoff are investiaged in the following linear model:

$Y = X\beta + \epsilon, \qquad \epsilon \sim N(0, \sigma^2)$

Notation:

Training dataset $\mathcal{T} = \{(y_t, x_t), t = 1, \dots, N\}$, x_t is a 1 x p row vector

In matrix notation, $Y = (y_t)_{t=1:N}$ N x 1 row vector, $X = (x_{ti})$ N x p matrix

Least-squares (LS) estimator = $\hat{\beta}^{LS}(\mathcal{T}) = \underset{\hat{\beta}}{\operatorname{argmin}}\, RSS(\hat{\beta}, \mathcal{T}) = \underset{\hat{\beta}}{\operatorname{argmin}} \frac{1}{N}\sum_{t=1}^{N}(y_t - x_t\hat{\beta})^2$

Assuming $N > p + 1$ (invertibility condition), $\hat{\beta}^{LS}(\mathcal{T}) = (X^T X)^{-1} X^T Y$

$\hat{\beta}_j^{\mathcal{M}}(\mathcal{T}) = \hat{\beta}_j^{LS}(\mathcal{T}^{\mathcal{M}})$ where $\mathcal{M} \subseteq \{1, \dots, p\}$

For a fixed location u, the expected square prediction error can be decomposed (given $y \perp \mathcal{T}$):

$$\begin{aligned}
\mathbb{E}_{\mathcal{T}}\left[(y - u\hat{\beta})^2\right] &= \mathbb{E}_{\mathcal{T}}\left[y^2 - 2yu\hat{\beta} + \hat{\beta}^T u^T u\hat{\beta}\right] \ where\ y = u\beta + \epsilon \\
&= Var_{\mathcal{T}}(y) + \mathbb{E}_{\mathcal{T}}[y]^2 + \mathbb{E}_{\mathcal{T}}\left[-2yu\hat{\beta} + \hat{\beta}^T u^T u\hat{\beta}\right] \\
&= \sigma^2 + \beta^T u^T u\beta \underbrace{-2u\beta\mathbb{E}_{\mathcal{T}}\left[u\hat{\beta}\right]}_{since\ y \perp \mathrm{T}} + \mathbb{E}_{\mathcal{T}}\left[\hat{\beta}^T u^T u\hat{\beta}\right] \\
&= \sigma^2 + \left(u\beta - \mathbb{E}_{\mathcal{T}}\left[u\hat{\beta}\right]\right)^2 + \mathbb{E}_{\mathcal{T}}\left[\hat{\beta}^T u^T u\hat{\beta}\right] - \mathbb{E}_{\mathcal{T}}\left[u\hat{\beta}\right]^2 \\
&= \underbrace{\sigma^2}_{irreducible\ variance} + \underbrace{\left(u\beta - \mathbb{E}_{\mathcal{T}}\left[u\hat{\beta}\right]\right)^2}_{bias} + \underbrace{Var_{\mathcal{T}}(u\hat{\beta})}_{variance}
\end{aligned}$$

Suppose $X^T X = I_p$, and the p[th] covariate is deleted:

σ^2 constant

$\hat{\beta} = \hat{\beta}^{LS}$:
$\hat{\beta} \sim N(\beta, \sigma^2(x^T x)^{-1}) = N(\beta, \sigma^2 I_p)$
$\Rightarrow bias^2 = 0, variance = \sigma^2(u_1^2 + \cdots + u_p^2)$

$\hat{\beta} = \hat{\beta}^{\{1,\dots,p-1\}}$:
$(\widehat{\beta_1}, \dots, \widehat{\beta_{p-1}}) \sim N\left(\beta_{(-p)}, \sigma^2\left(x_{(-p)}^T x_{(-p)}\right)^{-1}\right)$
$= N(\beta_{(-p)}, \sigma^2 I_{p-1})\ where\ x_{-p} = x\ with\ p^{th} column\ removed$
$\widehat{\beta_p} = 0$
$\Rightarrow bias^2 = u_p^2\beta_p^2, variance = \sigma^2(u_1^2 + \cdots + u_{p-1}^2)$

Hence
$\beta_p = 0 \Rightarrow bias^2\ stays\ at\ 0, variance\ decreases\ by\ \sigma^2 u_p^2$
$\beta_p \neq 0 \Rightarrow bias^2\ increases\ from\ 0\ to\ u_p^2\beta_p^2, variance\ decreases\ by\ \sigma^2 u_p^2$

Using the linear model with $p = 10, \sigma^2 = 1, X \sim N(0, I_p)$ with $\beta = (-0.5, 0.45, -0.4, 0.35, -0.3, 0.25, -0.2, 0.15, -0.1, 0.05)^T$, a training dataset is simulated with $N_{training} = 30$ and $N_{test} = 1000$ using the mvnrnd function.

```
%Generating training data
trainingX = mvnrnd(mu, eye(p), trainingSize);
trainingNoise = mvnrnd(zeros(1, trainingSize), sigmaSquared*eye(trainingSize), 1);
trainingY = trainingX * beta + trainingNoise';
%Generating test data
testX = mvnrnd(mu, eye(p), testSize);
testNoise = mvnrnd(zeros(1, testSize), sigmaSquared*eye(testSize), 1);
testY = testX * beta + testNoise';
```

For $\mathcal{M}_1 = \{1\}, \mathcal{M}_2 = \{1,2\}, \dots, \mathcal{M}_p = \{1, \dots, p\}$, the estimate $\hat{\beta}^{\mathcal{M}_j}$ and the corresponding RSS are then calculated using the given formula.

```
%Calculating estimate
betaEstimateMatrix = zeros(p);
trainingRSS = zeros(1, p);
testRSS = zeros(1, p);
for j = 1:p
    reducedX = trainingX(:, 1:j);
    reducedBetaLS = (reducedX'*reducedX)\(reducedX'*trainingY);
    betaEstimateMatrix(j, :) = [reducedBetaLS' zeros(1, p-j)];
    trainingRSS(j) = (1/trainingSize)*(norm(trainingY-reducedX*reducedBetaLS))^2;
    testRSS(j) = (1/testSize)*(norm(testY-testX(:, 1:j)*reducedBetaLS))^2;
end
```

This is repeated 100 times to calculate the average training and test RSS, which is then plotted against j (figure 1)

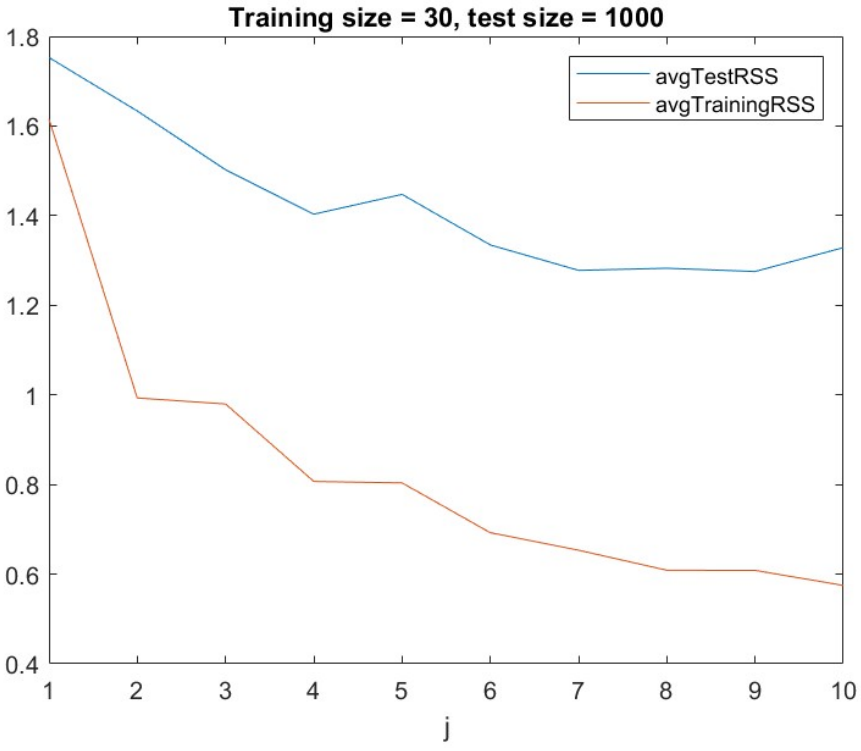

Figure 1: plot of average test/training RSS against model size for $N_{tr} = 30$

This is also repeated with $N_{tr} = 200$ (figure 2)

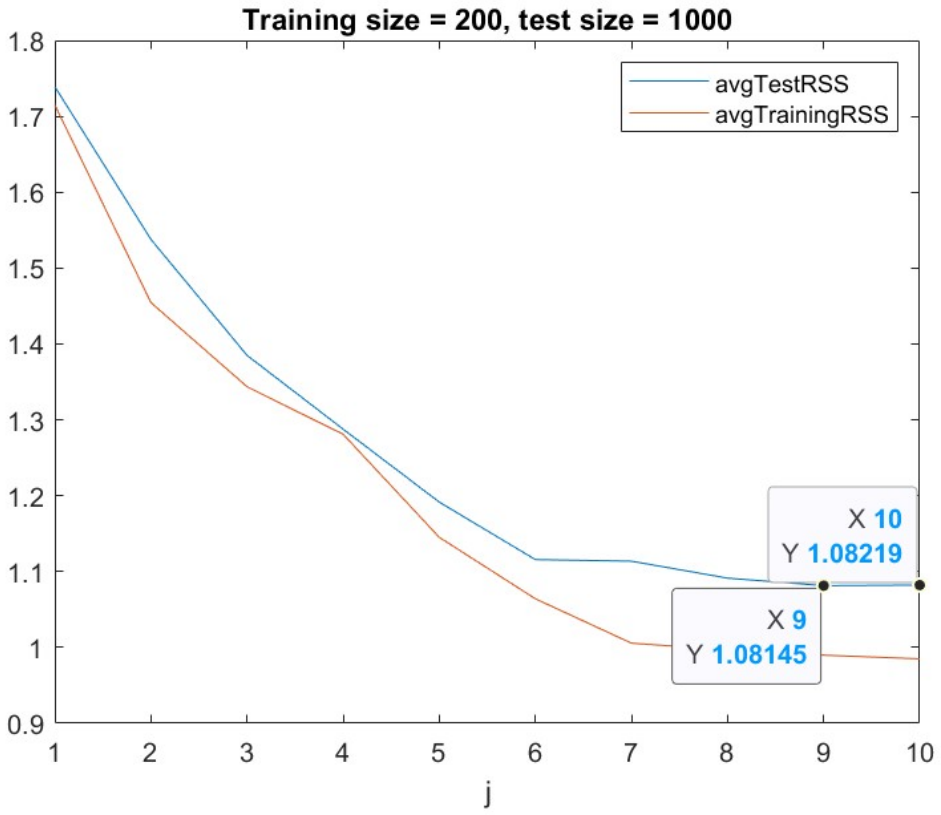

Figure 2: plot of average test/training RSS against model size for $N_{tr} = 200$

From figures 1, 2 above, it could be seen that a larger training size results in a greater average training RSS for any model size. Also, for training size = 200, j = 9 had a smaller average test RSS than j = 10, which suggests that a less complex model can give improve prediction accuracy sometimes (this is more apparent in the case with training size = 30).

Subset Selection

The best subset selection method selects the best performing model (with the lowest RSS) for a given model size. The function bestsubset.m below takes the training dataset as input and outputs a p x p matrix with the j[th] column containing $\hat{\beta}^{\mathcal{M}_j}$ where

$$\mathcal{M}_j(\mathcal{T}) = \underset{\mathcal{M}:\, \|\mathcal{M}\|=j}{\operatorname{argmin}} RSS\left(\hat{\beta}^{\mathcal{M}}(\mathcal{T}), \mathcal{T}\right)$$

```
function [B] = bestsubset(T)
    dim = size(T);
    N = dim(1);
    p = dim(2)-1;
    B = zeros(p);
    y = T(:, 1);
    x = T(:, 2:dim(2));
    for modelSize = 1:p
        %Finding the best subset of size modelSize
        models = nchoosek(1:p, modelSize);
        numModels = length(models(:,1));
        bestModel = models(1,:);
        %Looping through all the models of size modelSize
        for j = 1:numModels
            model = models(j,:);
            reducedX = x(:, model);
            beta = (reducedX'*reducedX)\(reducedX'*y);
            RSS = (1/N)*(norm(y-reducedX*beta))^2;
            if j == 1
                min = RSS;
                bestModelBeta = beta;
            else
                %Comparing RSS of current model to smallest RSS so far
                if RSS < min
                    min = RSS;
                    bestModel = models(j,:);
                    bestModelBeta = beta;
                end
            end
        end
        %Setting the corresponding values of B to values of beta^(M_j)
        for j = 1:modelSize
            B(bestModel(j), modelSize) = bestModelBeta(j);
        end
    end
end
```

Figure 3: bestsubset.m

The following code is used to test the function using the normally generated data from page 2. The output from the function is then compared to reducedBetaLS computed previously.

```
B = bestsubset([trainingY trainingX]);
disp(B);
disp(reducedBetaLS);
```

```
>> main
         0   -0.4457   -0.4846   -0.4693   -0.4791   -0.5030   -0.5220   -0.5116   -0.5120   -0.5095
    0.5172    0.5213    0.5233    0.5109    0.5358    0.5058    0.4882    0.4883    0.4907    0.4812
         0         0         0   -0.3391   -0.3384   -0.3496   -0.3467   -0.3560   -0.3459   -0.3512
         0         0         0         0         0    0.2719    0.2620    0.2601    0.2554    0.2537
         0         0   -0.3712   -0.3504   -0.3659   -0.3844   -0.4006   -0.4013   -0.4022   -0.3993
         0         0         0         0    0.2825    0.2761    0.2862    0.2849    0.2898    0.2904
         0         0         0         0         0         0   -0.2572   -0.2434   -0.2336   -0.2407
         0         0         0         0         0         0         0    0.0938    0.1001    0.1019
         0         0         0         0         0         0         0         0   -0.0844   -0.0839
         0         0         0         0         0         0         0         0         0    0.0725

   -0.5095
    0.4812
   -0.3512
    0.2537
   -0.3993
    0.2904
   -0.2407
    0.1019
   -0.0839
    0.0725
```

Figure 4: best k-regressor estimators and LS estimator

As seen from figure 4 above, the function gives a k-regressor estimator in column k and produces the same result for beta when all regressors are used, which suggests that the function works properly.

$\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, \dots, p\}\}$ has size $2^p - 1$ since each regressor is either included or excluded (minus the empty set)

$\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, \dots, p\}, |M| = j\}$ *has size* $\binom{p}{j}$

For fixed p, this is maximised when $j = \frac{p-1}{2}$ $(p\ odd)$ or $j = \frac{p}{2}$ $(p\ even)$

Using the MATLAB nchoosek function:

$$\binom{18}{9} = 48620, \binom{19}{9} = 92378, \binom{20}{10} = 184756$$

Hence, the smallest value of p with the search space exceeding 10^5 is 20.

The greedy subset selection method incrementally builds up the model sequence by adding the covariate that improves the RSS the most at each step.

```
function [B] = greedysubset(T)
    dim = size(T);
    N = dim(1);
    p = dim(2)-1;
    B = zeros(p);
    y = T(:, 1);
    x = T(:, 2:dim(2));

    %Finding the first regressor with lowest RSS
    RSSArray = zeros(1, p);
    for j = 1:p
        reducedX = x(:, j);
        beta = (reducedX'*reducedX)\(reducedX'*y);
        RSSArray(j) = (1/N)*(norm(y-reducedX*beta))^2;
    end
    [~, minIndex] = min(RSSArray);
    currentModel = minIndex;
    reducedX = x(:, currentModel);
    beta = (reducedX'*reducedX)\(reducedX'*y);
    B(currentModel, 1) = beta;

    while length(currentModel) < p
        numReg = length(currentModel);
        RSSArray = zeros(1, p-numReg);
        %Finding all the models containing M_d, with one extra regressor
        greedyModels = setdiff(1:p, currentModel)';
        greedyModels = [greedyModels repmat(currentModel, [p-numReg 1])];
        greedyModels = sort(greedyModels, 2);
        %Looping through all the models in greedyModels
        for j = 1:p-numReg
            model = greedyModels(j,:);
            reducedX = x(:, model);
            beta = (reducedX'*reducedX)\(reducedX'*y);
            RSSArray(j) = (1/N)*(norm(y-reducedX*beta))^2;
        end
        %Displaying RSS for each added covariate
        disp([setdiff(1:p, currentModel); RSSArray]);
        %Finding the regressor that gives the smallest RSS
        [~, minIndex] = min(RSSArray);
        currentModel = greedyModels(minIndex,:);
        %Calculating beta again
        reducedX = x(:, currentModel);
        beta = (reducedX'*reducedX)\(reducedX'*y);
        %Storing the values in B
        for j = 1:length(currentModel)
            B(currentModel(j), length(currentModel)) = beta(j);
        end
    end
end
```

Figure 5: greedysubset.m

Similar to bestsubset.m, this function is tested using the training data from page 2. RSSArray is displayed at every step to check that the covariate that gives the lowest RSS is selected.

```
>> main
    1.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000   10.0000
    1.4545    1.5628    1.6522    1.5898    1.6226    1.6568    1.6670    1.6748    1.6854

    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000   10.0000
    1.3436    1.3988    1.3164    1.3805    1.4071    1.4447    1.4383    1.4528

    3.0000    4.0000    6.0000    7.0000    8.0000    9.0000   10.0000
    1.2211    1.2469    1.2311    1.2574    1.3047    1.2993    1.3158
```

Figure 6: RSSArray for the first 3 steps

From figure 6, it could be seen that regressor 1 should be selected (in the second step), then regressor 5, regressor 3 and so on.

```
         0   -0.4457   -0.4846   -0.4693   -0.4791   -0.5030   -0.5220   -0.5116   -0.5120   -0.5095
    0.5172    0.5213    0.5233    0.5109    0.5358    0.5058    0.4882    0.4883    0.4907    0.4812
         0         0         0   -0.3391   -0.3384   -0.3496   -0.3467   -0.3560   -0.3459   -0.3512
         0         0         0         0         0    0.2719    0.2620    0.2601    0.2554    0.2537
         0         0   -0.3712   -0.3504   -0.3659   -0.3844   -0.4006   -0.4013   -0.4022   -0.3993
         0         0         0         0    0.2825    0.2761    0.2862    0.2849    0.2898    0.2904
         0         0         0         0         0         0   -0.2572   -0.2434   -0.2336   -0.2407
         0         0         0         0         0         0         0    0.0938    0.1001    0.1019
         0         0         0         0         0         0         0         0   -0.0844   -0.0839
         0         0         0         0         0         0         0         0         0    0.0725
```

Figure 7: output matrix B

This is consistent with the output matrix B above (figure 7), which shows that after the first step, the regressors selected are 1, 5, then 3 and so on.

Since $\mathcal{M}_0, \ldots, \mathcal{M}_p$ are nested, at the d[th] step, only the models of size d+1 containing $\mathcal{M}_d$ have to be checked (instead of all the models), which increases computational efficiency.

Using an F-test at the p = 0.05 level and the following F-statistic, an alternative greedy subset method can be used, which stops when adding another covariate does not significantly improve RSS:

$$\frac{RSS\left(\hat{\beta}^{\mathcal{M}_d}\right) - RSS\left(\hat{\beta}^{\mathcal{M}_{d+1}}\right)}{RSS\left(\hat{\beta}^{\mathcal{M}_{d+1}}\right)/(N-d-1)} \sim F_{1,N-d-1}$$

The F-statistic is then calculated using the given formula and compared to the p=0.05 critical value of the F-distribution.

```
[minRSS, minIndex] = min(RSSArray);
fStat = (currentRSS-minRSS)/(minRSS/(N-numReg-1));
prob = fcdf(fStat, 1, N-numReg-1);
if prob > 0.95
    currentModel = greedyModels(minIndex,:);
    %Calculating beta again
    reducedX = x(:, currentModel);
    beta = (reducedX'*reducedX)\(reducedX'*y);
    %Storing the values in B
    for j = 1:length(currentModel)
        B(currentModel(j), length(currentModel)) = beta(j);
    end
    currentRSS = minRSS;
else
    reg = B(:, numReg);
    for k = numReg+1:p
        B(:,k) = reg;
    end
    break
end
```

Using the previous training data again, it could be seen that the probability $\mathbb{P}(F < fStat)$ is greater than 0.95 up until the 8th step (figure 8 below). This means that the procedure should stop at step 7 and this is consistent with the output matrix B in figure 9.

This method does not work for best subset selection. This is because the models are not necessarily nested, so $RSS\left(\hat{\beta}^{\mathcal{M}_{d+1}}\right)$ could potentially be greater than $RSS\left(\hat{\beta}^{\mathcal{M}_d}\right)$. This shows that the given statistic does not follow an F-distribution and the F-test cannot be used.

```
prob =

    1.0000

prob =

    1.0000

prob =

    0.9999

prob =

    0.9998

prob =

    0.9996

prob =

    0.9990

prob =

    0.8097
```

Figure 8: $\mathbb{P}(F < fStat)$ for each step

```
         0   -0.4457   -0.4846   -0.4693   -0.4791   -0.5030   -0.5220   -0.5220   -0.5220   -0.5220
    0.5172    0.5213    0.5233    0.5109    0.5358    0.5058    0.4882    0.4882    0.4882    0.4882
         0         0         0   -0.3391   -0.3384   -0.3496   -0.3467   -0.3467   -0.3467   -0.3467
         0         0         0         0         0    0.2719    0.2620    0.2620    0.2620    0.2620
         0         0   -0.3712   -0.3504   -0.3659   -0.3844   -0.4006   -0.4006   -0.4006   -0.4006
         0         0         0         0    0.2825    0.2761    0.2862    0.2862    0.2862    0.2862
         0         0         0         0         0         0   -0.2572   -0.2572   -0.2572   -0.2572
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
```

Figure 9: output matrix B

The function crossval shown below (figure 10) takes T and sparse as input, where sparse is a function that takes training data as input and outputs a regression coefficient matrix B (e.g. bestsubset, greedysubset). Candidates are selected based on the lowest prediction error, which is estimated using 10-fold cross-validation.

```
function [betaCV] = crossval(T, sparse)
    dim = size(T);
    N = dim(1);
    p = dim(2)-1;
    y = T(:, 1);
    x = T(:, 2:dim(2));

    %Generating random permutation and permuting data by row
    perm = randperm(N);
    y = y(perm, :);
    x = x(perm, :);
    PE = zeros(1, p);

    B = sparse(T);
    for j = 1:p
            prevEnd = 0;
            sum = 0;
        for k = 1:10
            %Splitting T into T^k and T^-k
            testN = prevEnd+1:(N*k/10);
            testY = y(testN, :);
            testX = x(testN, :);
            trainN = setdiff(1:N, testN);
            trainY = y(trainN, :);
            trainX = x(trainN, :);
            prevEnd = floor(N*k/10);
            %Finding beta for training data T^-k
            betaMatrix = sparse([trainY trainX]);
            beta = betaMatrix(:, j);
            %Calculating RSS
            RSS = (1/N)*(norm(testY-testX*beta))^2;
            sum = sum + RSS;
        end
        PE(j) = sum/10;
    end
    %Finding and returning candidate with lowest PE
    [~, minIndex] = min(PE);
    betaCV = B(:, minIndex);
end
```

Figure 10: crossval.m

To test this function, the data from page 2 and bestsubset is used.

```
betaCV = crossval([trainingY trainingX], @bestsubset)
```

As shown below in figure 11, the function handle is working properly since the matrix B is the same as the output matrix in figure 4. The PE for each column of regressors is then calculated and stored in the PE array. As the lowest PE is achieved when j = 7, betaCV is set to the 7^{th} candidate and returned, as required.

```
B =

         0   -0.4457   -0.4846   -0.4693   -0.4791   -0.5030   -0.5220   -0.5116   -0.5120   -0.5095
    0.5172    0.5213    0.5233    0.5109    0.5358    0.5058    0.4882    0.4883    0.4907    0.4812
         0         0         0   -0.3391   -0.3384   -0.3496   -0.3467   -0.3560   -0.3459   -0.3512
         0         0         0         0         0    0.2719    0.2620    0.2601    0.2554    0.2537
         0         0   -0.3712   -0.3504   -0.3659   -0.3844   -0.4006   -0.4013   -0.4022   -0.3993
         0         0         0         0    0.2825    0.2761    0.2862    0.2849    0.2898    0.2904
         0         0         0         0         0         0   -0.2572   -0.2434   -0.2336   -0.2407
         0         0         0         0         0         0         0    0.0938    0.1001    0.1019
         0         0         0         0         0         0         0         0   -0.0844   -0.0839
         0         0         0         0         0         0         0         0         0    0.0725
```

Figure 11: output matrix of crossval.m

```
PE =

    0.1850    0.1477    0.1385    0.1412    0.1335    0.1210    0.1087    0.1130    0.1114    0.1097
```

```
betaCV =

   -0.5220
    0.4882
   -0.3467
    0.2620
   -0.4006
    0.2862
   -0.2572
         0
         0
         0
```

Lasso estimator

The Lasso estimator includes an L1 penalty term in addition to the RSS:

$$\hat{\beta}^{(L,\lambda)}(\mathcal{T}) = \underset{\hat{\beta}}{\operatorname{argmin}} \left\{ RSS(\hat{\beta}, \mathcal{T}) + \lambda \sum_{j=1}^{p} |\hat{\beta}_j| \right\} \ (*)$$

$Let\ \hat{\beta}^{(L,\lambda)}(\mathcal{T})\ solve\ (*)$
$\Rightarrow RSS\left(\hat{\beta}^{(L,\lambda)}(\mathcal{T}); \mathcal{T}\right) + \lambda \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1 \leq RSS(\beta; \mathcal{T}) + \lambda |\beta|_1 \ \forall \beta$
$\Rightarrow RSS\left(\hat{\beta}^{(L,\lambda)}(\mathcal{T}); \mathcal{T}\right) < RSS(\beta; \mathcal{T}) \ \forall \beta\ with\ |\beta|_1 < \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1$
$\hat{\beta}^{(L,\lambda)}(\mathcal{T}) = \underset{\beta}{\operatorname{argmin}} RSS(\beta, \mathcal{T}), |\beta|_1 \leq \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1$

$Let\ \hat{\beta} = \underset{\beta}{\operatorname{argmin}} RSS(\beta, \mathcal{T}), |\beta|_1 \leq \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1\ where\ \hat{\beta}^{(L,\lambda)}(\mathcal{T})\ solves\ (*)$
$\Rightarrow RSS\left(\hat{\beta}; \mathcal{T}\right) \leq RSS\left(\hat{\beta}^{(L,\lambda)}(\mathcal{T}); \mathcal{T}\right), \left|\hat{\beta}\right|_1 \leq \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1$
$\Rightarrow RSS\left(\hat{\beta}; \mathcal{T}\right) + \lambda \left|\hat{\beta}\right|_1 \leq RSS\left(\hat{\beta}^{(L,\lambda)}(\mathcal{T}); \mathcal{T}\right) + \lambda \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1$
$\Rightarrow RSS\left(\hat{\beta}; \mathcal{T}\right) + \lambda \left|\hat{\beta}\right|_1 \leq RSS(\beta; \mathcal{T}) + \lambda |\beta|_1 \ \forall \beta$
$\Rightarrow \hat{\beta}\ solves\ (*)$

So $(*)$ is equivalent to minimising $RSS(\beta, \mathcal{T})$, which is a quadratic in β, with constraints $|\beta|_1 \leq \left|\hat{\beta}^{(L,\lambda)}(\mathcal{T})\right|_1$
The given program monotonic_lars.m, the LARS algorithm, computes a Lasso solution for each model size.

Consider the given prostate cancer dataset, prostate.dat. The dataset contains the following covariates: lpsa, lcavol, lweight, age, lbph, svi, lcp, gleason and pgg45. Four independent and randomly generated normal variables are added to the dataset. After separating the dataset into a training dataset of size 70 and testing dataset of size 27, and using the training dataset, an output matrix B is obtained for each of the method above (bestsubset, greedysubset, greedysubsetF, monotonic_lars). For each matrix B, the best estimator is selected using the crossval function using the code below (figure 12), which gives this bestRegressorMatrix (figure 13):

```
bestRegressorMatrix = zeros(12, 4);
functions = {@bestsubset, @greedysubset, @greedysubsetF, @monotonic_lars};
for i = 1:4
    bestRegressorMatrix(:,i) = crossval(trainingData{:,:}, functions{i});
end
```

Figure 12: using crossval on 4 different sets of estimators

bestRegressorMatrix

12x4 double

	1	2	3	4	5
1	0.7546	0.7546	0.6251	0.6789	
2	0.1920	0.1920	0.2661	0.1797	
3	-0.1479	-0.1479	0	-0.0998	
4	0.1960	0.1960	0	0.1642	
5	0.3603	0.3603	0.2584	0.3166	
6	-0.3186	-0.3186	0	-0.1999	
7	0	0	0	0.0817	
8	0.2213	0.2213	0	0.0966	
9	0	0	0	0	
10	0	0	0	0	
11	0	0	0	0.0387	
12	0	0	0	-0.0156	
13					
14					

Figure 13: best estimator selected from each set using cross validation

The best estimator for bestsubset and greedysubset are the same, and monotonic_lars is the only one that has non-zero entries for the last 4 (randomly generated, independent normal) covariates. Using these 3 estimators, the mean squared prediction error can be computed using this formula and the code shown in figure 14 below:

$$\frac{1}{27}\sum_{i=1}^{27}(Y_i - X_i\beta)^2$$

```
xbeta = testData{:,2:end}*bestRegressorMatrix;
y = testData{:,1};
mspe = zeros(1, 4);
for i = 1:4
    mspe(i) = (1/27)*(norm(y-xbeta(:,i)))^2;
end
```

Figure 14: code computing MSPE for each estimator

This gives the following mean squared prediction error for the 4 methods above:

```
>> mspe

mspe =

    0.6821    0.6821    0.5761    0.6547
```

This suggests that the estimator given by greedysubsetF is the best estimator out of the 4, while variables 1 and 5 seem to also have the biggest impact on the response for the LARS estimator (by comparing magnitudes). Repeating this process with only 7 covariates (1, 2, 5 and the 4 random covariates) gives the following result:

```
trainingData = newData(1:70, [1,2,3,6,10:13]);
testData = newData(71:97, [1,2,3,6,10:13]);

newBestRegressorMatrix = zeros(7, 4);
for i = 1:4
    newBestRegressorMatrix(:,i) = crossval(trainingData{:,:}, functions{i});
end

xbeta = testData{:,2:end}*newBestRegressorMatrix;
y = testData{:,1};
newmspe = zeros(1, 4);
for i = 1:4
    newmspe(i) = (1/27)*(norm(y-xbeta(:,i)))^2;
end
```

newBestRegressorMatrix

7x4 double

	1	2	3	4	5
1	0.6251	0.6251	0.6251	0.5964	
2	0.2661	0.2661	0.2661	0.1970	
3	0.2584	0.2584	0.2584	0.1927	
4	0	0	0	0	
5	0	0	0	0	
6	0	0	0	0	
7	0	0	0	0	
8					
9					
10					
11					

```
>> newmspe

newmspe =

    0.5761    0.5761    0.5761    0.6102
```

The new mean squared prediction errors are smaller than the previous mean squared prediction errors, which suggests that the new model will likely perform better. From the values in newBestRegressorMatrix, lcavol has the greatest effect, while lweight and svi have less of an impact on the response, with the following linear relationship:

$$lpsa = \alpha * lcavol + \beta * lweight + \gamma * svi, \alpha > \beta \approx \gamma > 0$$

Reference:

https://www.maths.cam.ac.uk/undergrad/catam/II/10pt15.pdf