**10.15**

# Statistics

# Variable Selection and the Bias-Variance Tradeoff

# 1 The Bias-Variance tradeoff

**Question 1**

Assuming model (1), we can write $y = u\beta + \epsilon$. Then,

$$\begin{aligned}\mathbb{E}_{\mathcal{T},y|u}(y - u\hat{\beta})^2 &= \mathbb{E}_{\mathcal{T},y|u}(u\beta + \epsilon - u\hat{\beta})^2 \\ &= \mathbb{E}_{\mathcal{T}}(u\beta - u\hat{\beta})^2 + 2\mathbb{E}_{\mathcal{T},y|u}((u\beta - u\hat{\beta})\epsilon) + \mathbb{E}_{\mathcal{T},y|u}(\epsilon^2)\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{\mathcal{T},y|u}(u\beta\epsilon) &= u\beta\mathbb{E}_{\mathcal{T},y|u}(\epsilon) \\ &\quad \text{since } u\beta \text{ is non-random.} \\ \mathbb{E}_{\mathcal{T},y|u}(u\hat{\beta}\epsilon) &= \mathbb{E}_{\mathcal{T},y|u}(u\hat{\beta})\mathbb{E}_{\mathcal{T},y|u}(\epsilon) \\ &\quad \text{since } \; y,u\beta \perp \mathcal{T} \implies \epsilon \perp \mathcal{T} \implies \epsilon \perp \hat{\beta}.\end{aligned}$$

$$\begin{aligned}\text{Hence,} \qquad \mathbb{E}_{\mathcal{T},y|u}((u\beta - u\hat{\beta})\epsilon) &= \left(u\beta - \mathbb{E}_{\mathcal{T},y|u}(u\hat{\beta})\right)\mathbb{E}_{\mathcal{T},y|u}(\epsilon) \\ &= 0 \;\; \text{since} \;\; \mathbb{E}_{\mathcal{T},y|u}(\epsilon) = 0.\end{aligned}$$

$$\begin{aligned}\text{Let} \quad \mathbb{E}_{\mathcal{T},y|u}(\epsilon^2) = \mathbb{E}(\epsilon^2) = \sigma^2. \quad \text{Then we have,} \\ \mathbb{E}_{\mathcal{T},y|u}(y - u\hat{\beta})^2 &= \mathbb{E}_{\mathcal{T}}(u\beta - u\hat{\beta})^2 + \sigma^2 \\ &= \mathbb{E}_{\mathcal{T}}\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta}) + \mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)^2 + \sigma^2 \\ &= \mathbb{E}_{\mathcal{T}}\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)^2 + \mathbb{E}_{\mathcal{T}}\left(\mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)^2 \\ &\quad + 2\mathbb{E}_{\mathcal{T}}\left\{\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)\left(\mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)\right\} + \sigma^2\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{\mathcal{T}}\left(\mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)^2 &= \text{Var}_{\mathcal{T}}(u\hat{\beta}) \\ \mathbb{E}_{\mathcal{T}}\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)^2 &= \left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)^2 \\ \mathbb{E}_{\mathcal{T}}\left\{\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)\left(\mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)\right\} &= \mathbb{E}\left[\mathbb{E}_{\mathcal{T}}\left\{\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)\left(\mathbb{E}_{\mathcal{T}}(u\hat{\beta}) - u\hat{\beta}\right)\Big|u\right\}\right] \\ &= \mathbb{E}\left\{\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)u\mathbb{E}_{\mathcal{T}}(\hat{\beta} - \hat{\beta})\Big|u\right\} \\ &= 0\end{aligned}$$

$$\text{Hence,} \qquad \mathbb{E}_{\mathcal{T},y|u}(y - u\hat{\beta})^2 = \sigma^2 + \left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta})\right)^2 + \text{Var}_{\mathcal{T}}(u\hat{\beta}) \quad \square$$

Deleting the $p$th covariate has no effect on the irreducible variance $\sigma^2$.

For the squared estimation bias:

$$\begin{aligned}\hat{\beta}^{\text{LS}} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \;\; \text{with} \;\; \mathbf{y} = \mathbf{x}\beta + \boldsymbol{\epsilon} \text{ by assumption.} \\ \text{Thus,} \quad \mathbb{E}_{\mathcal{T}}(\hat{\beta}^{\text{LS}}) &= \mathbb{E}_{\mathcal{T}}((\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T(\mathbf{x}\beta + \boldsymbol{\epsilon})) \\ &= \beta \;\; \text{since } \boldsymbol{\epsilon} \perp \mathbf{x} \text{ and } \mathbb{E}(\boldsymbol{\epsilon}) = 0 \\ \text{and so,} \quad \left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta}^{LS})\right)^2 &= 0\end{aligned}$$

Let $\tilde{\mathbf{x}} = \mathbf{x}^{\{1,\ldots,p-1\}}$ be the $N \times (p-1)$ matrix formed by removing the last column from $\mathbf{x}$.

$$\begin{aligned}\hat{\beta}^{\{1,\ldots,p-1\}} &= (\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T\mathbf{y} \\ \implies \mathbb{E}_{\mathcal{T}}(\hat{\beta}^{\{1,\ldots,p-1\}}) &= \mathbb{E}_{\mathcal{T}}((\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T\mathbf{x})\beta \\ \mathbf{x} = \begin{bmatrix}\tilde{\mathbf{x}} & \mathbf{x}^p\end{bmatrix} \;\; \text{where} \;\; \mathbf{x}^p \;\; \text{is} \;\; \text{the } p^{th} \text{ column of } \mathbf{x}\end{aligned}$$

$$\implies \mathbb{E}_{\mathcal{T}}(\hat{\beta}^{\{1,\dots,p-1\}}) = \mathbb{E}_{\mathcal{T}}\left(\begin{bmatrix}\mathbf{I}_{p-1} & (\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T\mathbf{x}^p\end{bmatrix}\right)\beta$$

Let $\beta^{p-1}$ be the $(p-1)\times 1$ column vector formed by removing $\beta_p$ from $\beta$. Then,

$$\mathbb{E}_{\mathcal{T}}(\hat{\beta}^{\{1,\dots,p-1\}}) = \beta^{p-1} + \mathbb{E}_{\mathcal{T}}((\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T\mathbf{x}^p)\beta_p$$

$$\left(u\beta - \mathbb{E}_{\mathcal{T}}(u\hat{\beta}^{\{1,\dots,p-1\}})\right)^2 = u_p^2\beta_p^2\left(1 - \mathbb{E}_{\mathcal{T}}((\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T\mathbf{x}^p)\right)^2$$

Thus, when $\beta_p = 0$ the squared bias term is 0 for both models. However, when $\beta_p \neq 0$ the squared bias term can be positive for the reduced model while it is always 0 for the full model. So we get larger squared estimation bias for the reduced model in this case.

For the estimation variance:

$$\mathrm{Var}_{\mathcal{T}}(u\hat{\beta}) = \sum_{i=1}^{p} u_i^2 \mathrm{Var}_{\mathcal{T}}(\hat{\beta}_i)$$

$$\mathrm{Var}_{\mathcal{T}}(\hat{\beta}_i) \text{ from } (i,i) \text{ entry of } \mathrm{Cov}_{\mathcal{T}}(\hat{\beta})$$

$$\mathrm{Cov}_{\mathcal{T}}(\hat{\beta}^{\mathrm{LS}}) = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathrm{Cov}_{\mathcal{T}}(\mathbf{y})\mathbf{x}(\mathbf{x}^T\mathbf{x})^{-1}$$

$$\mathrm{Cov}_{\mathcal{T}}(\mathbf{y}) = \sigma^2\mathbf{I}_p \text{ since } (x_t, \epsilon_t) \text{ i.i.d. assumed}$$

$$\implies \mathrm{Cov}_{\mathcal{T}}(\hat{\beta}^{\mathrm{LS}}) = \sigma^2(\mathbf{x}^T\mathbf{x})^{-1}$$

$$\text{Likewise, } \mathrm{Cov}_{\mathcal{T}}(\hat{\beta}^{\{1,\dots,p-1\}}) = \sigma^2(\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}$$

Fixing $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{x} = \mathbf{I}_p$ we get,

$$\mathrm{Cov}_{\mathcal{T}}(\hat{\beta}^{\mathrm{LS}}) = \sigma^2\mathbf{I}_p$$

$$\implies \mathrm{Var}_{\mathcal{T}}(u\hat{\beta}^{\mathrm{LS}}) = \sigma^2\sum_{i=1}^{p} u_i^2$$

$$\mathrm{Cov}_{\mathcal{T}}(\hat{\beta}^{\{1,\dots,p-1\}}) = \sigma^2\mathbf{I}_{p-1}$$

$$\implies \mathrm{Var}_{\mathcal{T}}(u\hat{\beta}^{\{1,\dots,p-1\}}) = \sigma^2\sum_{i=1}^{p-1} u_i^2$$

Hence, when fixing $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{x} = \mathbf{I}_p$, we have that the estimation vaiance term of the smaller model is $\sigma^2u_p^2$ less than the corresponding term of the full model. Without fixing $\mathbf{x}^T\mathbf{x} = \mathbf{I}_p$, we still expect the estimation variance to decrease as there is only a contribution from $\mathrm{Var}_{\mathcal{T}}(\hat{\beta}_p)u_p^2$ in the full model while the other variance terms will be close between both models.

**Question 2**

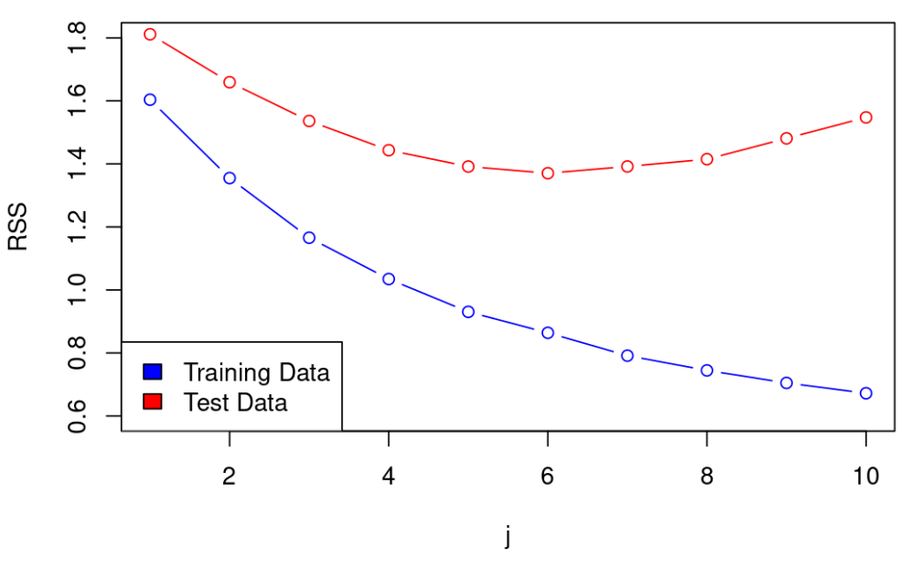


Figure 1: RSS against model size for $N_{\mathrm{tr}} = 30$

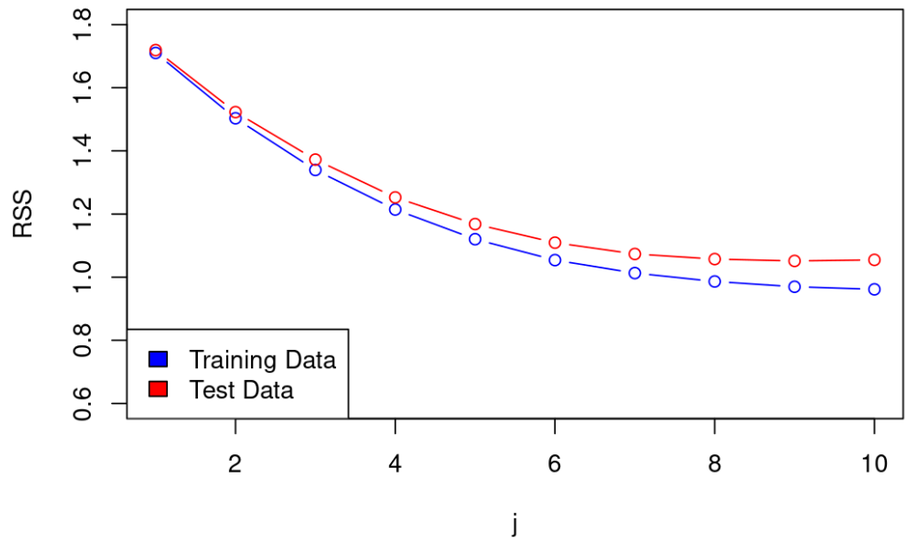


Figure 2: RSS against model size for $N_{\text{tr}} = 200$

When $N_{\text{tr}} = 200$, we observe changes to the plots of both the training and test error as well as their relative positions.

The test error decreases for all values of $j$ when $N_{\text{tr}}$ is increased to 200. This is because the models are being trained on more data so we expect to get a more accurate value of $\hat{\beta}$. In addition, when $N_{\text{tr}} = 30$ the plot of the test RSS increases for larger values fo $j$ while this does not happen for $N_{\text{tr}} = 200$. This can be explained using the bias-variance decomposition from question 1. The estimation variance growing as $j$ increases is what causes the increase in RSS for $N_{\text{tr}} = 30$. However, when $N_{\text{tr}} = 200$ the estimation variance is significantly reduced while the squared estimation bias is unchanged.

Conversely, the training error increases for all values of $j$ when $N_{\text{tr}}$ is increased to 200. This is because the model is more overfitted when using a smaller dataset. This difference can be described quantitatively. Let, $\mathbf{H} = \tilde{\mathbf{x}}(\tilde{\mathbf{x}}^T\tilde{\mathbf{x}})^{-1}\tilde{\mathbf{x}}^T$ where $\tilde{\mathbf{x}}$ is the $N \times j$ matrix formed by taking the first $j$ observations from $\mathbf{x}$. Let $\text{RSS}_{N_{\text{tr}}}$ be the training RSS calculated on a training data set of size $N_{\text{tr}}$. We have,

$$\begin{aligned}
\mathbb{E}\left(\text{RSS}_{N_{\text{tr}}} \mid \mathbf{x}\right) &= \mathbb{E}\left(||\mathbf{y} - \mathbf{x}\hat{\beta}^{\mathcal{M}_j}||^2 \mid \mathbf{x}\right) \\
&= \mathbb{E}\left(||(\mathbf{I} - \mathbf{H})\mathbf{y}||^2 \mid \mathbf{x}\right) \\
&= \mathbb{E}\left(||(\mathbf{I} - \mathbf{H})(\mathbf{x}\beta + \boldsymbol{\epsilon})||^2 \mid \mathbf{x}\right) \\
&= ||(\mathbf{I} - \mathbf{H})\mathbf{x}\beta||^2 + \mathbb{E}\left(||(\mathbf{I} - \mathbf{H})\boldsymbol{\epsilon}||^2 \mid \mathbf{x}\right) \\
&= ||(\mathbf{I} - \mathbf{H})\mathbf{x}\beta||^2 + (N - j)\sigma^2 \qquad (1)
\end{aligned}$$

The expected value of $\frac{1}{N_{\text{tr}}}||(\mathbf{I} - \mathbf{H})\mathbf{x}\beta||^2$ cancels for different values of $N_{\text{tr}}$ so applying the tower property of expectations,

$$\begin{aligned}
\frac{1}{200}\mathbb{E}(\text{RSS}_{200}) - \frac{1}{30}\mathbb{E}(\text{RSS}_{30}) &= j\left(\frac{1}{200} - \frac{1}{30}\right) \\
&= -\frac{17j}{600} \approx -0.283j
\end{aligned}$$

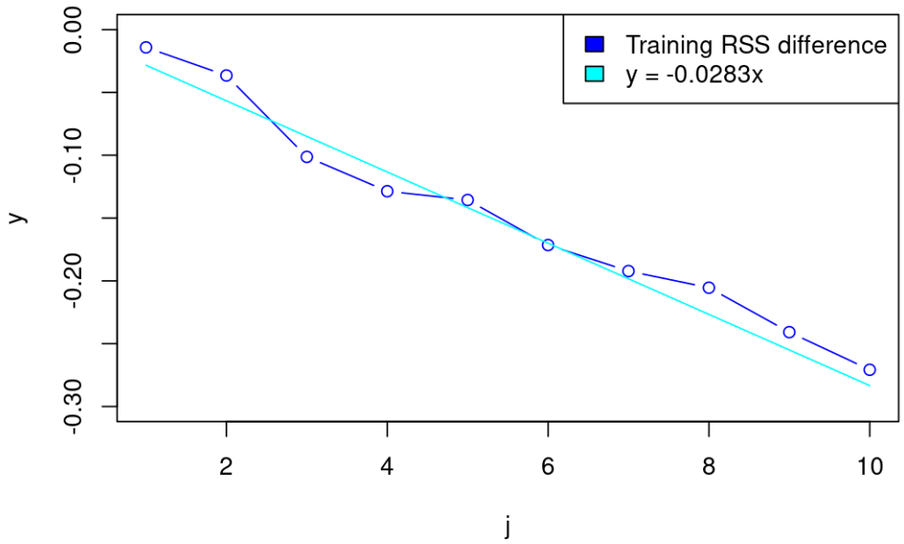


Figure 3: Difference between training RSS for $N_{\text{tr}} = 30$ and $N_{\text{tr}} = 200$ for different values of $j$

Qualitatively, we know the gap between the graphs for test error and training error will reduce for higher values of $N_{\text{tr}}$ since, from Figures 1 & 2, the test error decreases while the training error increases. We can also give a rough quantitative description of these gaps. We have,

$$\begin{aligned}
\sum_{i=1}^{n} \text{Var}(\mathbf{x}_i^T \hat{\beta}^{\mathcal{M}_j} \mid \mathbf{x}) &= \sum_{i=1}^{n} \text{Var}((\mathbf{H}\mathbf{y})_i) \\
&= \sum_{i=1}^{n} (\mathbf{H}\text{Cov}\mathbf{H}^T)_{ii} \\
&= \sum_{i=1}^{n} \sigma^2 \mathbf{H}_{ii}^2 \\
&= \sigma^2 \text{Tr}(\mathbf{H}) \quad \text{using} \quad \mathbf{H}^2 = \mathbf{H} \\
&= j\sigma^2 \quad \text{using} \quad \text{Tr}(\mathbf{H}) = \text{rank}(\mathbf{H}) = j
\end{aligned}$$

Then applying the bias-variance tradeoff,

$$\begin{aligned}
\implies \sum_{i=1}^{N_{\text{tr}}} \mathbb{E}_{\mathcal{T},y|u}(y - \mathbf{x}_i^T \hat{\beta}^{\mathcal{M}_j})^2 &= \sum_{i=1}^{N_{\text{tr}}} \left( \mathbf{x}_i^T \beta - \mathbb{E}_{\mathcal{T}}(\mathbf{x}_i^T \hat{\beta}^{\mathcal{M}_j}) \right)^2 + (N_{\text{tr}} + j)\sigma^2 \\
&= \mathbb{E}\left(\text{RSS}_{N_{\text{tr}}} \mid \mathbf{x}\right) + 2j\sigma^2 \quad \text{using equation (1)}
\end{aligned}$$

The LHS of the equation above is an estimate of the test error. Since it is calculated over the training $\mathbf{x}_i$, it is slightly innaccurate, particularly for smaller values of $N_{\text{tr}}$. Nonetheless, we obtain from the equation a crude estimate $2j\sigma^2/N_{\text{tr}}$ for the difference between the training and test error for fixed $N_{\text{tr}}$. This is known as the optimism and is demonstrated in Figures 4 & 5 below.

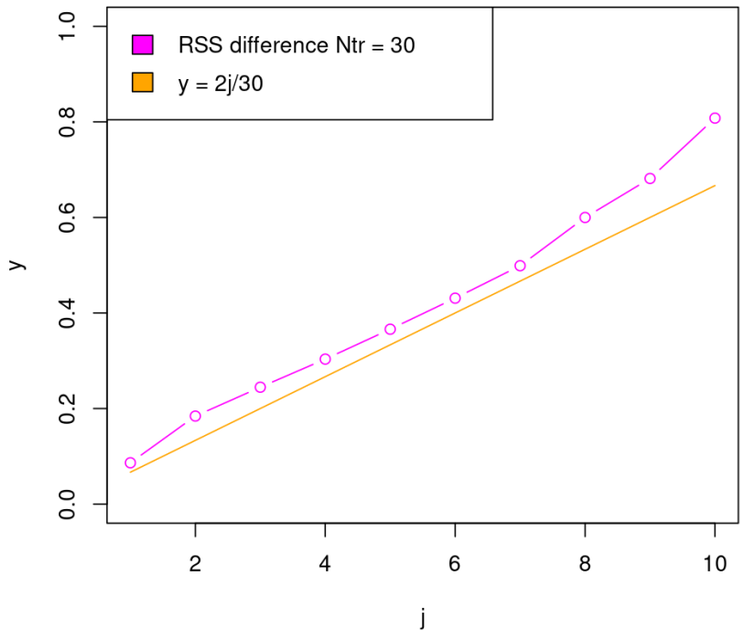


Figure 4: Gap between training and test RSS for $N_{\text{tr}} = 30$

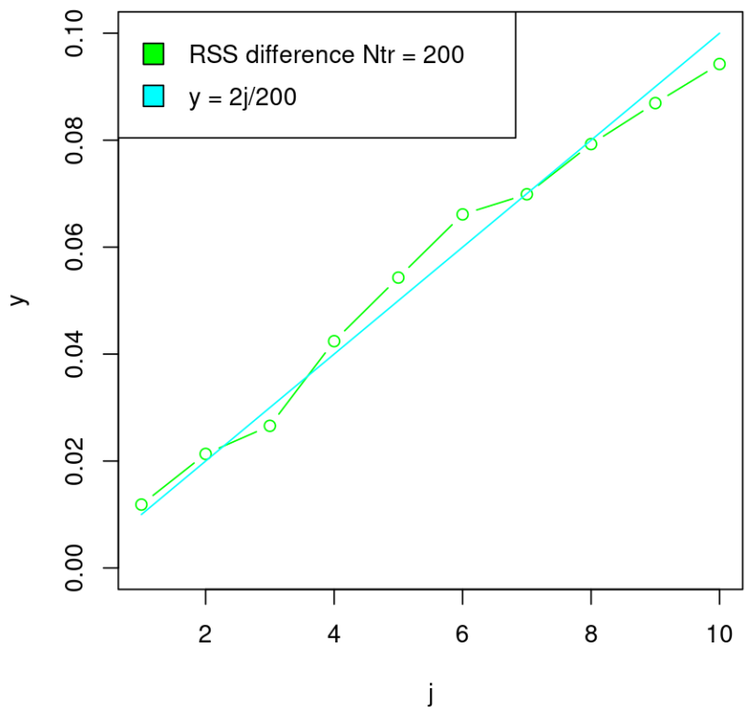


Figure 5: Gap between training and test RSS for $N_{\text{tr}} = 30$

# 2 Variable selection methods

### Question 3

Figure 6 is a $5 \times 5$ matrix $B$ produced using training data as in question 2 but with $p = 5$, $\beta = (-0.4, 0.35, -0.35, 0.3, -0.3)^T$ and $N_{\text{tr}} = 10$. There are the correct number of zeros in each column as specified by the decription of $B$. In addition, we see in row 2 that a zero appears following a non-zero entry, highighting the difference with the procedure *greedysubset* in question 4.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0.000000 | 0.0000000 | -0.4611837 | -0.48424111 | -0.49090767 |
| 2 | 0.000000 | 0.4952818 | 0.0000000 | 0.00000000 | -0.01852304 |
| 3 | 0.000000 | 0.0000000 | 0.0000000 | 0.07009682 | 0.07270943 |
| 4 | 0.000000 | 0.0000000 | 0.4907109 | 0.45882970 | 0.47304246 |
| 5 | -0.650901 | -0.6695713 | -0.8185213 | -0.80599961 | -0.81287961 |

Figure 6: Sample output of the *bestsubset* procedure

The size of the model space $\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, ..., p\}\}$ is $2^p - 1$ (assuming the empty set does not constitute a model).

*Proof.*

The number of models with $||\mathcal{M}|| = j$ is $\binom{p}{j}$.

$$\sum_{j=0}^{p} \binom{n}{j} x^j = (1+x)^p$$

$$\text{Put } x = 1$$

$$\implies \sum_{j=0}^{p} \binom{n}{j} = 2^p$$

$$\implies \sum_{j=1}^{p} \binom{n}{j} = 2^p - 1 \quad \square$$

The size of $\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, ..., p\}, ||\mathcal{M}|| = j\}$ is $\binom{p}{j}$. Since $\binom{p}{j}$ is increasing for $j \leq \lfloor \frac{p}{2} \rfloor$ and decreasing for $j \geq \lfloor \frac{p}{2} \rfloor$, we have,

$$\max_{j \in \{1,...,p\}} |\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, ..., p\}, ||\mathcal{M}|| = j\}| = \binom{p}{\lfloor \frac{p}{2} \rfloor}$$

$$\binom{20}{10} = 184,756 > 10^5$$

$$\binom{19}{9} = 92,378 < 10^5$$

Hence, the smallest value of $p$ for which the size of the search space $\{\mathcal{M} \mid \mathcal{M} \subseteq \{1, ..., p\}, ||\mathcal{M}|| = j\}$ exceeds $10^5$ for any $j \in \{1, ..., p\}$ is $p = 20$.

**Question 4**

The output in Figure 7 was produced using the exact same data set as for the figure in question 3. We observe that in this case non-zero rows remain non-zero, hence the third and fourth columns of Figures 6 & 7 differ. This is as expected.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0.000000 | 0.0000000 | -0.2874215 | -0.45136881 | -0.49090767 |
| 2 | 0.000000 | 0.4952818 | 0.4771862 | 0.03130621 | -0.01852304 |
| 3 | 0.000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.07270943 |
| 4 | 0.000000 | 0.0000000 | 0.0000000 | 0.46468127 | 0.47304246 |
| 5 | -0.650901 | -0.6695713 | -0.5765600 | -0.80610449 | -0.81287961 |

Figure 7: Sample output of the *greedysubset* procedure

At the moment, my program calculates $\hat{\beta}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}}$ individually for each $j \notin \mathcal{M}_d(\mathcal{T})$ and then the RSS via,

$$\hat{\beta}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}} = \left( (\mathbf{x}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}})^T \mathbf{x}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}} \right)^{-1} \mathbf{x}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}} \mathbf{y}$$

$$\text{RSS} = ||\mathbf{y} - \mathbf{x}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}} \hat{\beta}^{\mathcal{M}_d(\mathcal{T}) \cup \{j\}}||^2$$

- Computing $(\mathbf{x}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}})^T\mathbf{x}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$ takes $\mathcal{O}(Nd^2)$ time.
- The function *solve(t(X_M) %*% X_M, t(X_M) %*% Y)* on page 19 will then compute $\hat{\beta}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$ in $\mathcal{O}(d^3)$ time as this is the time taken to compute a Cholesky decomposition of $(\mathbf{x}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}})^T\mathbf{x}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$.
- The RSS can then be computed in $\mathcal{O}(Nd)$ time given the value of $\hat{\beta}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$.
- Overall cost of computing $\hat{\beta}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$ and the RSS is then $\mathcal{O}(Nd^2)$. The cost of computing the matrix $B$ will be $\mathcal{O}(Nd^4)$.

We can use the nested property of the models to reduce the compuational complexity of computing $\hat{\beta}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$. Let $\mathbf{x}^j$ be the $j^{\text{th}}$ column of $\mathbf{x}$. Then,

$$
\begin{aligned}
(\mathbf{x}^{\mathcal{M}_d\cup\{j\}})^T\mathbf{x}^{\mathcal{M}_d\cup\{j\}} &= \begin{bmatrix}(\mathbf{x}^{\mathcal{M}_d})^T\\ (\mathbf{x}^j)^T\end{bmatrix}\begin{bmatrix}\mathbf{x}^{\mathcal{M}_d} & \mathbf{x}^j\end{bmatrix}\\
&= \begin{bmatrix}(\mathbf{x}^{\mathcal{M}_d})^T\mathbf{x}^{\mathcal{M}_d} & (\mathbf{x}^{\mathcal{M}_d})^T\mathbf{x}^j\\ (\mathbf{x}^j)^T\mathbf{x}^{\mathcal{M}_d} & ||\mathbf{x}^j||^2\end{bmatrix}\\
\text{Use, } \begin{bmatrix}M & b\\ b^T & a\end{bmatrix}^{-1} &= \begin{bmatrix}M^{-1}+s^{-1}M^{-1}bb^TM^{-1} & -s^{-1}M^{-1}b\\ -s^{-1}b^TM^{-1} & s^{-1}\end{bmatrix}\\
&\text{where } s = a - b^TM^{-1}b\\
&\text{with } M = (\mathbf{x}^{\mathcal{M}_d})^T\mathbf{x}^{\mathcal{M}_d},\ b = (\mathbf{x}^{\mathcal{M}_d})^T\mathbf{x}^j
\end{aligned}
$$

It takes $\mathcal{O}(Nd)$ time to compute $b$. Given $M^{-1}$, it takes $\mathcal{O}(d^2)$ time to compute $M^{-1}b$. We can take $M^{-1}$ to be given using the fact that the models are nested. Hence, it takes $\mathcal{O}(Nd)$ time to compute $\hat{\beta}^{\mathcal{M}_d(\mathcal{T})\cup\{j\}}$ rather than the $\mathcal{O}(Nd^2)$ of the primitive approach, and the total cost for computing the matrix $B$ is now reduced to $\mathcal{O}(Nd^3)$.

**Question 5**

The output in Figure 8 was produced using data generated in the same way as question 3 except with $N = 60$ instead of 10. It demonstrates the procedure terminating with some components of $\hat{\beta}$ still 0. Notice that for larger $N$, the F-statistic is larger so we expect *greedysubset* to take longer to terminate. This agrees with the idea that for larger $N$ we can be more certain whether a component of $\beta$ is non-zero.

| | | | |
|---|---|---|---|
| 1 | -0.5048078 | -0.4856042 | -0.5155908 |
| 2 | 0.0000000 | 0.0000000 | 0.3157846 |
| 3 | 0.0000000 | 0.0000000 | 0.0000000 |
| 4 | 0.0000000 | 0.3168539 | 0.3362577 |
| 5 | 0.0000000 | 0.0000000 | 0.0000000 |

Figure 8: Sample output of the amended *greedysubset* procedure

This method of using an F-statistic would not work for best subset selection. This is because to do an F-test the predictors of the smaller model must be a subset of the predictors of the larger model. This is not the case in best subset selection where predictors used in $\mathcal{M}_j$ may not be included in $\mathcal{M}_{j+1}$ as the models are not guaranteed to be nested.

To prove that predictors of the smaller model must be a subset of the predictors of the larger model consider,

$$
\begin{aligned}
\frac{\text{RSS}(\hat{\beta}^{\mathcal{M}_d}) - \text{RSS}(\hat{\beta}^{\mathcal{M}_{d+1}})}{\text{RSS}(\hat{\beta}^{\mathcal{M}_{d+1}})/(N-d-1)} &= \frac{||(\mathbf{H}_{d+1}-\mathbf{H}_d)\mathbf{y}||^2}{||(\mathbf{I}-\mathbf{H}_{d+1})\mathbf{y}||^2/(N-d-1)} \qquad (2)\\
\text{where } \mathbf{H}_d &= \mathbf{x}^{\mathcal{M}_d}\left((\mathbf{x}^{\mathcal{M}_d})^T\mathbf{x}^{\mathcal{M}_d}\right)^{-1}(\mathbf{x}^{\mathcal{M}_d})^T
\end{aligned}
$$

Then the numerator and denominator of equation 2 must be independent for the test statistic to have an F-distribution. They are independent iff $\mathbf{H}_{d+1}\mathbf{H}_d = \mathbf{H}_d$, i.e. iff $\mathcal{M}_d \subset \mathcal{M}_{d+1}$. $\square$

**Question 6**

```
> T <- simulate_dataset(50)
> print(crossval(T, greedysubset))
[1] 0.8532748 0.5824782 0.0000000 0.2895601
> T <- simulate_dataset(50)
> print(crossval(T, greedysubset))
[1] 0.9979800 0.3561444 0.3287958 0.2147816
> T <- simulate_dataset(50)
> print(crossval(T, greedysubset))
[1] 0.9441178 0.2818161 0.4595399 0.0000000
> T <- simulate_dataset(50)
> print(crossval(T, greedysubset))
[1] 0.8723210 0.6176656 0.0000000 0.0000000
```

Figure 9: Sample output of the *crossval* procedure

Figure 9 demonstrates the functionality of the *crossval* procedure where the input $\mathcal{T}$ is generated as in question 2 but with $\beta = (1, 0.5, 0.2, 0.1)^T$ and $N = 50$ and we take *greedysubset* as our sparse estimator. The output shows that we can have a different number of non-zero elements of $\hat{\beta}^{CV}(\mathcal{T})$ depending on the value of $j^*$, and that these can be in different positions. This is what is intended by cross-validation where we try to find the model which trades of the bias and variance most favourably, thereby reducing overfitting. The procedure gives a biased estimate of the expected prediction error since the 10 RSS values which are averaged are not independent. Nonetheless, this estimate is still useful for comparing different models.

**Question 7**

*Claim.*

$$\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T}) = \arg\min_{\hat{\beta}} \left\{ \text{RSS}\left(\hat{\beta}; \mathcal{T}\right) \right\}$$
$$\text{subject to } \sum_{j=1}^{p} |\hat{\beta}_j| \leq t \qquad (*)$$

Where we will show below that $t = \sum_{j=1}^{p} |\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T})_j|$.

*Proof.*
Take $t = \sum_{j=1}^{p} |\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T})_j|$ and suppose $\hat{\beta}^*$ is a solution to the constrained problem (*). We can show that $\hat{\beta}^* = \hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T})$. We have,

$$\sum_{j=1}^{p} |\hat{\beta}_j^*| \leq t = \sum_{j=1}^{p} |\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T})_j| \quad \text{from which it follows,}$$
$$\text{RSS}\left(\hat{\beta}^*; \mathcal{T}\right) \leq \text{RSS}\left(\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T}); \mathcal{T}\right) \quad \text{by the definition of } \hat{\beta}^*$$
$$\implies \text{RSS}\left(\hat{\beta}^*; \mathcal{T}\right) + \lambda \sum_{j=1}^{p} |\hat{\beta}_j^*| \leq \text{RSS}\left(\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T}); \mathcal{T}\right) + \lambda \sum_{j=1}^{p} |\hat{\beta}^{(\mathcal{L},\lambda)}(\mathcal{T})_j|$$
$$\implies \hat{\beta}^* = \hat{\beta}^{(\mathcal{L},\lambda)} \quad \square$$

The last implication follows from the definition of $\hat{\beta}^{(\mathcal{L},\lambda)}$ as the solution to the L1 norm penalty problem and the uniqueness of $\hat{\beta}^{(\mathcal{L},\lambda)}$ which is guaranteed since the expression $\text{RSS}\left(\hat{\beta}; \mathcal{T}\right) + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|$ is convex in $\hat{\beta}$.

Hence, the Lasso has been expressed as a quadratic program with linear constraints in (*).

**Question 8**

Figure 10 shows the matrix outputted by the bestsubset procedure when applied to an instance of our training dataset. We can see that the added random variables are included in the later models, although not necessarily the last ones as shown by row 11 where the third added random variable is included in the model of size 6. We will say more about this in the analysis later. In isolation, this matrix is not useful for variable selection as we have no way of knowing which model size will be best in terms of the predictive power. Hence we must pair it with a method such as *crossval* to choose a model.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.8344862 | 0.7718500 | 0.6426791 | 0.6779724 | 0.7024965 | 0.71328850 | 0.7378901 | 0.75399050 | 0.76582012 | 0.76278757 | 0.75799656 | 0.758048401 |
| 2 | 0.0000000 | 0.2686037 | 0.2848196 | 0.3322263 | 0.2254536 | 0.21864308 | 0.2338166 | 0.22432894 | 0.22189530 | 0.21935445 | 0.21875771 | 0.218984885 |
| 3 | 0.0000000 | 0.0000000 | 0.0000000 | -0.1460093 | -0.1875159 | -0.18959970 | -0.2064020 | -0.20600132 | -0.19842913 | -0.19562524 | -0.19395566 | -0.193899917 |
| 4 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.2058044 | 0.21387381 | 0.1952188 | 0.20562354 | 0.20469590 | 0.19536741 | 0.19277168 | 0.192675612 |
| 5 | 0.0000000 | 0.0000000 | 0.2709871 | 0.2731239 | 0.3015438 | 0.31100710 | 0.3225560 | 0.33825677 | 0.33125661 | 0.32832686 | 0.32829697 | 0.328134804 |
| 6 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | -0.1503935 | -0.15728574 | -0.14525795 | -0.13892606 | -0.13867691 | -0.138223032 |
| 7 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.0000000 | 0.00000000 | -0.07173363 | -0.06846594 | -0.06554820 | -0.065514505 |
| 8 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.1107086 | 0.10128964 | 0.14654197 | 0.13605563 | 0.13432986 | 0.133922312 |
| 9 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.0000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | -0.001117015 |
| 10 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.0000000 | 0.00000000 | 0.00000000 | 0.00000000 | -0.01982290 | -0.019834586 |
| 11 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.05412257 | 0.0000000 | 0.05509555 | 0.06172721 | 0.06469893 | 0.06579040 | 0.065770853 |
| 12 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.0000000 | 0.00000000 | 0.0000000 | 0.00000000 | 0.00000000 | -0.03663977 | -0.03894425 | -0.039216959 |

Figure 10: Output of the bestsubset procedure applied to our training dataset

Once we have applied the *crossval* procedure to our *bestsubset* and *greedysubset* procedures, we would like a way to compare the various methods we have developed in order to choose the one which will be most useful for variable selection. Figure 11 gives the results of the crossval procedure with the various sparse estimators as well as the final column of the the greedysubset procedure with F-test and presents this data in the form of boxplots. I find boxplots to be suitable in this case because they allow for a visual comparison of a large amount of data. Crossval of the LARS algorithm works in largely the same way as the subset selection methods; we are just using a different method of forming the matrix of Figure 10. Note there is a small discrepency between the LARS algorithm used here and what the MATLAB algorithm calculates since I have used a package in R instead.

The Lasso can be viewed as a convex relaxation of best subset selection since this is equivalent to the Lasso estimator with an L0 penalty. With this interpretation, the Lasso should not outperform best subset selection since it is trying to approximate best subset selection. Hence, the utility of LARS is the gain in computational efficiency which we saw was an issue for best subset selection in question 3. Using this fact, we would not expect a statistically significant difference between the test accuracy of the Lasso estimate and the best subset estimate under crossvalidation. This agrees with the observed similarity in average and spread of all of the methods presented in the boxplots.

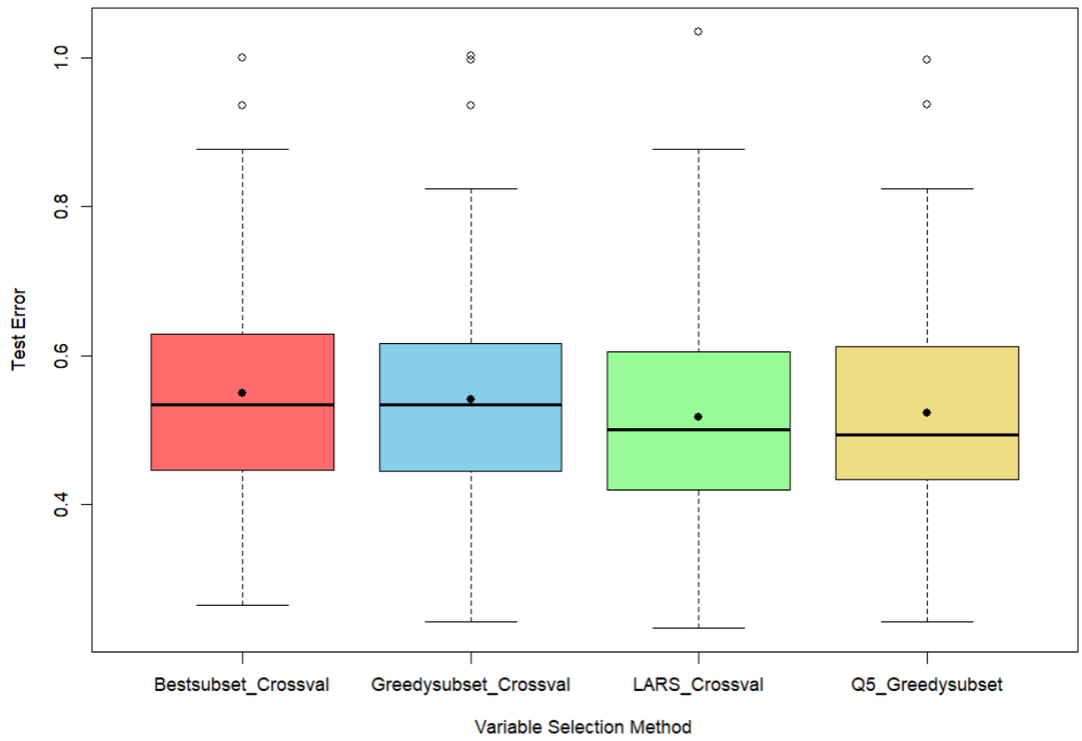


Figure 11: Boxplots of the test error from different variable selection methods taken over 100 tests. Black dots represent mean

If we are to use any of the methods we have presented, it is necessary to ask how happy we are with the variables selected; perhaps we have selected too many, perhaps too few. For example, in Table 1 where each method is run once, the output from *greedysubset* and LARS when crossvalidated contains one of the extra four random variables. Then we can safely say our output is overfitted. More generally we can imagine a scale with the following cases:

- Overfitted (correct model + some accidental variables)
- Correct model
- Overregularised (correct model - some true variables)

| Method | $\hat{\beta}^T$ |
|---|---|
| Bestsubset Crossval | (0.6624393, 0, 0, 0.2493837, 0.3580607, 0, 0, 0, 0, 0, 0, 0) |
| Greedysubset Crossval | (0.6863577, 0.1952665, -0.1420235, 0.2077256, 0.4020082, -0.1881193, 0.1358441, 0, 0, -0.2154226, 0, 0) |
| LARS Crossval | (0.5684567, 0.1264915, 0, 0.1352932, 0.3098531, 0, 0.05164052, 0, 0, -0.1410378, 0, 0) |
| Q5 Greedysubset | (0.6470449, 0 0 0.2668897, 0.3810977, 0, 0, 0, 0, -0.2015432, 0, 0) |

Table 1: Table showing the models selected by one exapmple test run

Crossvalidation typically gives overfitted models since the procedure gives a biased estimate of the expected prediction error. This also means that using the LARS algorithm selects a smaller value of $\lambda$ than the correct model. Despite the fact that our variable selection methods are mostly overfitted, we would rather this be the case then for them to be overregularised since this would be more detrimental to predictive accuracy.

The amended *greedysubset* procedure from question 5 is particularly helpful in dealing with the issue of overfitting. We know for certain that the variables selected are statistically significant which provides a sufficient reason to include them in the model.

Thus, we can see the benefits of this procedure for model interpretability when compared with the other methods; the reason for a variable being included in the model is clearer than for the other methods. Table 2 demonstrates that the reduction in overfitting is dramatic for the amended *greedysubset* procedure (certainly statistically significant although this is difficult to test since the t-test does not apply when the variance differs between groups). We can see this because this method includes much fewer of the four additional random variables on average. Since the test results seen in Figure 11 are on par with the other methods, it seems the natural choice of method to use.

| Method | Bestsubset Crossval | Greedysubset Crossval | LARS Crossval | Q5 Greedysubset |
|---|---|---|---|---|
| Mean | 1.38 | 1.29 | 1.63 | 0.25 |
| Variance | 0.76 | 0.72 | 1.30 | 0.21 |

Table 2: Table showing the mean and variance of the number of non-zero entries included in the model from the four random variables calculated over 100 tests

Taking this into account, I run the amended *greedysubset* algorithm on 100 different datasets (each with new random varaibles added and a random 70:27 split). Then we obtain the following tally for the number of times each variable is selected,

| Variable | Tally | Average Value |
|---|---|---|
| lcavol | 100 | 0.624 |
| lweight | 83 | 0.255 |
| age | 3 | -0.004 |
| lbph | 17 | 0.031 |
| svi | 87 | 0.280 |
| lcp | 1 | 0.000 |
| gleason | 0 | 0.000 |
| pgg45 | 4 | 0.000 |
| rv1 | 4 | 0.004 |
| rv2 | 3 | -0.010 |
| rv3 | 7 | 0.003 |
| rv4 | 5 | 0.002 |

Table 3: Table of tally and average values as described in paragraph above

We find a mean of 3.14 variables selected for each run. Thus, selecting the 3 most commonly occuring variables, our final model will include *lcavol, lweight* and *svi*. In particular, we can simply read off the average value for the coefficient of each of these variables for our final model. Hence, overall, balancing predictive accuracy with interpretability and model regularity, I would select the following model:

$$\hat{\beta} = (0.624, 0.255, 0, 0, 0.280, 0, 0, 0)^T$$

**Q2_simulate_dataset.R for Question 2**

```
# PAT: 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

library(MASS)

simulate_dataset <- function(N) {
  sigma_sq <- 1
  #beta <- c(-0.5, 0.45, -0.4, 0.35, -0.3,
  #          0.25, -0.2, 0.15, -0.1, 0.05) # Use in Q2
  # beta <- c(-0.4, 0.35, -0.35, 0.3, -0.3) # Use in Q3, Q4 & Q5
  beta <- c(1, 0.5, 0.2, 0.1) # for Q6
  p <- length(beta)
  epsilon_vec <- rnorm(N, 0, sigma_sq)
  X_matrix <- mvrnorm(N, rep(0, p), diag(p))
  Y_vec <- X_matrix %*% beta + epsilon_vec

  return(list(Y_vec, X_matrix))
}
```

**Q2_compute_error.R for Question 2**

```
source('Q2_simulate_dataset.R')

compute_error <- function(N_tr = 30) {
  N_te = 1000
  p <- 10

  training_RSS <- vector() # empty vector
  test_RSS <- vector()

  training_data <- simulate_dataset(N_tr)
  Y_training <- training_data[[1]]
  X_training <- training_data[[2]]
  test_data <- simulate_dataset(N_te)
  Y_test <- test_data[[1]]
  X_test <- test_data[[2]]

  for (j in 1:p) {
    X_M <- X_training[,1:j]
    # columns 1 to j of X
    beta_M <- (solve(t(X_M) %*% X_M)
               %*% t(X_M) %*% Y_training)
    # solve finds matrix inverse
    beta_M <- c(beta_M, integer(p - j))
    # integer(p - j) is p-j length vector of 0's
    training_RSS <- c(training_RSS,
                      t(Y_training - X_training %*% beta_M) %*%
                      (Y_training - X_training %*% beta_M) / N_tr)
    test_RSS <- c(test_RSS,
                  t(Y_test - X_test %*% beta_M) %*%
                  (Y_test - X_test %*% beta_M) / N_te)
    # each entry of these vectors is the RSS for j = 1,...,p
  }

  return(list(training_RSS, test_RSS))
}
```

**Q2_plot for Question 2**

```
source('Q2_compute_error.R')

generate_plot <- function() {
  p <- 10
  training_RSS <- rep(0, p)
  test_RSS <- rep(0, p)

  for (exp_no in 1:100) {
    error_data <- compute_error(N_tr = 30)
    new_tr_RSS <- error_data[[1]]
    new_te_RSS <- error_data[[2]]
    training_RSS <- training_RSS + new_tr_RSS
    test_RSS <- test_RSS + new_te_RSS
  }

  training_RSS <- training_RSS / 100
  test_RSS <- test_RSS / 100

  j <- 1:10

  plot(j, training_RSS,
       main = 'RSS against Model Size',
       xlab = 'j', ylab = 'RSS',
       type = 'b',
       col = 'blue',
       ylim = c(0.6, 1.8))
  lines(j, test_RSS,
        type = 'b',
        col = 'red')
  legend('bottomleft',
         legend = c('Training Data', 'Test Data'),
         fill = c('blue','red'),
         bty = 'o') # bty = 'n' means no box around legend
}
```

**Q2_plot_comparison.R for Question 2**

```
source('Q2_compute_error.R')

plot_comparison <- function() {
  p <- 10
  training_RSS_diff <- rep(0, p)
  test_RSS_diff <- rep(0, p)
  # have vectors for differences in RSS for
  # different values of Ntr

  for (exp_no in 1:100) {
    error_data_30 <- compute_error(N_tr = 30)
    error_data_200 <- compute_error(N_tr = 200)
    new_tr_RSS_30 <- error_data_30[[1]]
    new_te_RSS_30 <- error_data_30[[2]]
    new_tr_RSS_200 <- error_data_200[[1]]
    new_te_RSS_200 <- error_data_200[[2]]
    training_RSS_diff <- training_RSS_diff +
      new_tr_RSS_30 - new_tr_RSS_200
    test_RSS_diff <- test_RSS_diff +
      new_te_RSS_30 - new_te_RSS_200
  }

  training_RSS_diff <- training_RSS_diff / 100
  test_RSS_diff <- test_RSS_diff / 100

  j <- 1:10

  plot(j, training_RSS_diff,
       main = 'RSS difference against Model Size',
       xlab = 'j', ylab = 'y',
       type = 'b',
       col = 'blue',
       ylim = c(-0.3, 0))
  #ticks = c(-0.3, -0.2, -0.1, 0)
  #axis(side = 2, at = ticks)
  #lines(j, test_RSS_diff,
         # Don't actually want test RSS for this graph
  #       type = 'b',
  #       col = 'red')
  lines(j, -17 * j / 600,
        type = 'l',
        col = 'cyan')
  legend('topright',
         legend = c('Training RSS difference',
                    'y = -0.0283x'),
         fill = c('blue', 'cyan'),
         bty = 'o') # bty = 'n' means no box around legend
}
```

**Q2_plot_gaps.R for Question 2**

```
source('Q2_compute_error.R')

plot_gaps <- function() {
  Ntr = 200
  p <- 10
  gap <- rep(0, p)
  # have vector for gaps between test and training
  # RSS

  for (exp_no in 1:100) {
    error_data_Ntr <- compute_error(N_tr = Ntr)
    new_tr_RSS <- error_data_Ntr[[1]]
    new_te_RSS <- error_data_Ntr[[2]]

    gap <- gap + new_te_RSS - new_tr_RSS
  }

  gap <- gap / 100

  j <- 1:10

  plot(j, gap,
       main = 'Error gap Model Size',
       xlab = 'j', ylab = 'y',
       type = 'b',
       col = 'green',
       ylim = c(0, 0.1)) # EDIT for desired Ntr
  #ticks = c(-0.3, -0.2, -0.1, 0)
  #axis(side = 2, at = ticks)
  lines(j, 2*j/Ntr,
        type = 'l',
        col = 'cyan')
  legend('topleft',
         legend = c('RSS difference Ntr = 200',
                    'y = 2j/200'),  # EDIT for desired Ntr
         fill = c('green', 'cyan'),
         bty = 'o') # bty = 'n' means no box around legend
}
```

**Q3_bestsubset.R for Question 3**

```
get_search_space <- function(p) {
  # This function will generate all unique subsets of
  # {1,...,p} (power set)
  output <- list(c())
  for (i in 1:p) {
    output_copy <- output
    for (seq in output_copy) {
      output[[length(output) + 1]] <- c(seq, i)
    }
  }
  return(output)
}

bestsubset <- function(T) {
  # T is the data set. Assume it is in the format as returned
  # by Q2_simulate_dataset.R
  Y <- T[[1]]
  X <- T[[2]]
  p <- ncol(X)  # This is the length of x_i

  B <- matrix(nrow = p, ncol = 0)
  search_space <- get_search_space(p)

  for (j in 1:p) {
    # Setup :
    search_space_j <- list()
    for (seq in search_space) {
      if (length(seq) == j) {
        search_space_j[[length(search_space_j) + 1]] <- seq
      }
    }  # Taken all the length j sequences from the search space

    min_RSS <- Inf
    best_beta_M <- c()  # Empty vector for now
    best_M <- c()

    # Finding optimal beta:
    for (M in search_space_j) {
      X_M <- X[, M]
      beta_M <- solve(t(X_M) %*% X_M, t(X_M) %*% Y)
      RSS <- t(Y - X_M %*% beta_M) %*%
               (Y - X_M %*% beta_M)
      if (RSS < min_RSS) {
        best_beta_M <- beta_M
        best_M <- M
        min_RSS <- RSS
      }
    }

    # Make it a col vector of length p:
    new_col <- rep(0, p)
    beta_index = 1
    for (index in best_M) {
      new_col <- replace(new_col, index, best_beta_M[beta_index])
      beta_index <- beta_index + 1
    }
```

```
    # Append to matrix B
    B <- cbind(B, new_col)
  }
  return(B)
}

# Code to run (in terminal) for example output:
# source('Q2_simulate_dataset.R')
# source('Q3_bestsubset.R')
# T <- simulate_dataset(10)
# B <- bestsubset(T)
# View(B)
```

**Q4_greedysubset.R for Question 4**

```
greedysubset <- function(T) {
  Y <- T[[1]]
  X <- T[[2]]
  p <- ncol(X)

  B <- matrix(nrow = p, ncol = 0)
  prev_model <- c()

  for (j in 1:p) {
    search_space <- list()
    for (l in 1:p) {
      if (!is.element(l, prev_model)) {
        search_space[[length(search_space) + 1]] <-
          c(prev_model, l)
      }
    }

    # Same code as Q3_bestsubset.R :
    min_RSS <- Inf
    best_beta_M <- c()  # Empty vector for now
    best_M <- c()

    for (M in search_space) {
      X_M <- X[, M]
      beta_M <- solve(t(X_M) %*% X_M, t(X_M) %*% Y)
      RSS <- t(Y - X_M %*% beta_M) %*%
        (Y - X_M %*% beta_M)
      if (RSS < min_RSS) {
        best_beta_M <- beta_M
        best_M <- M
        min_RSS <- RSS
      }
    }

    new_col <- rep(0, p)
    beta_index = 1
    for (index in best_M) {
      new_col <- replace(new_col, index, best_beta_M[beta_index])
      beta_index <- beta_index + 1
    }
    B <- cbind(B, new_col)

    prev_model <- best_M  # Update best M for loop
  }
  return(B)
}

# Code to run (in terminal) for example output:
# source('Q2_simulate_dataset.R')
# source('Q4_greedysubset.R')
# T <- simulate_dataset(10)
# B <- greedysubset(T)
# View(B)
```

**Q5_greedysubset.R for Question 5**

```
# This will be the amended version of Q4_greedysubset.R
# as specified by the question

Q5_greedysubset <- function(T) {
  Y <- T[[1]]
  X <- T[[2]]
  p <- ncol(X)
  N <- nrow(X)

  B <- matrix(nrow = p, ncol = 0)
  prev_model <- c()

  for (j in 1:p) {
    search_space <- list()
    for (l in 1:p) {
      if (!is.element(l, prev_model)) {
        search_space[[length(search_space) + 1]] <-
          c(prev_model, l)
      }
    }

    min_RSS <- Inf
    best_beta_M <- c()  # Empty vector for now
    best_M <- c()

    for (M in search_space) {
      X_M <- X[, M]
      beta_M <- solve(t(X_M) %*% X_M, t(X_M) %*% Y)
      RSS <- t(Y - X_M %*% beta_M) %*%
        (Y - X_M %*% beta_M)
      if (RSS < min_RSS) {
        best_beta_M <- beta_M
        best_M <- M
        min_RSS <- RSS
      }
    }
    # F-test added here:
    if (j >= 2) { # Cannot do F-test when j = 1
      F_quantile <- qf(0.95, 1, N - j - 1)
      test_stat <- (prev_min_RSS - min_RSS) /
        (min_RSS / (N - j - 1))
      if (test_stat < F_quantile) {
        # < ! because if this is the case then d+1 is not
        # significant improvement
        break # escapes loop for (j in 1:p)
      }
    }

    new_col <- rep(0, p)
    beta_index = 1
    for (index in best_M) {
      new_col <- replace(new_col, index, best_beta_M[beta_index])
      beta_index <- beta_index + 1
    }
    B <- cbind(B, new_col)
```

```
    prev_model <- best_M  # Update best M for loop
    prev_min_RSS <- min_RSS # Store for F-test
  }
  return(B)
}

# Code to run (in terminal) for example output:
# source('Q2_simulate_dataset.R')
# source('Q5_greedysubset.R')
# T <- simulate_dataset(60)
# (For larger N can be more certain about components of beta)
# B <- Q5_greedysubset(T)
# View(B)
```

**Q6_crossval.R for Question 6**

```
crossval <- function(T, sparse_estimator) {
# sparse_estimator is a function. e.g. bestsubset or greedysubset
  Y <- T[[1]]
  X <- T[[2]]
  p <- ncol(X)
  N <- nrow(X)

  # Generate the folds in the data:
  folds <- list()
  fold_complements <- list()
  random_permutation <- sample(c(1:N), N)
  n <- 1
  for (k in 1:10) {
    Y_fold <- c()  # This is the y vector added to the new fold
    X_fold <- matrix(nrow = 0, ncol = p)
    Y_complement <- Y # This will be the complement of Y_fold
    X_complement <- X
    entries_to_remove <- c() # Will use to make Y_complement
    while (n <= k * N / 10) {
      pi_n <- random_permutation[n]
      entries_to_remove <- append(entries_to_remove, pi_n)
      y_perm <- Y[pi_n]
      x_perm <- X[pi_n, ] # Accesses row of X
      Y_fold <- append(Y_fold, y_perm)
      X_fold <- rbind(X_fold, x_perm)
      n <- n + 1
    }
    Y_complement <- Y_complement[-entries_to_remove]
    X_complement <- X_complement[-entries_to_remove,]
    new_fold <- list(Y_fold, X_fold)
    new_complement <- list(Y_complement, X_complement)
    folds[[k]] <- new_fold
    fold_complements[[k]] <- new_complement
  }

  # Use the sparse estimator to find the matrix B
  # (as in Question 4) using the complement of each
  # fold as training data:
  B_matrix_list <- list()
  # ^ list of B matrix generated by the sparse estimator
  # using the 10 different fold complements
  for (k in 1:10) {
    complement_fold <- fold_complements[[k]]
    B <- sparse_estimator(complement_fold)
    B_matrix_list[[k]] <- B
  }

  # Find j* based on estimated prediction error:
  j_star <- 1
  best_prediction_error <- Inf
  for (j in 1:p) {
    prediction_error <- 0
    for (k in 1:10) {
      Y_k <- folds[[k]][[1]]
      X_k <- folds[[k]][[2]]
      B <- B_matrix_list[[k]]
```

```
      beta_jk <- B[,j]  # jth column of b matrix
      RSS <- t(Y_k - X_k %*% beta_jk) %*%
                (Y_k - X_k %*% beta_jk) / (length(Y_k))
      prediction_error <- prediction_error + RSS
    }
    prediction_error < prediction_error / 10
    if (prediction_error < best_prediction_error) {
      best_prediction_error <- prediction_error
      j_star <- j
    }
  }

  B <- sparse_estimator(T)
  # B matrix on entire data set
  beta_cv <- B[, j_star]
  # Assume question means to find beta_cv using sparse estimator
  # still considering that's what was used to find j*
  # Otherwise use best_subset just for subsets of size j.
  # (Confirmed by CATAM help email)

  return(beta_cv)
}

# Code for sample output:
# source('Q6_crossval.R')
# source('Q2_simulate_dataset.R')
# source('Q4_greedysubset.R')
# T <- simulate_dataset(100)
# print(crossval(T, greedysubset))
```

**Q8_lars.R for Question 8**

```
# An amended version of the 'lars' function from the 'lars'
# package in R which will work as input into Q6_crossval.R

library('lars')

Q8_lars <- function(T) {
  Y <- T[[1]]
  X <- T[[2]]
  object <- lars(X, Y, type = 'lasso')
  coef_matrix <- predict.lars(object, type = 'coef')$coefficients
  B <- coef_matrix[-1,]
  B <- t(B)
  return(B)
}
```

**Q8_getresults.R for Question 8**

```
source('Q3_bestsubset.R')
source('Q4_greedysubset.R')
source('Q5_greedysubset.R')
source('Q6_crossval.R')
source('Q8_lars.R')
library('lars')

# Generating the dataset:

prostate_data <- read.table(
        "~/CATAM-10.15-RStudio/II-10-15-2022-prostate.dat",
        quote="\"", comment.char="")

Q8_getresults <- function() {
  for (i in 1:4) {
    new_col <- rnorm(97)
    prostate_data <- cbind(prostate_data, new_col)
  }
  random_permutation <- sample(97)
  test_data_indices <- random_permutation[c(1:27)]
  test_data <- list(prostate_data[test_data_indices,1],
              data.matrix(prostate_data[test_data_indices,-1]))
  training_data <- list(prostate_data[-test_data_indices,1],
              data.matrix(prostate_data[-test_data_indices,-1]))


  # Subset selection outputs:

  bestsubset_output <- bestsubset(training_data)
  greedysubset_output <- greedysubset(training_data)
  Q5_greedysubset_output <- Q5_greedysubset(training_data)


  # Shrinkage-based methods outputs:

  bestsubset_crossval <- crossval(training_data, bestsubset)
  # Notice that the above is equivalent to L0 norm Lasso
  greedysubset_crossval <- crossval(training_data, greedysubset)
  lars_crossval <- crossval(training_data, Q8_lars)
  # lars_crossval can be thought of as the L1 Lasso estimator
  # which performs best in terms of prediction error
  # (i.e. we have selected the value of lambda which gives
  # this optimal estimator)


  # Use test data to compute error of all the methods above:

  Y_test <- test_data[[1]]
  X_test <- test_data[[2]]
  bestsubset_test_error <- c()
  for (i in 1:ncol(bestsubset_output)) {
    bestsubset_test_error[i] <-
      t(Y_test - X_test %*% bestsubset_output[,i]) %*%
      (Y_test - X_test %*% bestsubset_output[,i]) / 30
  }
  greedysubset_test_error <- c()
```

```
  for (i in 1:ncol(greedysubset_output)) {
    greedysubset_test_error[i] <-
      t(Y_test - X_test %*% greedysubset_output[,i]) %*%
      (Y_test - X_test %*% greedysubset_output[,i]) / 30
  }
  Q5_greedysubset_test_error <- c()
  for (i in 1:ncol(Q5_greedysubset_output)) {
    Q5_greedysubset_test_error[i] <-
      t(Y_test - X_test %*% Q5_greedysubset_output[,i]) %*%
      (Y_test - X_test %*% Q5_greedysubset_output[,i]) / 30
  }

  bestsubset_crossval_test_error <-
    t(Y_test - X_test %*% bestsubset_crossval) %*%
    (Y_test - X_test %*% bestsubset_crossval) / 30
  greedysubset_crossval_test_error <-
    t(Y_test - X_test %*% greedysubset_crossval) %*%
    (Y_test - X_test %*% greedysubset_crossval) / 30
  lars_crossval_test_error <-
    t(Y_test - X_test %*% lars_crossval) %*%
    (Y_test - X_test %*% lars_crossval) / 30

  return(list(bestsubset_test_error,
              greedysubset_test_error,
              Q5_greedysubset_test_error,
              as.numeric(bestsubset_crossval_test_error),
              as.numeric(greedysubset_crossval_test_error),
              as.numeric(lars_crossval_test_error),
              Q5_greedysubset_output[,i],
              bestsubset_crossval,
              greedysubset_crossval,
              lars_crossval))
  # Last four entries of list are for Q8_displaycount
}
```

**Q8_displayresults.R for Question 8**

```
source('Q8_getresults.R')

Q8_displayresults <- function() {
  bestsubset_cv_error <- c()
  greedysubset_cv_error <- c()
  lars_cv_error <- c()
  Q5_greedy_end_error <- c()
  # ^ will be error from final column of Q5_greedysubset
  for (i in 1:100) {
    print(i)
    errors <- Q8_getresults()
    # All of the test data RSS in a list
    bestsubset_error <- errors[[1]]  # Unused (not useful)
    greedysubset_error <- errors[[2]]  # Unused (not useful)
    Q5_greedysubset_error <- errors[[3]]
    bestsubset_cv_error[i] <- errors[[4]]
    greedysubset_cv_error[i] <- errors[[5]]
    lars_cv_error[i] <- errors[[6]]
    Q5_greedy_end_error[i] <- tail(errors[[3]], n = 1)
  }
  data <- data.frame(Bestsubset_Crossval = bestsubset_cv_error,
                     Greedysubset_Crossval = greedysubset_cv_error,
                     LARS_Crossval = lars_cv_error,
                     Q5_Greedysubset = Q5_greedy_end_error)
  means <- colMeans(data)
  boxplot(data,
          xlab = 'Variable Selection Method',
          ylab = 'Test Error',
          col=c("indianred1","skyblue",
                "palegreen","lightgoldenrod"))
  points(x = 1:4,  # Add mean points to plot
         y = means,
         col = "black",
         pch = 16)
}
```

**Q8_displaycount.R for Question 8**

```
# This file will count the number of times the four
# random variables are included in the model
source('Q8_getresults.R')

Q8_displaycount <- function() {
  Q5_greedy_count <- c()
  bestsubset_count <- c()
  greedy_count <- c()
  lars_count <- c()
  # vector for number of zeros in last 4 entries for each trial

  Q5_greedy_tally <- numeric(12)
  # ^ counts number of times each variable is included in a
  # run of Q5_greedy
  Q5_greedy_values <- numeric(12)
  # adds all the Q5_greedy vectors together

  for (i in 1:10) {
    print(i)
    errors <- Q8_getresults()
    Q5_greedy <- errors[[7]]
    bestsubset_crossval <- errors[[8]]
    greedysubset_crossval <- errors[[9]]
    lars_crossval <- errors[[10]]

    Q5_greedy_tally <- Q5_greedy_tally + (Q5_greedy != 0)
    Q5_greedy_values <- Q5_greedy_values + Q5_greedy


    Q5_greedy_count[i] <- length(which(Q5_greedy[9:12] != 0))
    bestsubset_count[i] <- length(which(
                              bestsubset_crossval[9:12] != 0))
    greedy_count[i] <- length(which(
                              greedysubset_crossval[9:12] != 0))
    lars_count[i] <- length(which(lars_crossval[9:12] != 0))
    # ^ counts number of non-zero elements in last four
    # components

    # cat(Q5_greedy, '\n')
    # cat(bestsubset_crossval, '\n')
    # cat(greedysubset_crossval, '\n')
    # cat(lars_crossval, '\n')
    # ^ used in the figure in Q8 which shows outputted models
    # uncomment it to use
  }
  print( c( var(Q5_greedy_count), var(bestsubset_count),
            var(greedy_count), var(lars_count) ) )
  print( c( mean(Q5_greedy_count), mean(bestsubset_count),
            mean(greedy_count), mean(lars_count) ) )

  print(Q5_greedy_tally)
  print(Q5_greedy_values / 100)
  print(sum(Q5_greedy_tally) / 100)
}
```