2131 HW9

Orly Olbum

Problem 1

Consider the data "Steam.txt" where you are trying to predict steam usage using fat, glycerine, wind, freezday and temp. Perform forward and backward selection with alpha1 = 0.1 and alpha2 = 0.2, respectively.

Which covariates did you include in the full model?

How does this compare when you perform best subset regression using AIC to rank models?

Does the model chosen with best subset regression change when you use BIC to rank models? Example R code can be found on Canvas to help you get started.

First backward selection. We will start by including all covariates, and remove the term with the largest p-value greater than alpha2 = 0.2.

```
start_model = lm(steam ~ fat + glycerine + wind + frezday + temp, data = steam)
summary(start_model)$coefficients
```

```
##                 Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  9.879818435 2.16523631  4.56292849 0.0002125811
## fat          0.684237984 0.54285705  1.26043862 0.2227715244
## glycerine    0.407284127 3.45093708  0.11802131 0.9072895429
## wind         0.002433783 0.10271441  0.02369465 0.9813432540
## frezday     -0.007772064 0.02859539 -0.27179433 0.7887115427
## temp        -0.083785859 0.01811323 -4.62567271 0.0001844331
```

We see that **wind** has the largest p-value over 0.2, so we remove it and re-run the model.

```
next_model = lm(steam ~ fat + glycerine + frezday + temp, data = steam)
summary(next_model)$coefficients
```

```
##                 Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)  9.907403967 1.77941444  5.5677889 1.891012e-05
## fat          0.682736321 0.52550129  1.2992096 2.086504e-01
## glycerine    0.413119118 3.35503209  0.1231342 9.032299e-01
## frezday     -0.007767227 0.02787104 -0.2786845 7.833481e-01
## temp        -0.083938797 0.01649589 -5.0884667 5.610177e-05
```

Next, **glycerine** has the largest p-value over 0.2, so we remove it and re-run the model.

```
next_model_2 = lm(steam ~ fat + frezday + temp, data = steam)
summary(next_model_2)$coefficients
```

```
##                 Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)  9.841732184 1.65732261  5.9383322 6.797703e-06
## fat          0.743542454 0.17543899  4.2381824 3.676151e-04
## frezday     -0.007463963 0.02710321 -0.2753904 7.857079e-01
## temp        -0.083574482 0.01584329 -5.2750717 3.139613e-05
```

Next, **frezday** has a large p-value over 0.2.

```
next_model_3 = lm(steam ~ fat + temp, data = steam)
summary(next_model_3)$coefficients
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept)  9.47422200 0.961894080   9.849548 1.589445e-09
## fat          0.76164829 0.159200936   4.784195 8.895837e-05
## temp        -0.07976077 0.007532872 -10.588361 4.223486e-10
```

We now have a model that has all p-values under the alpha2 threshold. We keep covariates **fat** and **temp**.

Now, let's try forward selection. We start with a model that only has the intercept.

```
for_model = lm(steam ~ 1, data = steam)
summary(for_model)$coefficients
```

```
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept)    9.424  0.3261283 28.8966 3.694198e-20
```

Now we run a model for each of the covariates, and keep the one with the smallest p-value under alpha1 = 0.1.

```
for_model_1 = lm(steam ~ fat, data = steam)
summary(for_model_1)$coefficients
```

```
##              Estimate Std. Error  t value   Pr(>|t|)
## (Intercept) 5.2615153  2.1147283 2.488034 0.02052837
## fat         0.7648252  0.3844298 1.989506 0.05866522
```

```
for_model_2 = lm(steam ~ glycerine, data = steam)
summary(for_model_2)$coefficients
```

```
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 6.672118   1.816085 3.673902 0.0012596
## glycerine   3.958403   2.572162 1.538940 0.1374645
```

```
for_model_3 = lm(steam ~ wind, data = steam)
summary(for_model_3)$coefficients
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 6.6214125  1.1236101 5.892981 5.245983e-06
## wind        0.4409357  0.1706511 2.583844 1.659883e-02
```

```
for_model_4 = lm(steam ~ frezday, data = steam)
summary(for_model_4)$coefficients
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 8.4933504  0.3457268 24.566655 5.235197e-18
## frezday     0.1015993  0.0253909  4.001405 5.605820e-04
```

```
for_model_5 = lm(steam ~ temp, data = steam)
summary(for_model_5)$coefficients
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 13.62298927 0.58146349 23.428795 1.496788e-17
## temp        -0.07982869 0.01052358 -7.585697 1.054950e-07
```

The lowest p-value belongs to the temp term, so we add that to the model and repeat with the remaining 4 covariates.

```
for_model_next_1 = lm(steam ~ temp + fat, data = steam)
summary(for_model_next_1)$coefficients
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept)  9.47422200 0.961894080   9.849548 1.589445e-09
## temp        -0.07976077 0.007532872 -10.588361 4.223486e-10
## fat          0.76164829 0.159200936   4.784195 8.895837e-05
```

```
for_model_next_2 = lm(steam ~ temp + glycerine, data = steam)
summary(for_model_next_2)$coefficients
```

```
##                Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) 10.46302293 0.84622584  12.364339 2.233646e-11
## temp        -0.08216043 0.00790277 -10.396408 5.923113e-10
## glycerine    4.72182923 1.08404378   4.355755 2.530172e-04
```

```
for_model_next_3 = lm(steam ~ temp + wind, data = steam)
summary(for_model_next_3)$coefficients
```

```
##                Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) 14.29780562 1.41874039 10.0778167 1.048165e-09
## temp        -0.08420755 0.01357975 -6.2009638 3.057277e-06
## wind        -0.06993211 0.13366832 -0.5231763 6.060815e-01
```

```
for_model_next_4 = lm(steam ~ temp + frezday, data = steam)
summary(for_model_next_4)$coefficients
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 15.44262903 1.33097172 11.602522 7.570537e-11
## temp        -0.10562638 0.01991468 -5.303946 2.531784e-05
## frezday     -0.05051108 0.03344025 -1.510488 1.451497e-01
```

The fat term has the lowest p-value under alpha1, so we add it to the model and repeat with the remaining 3 covariates.

```r
for_model_next_next_1 = lm(steam ~ temp + fat + glycerine, data = steam)
summary(for_model_next_next_1)$coefficients
```

```
##                Estimate  Std. Error     t value     Pr(>|t|)
## (Intercept)  9.51481397 1.062968708   8.9511703 1.299388e-08
## temp        -0.07992826 0.007884145 -10.1378480 1.523605e-09
## fat          0.71359168 0.502297499   1.4206555 1.700955e-01
## glycerine    0.33049680 3.267693788   0.1011407 9.203982e-01
```

```r
for_model_next_next_2 = lm(steam ~ temp + fat + wind, data = steam)
summary(for_model_next_next_2)$coefficients
```

```
##                 Estimate  Std. Error     t value     Pr(>|t|)
## (Intercept)  9.441516755 1.467062684  6.43566008 2.229241e-06
## temp        -0.079576835 0.009841903 -8.08551338 6.941701e-08
## fat          0.762450563 0.165113796  4.61772777 1.484824e-04
## wind         0.002936413 0.097655099  0.03006922 9.762958e-01
```

```r
for_model_next_next_3 = lm(steam ~ temp + fat + frezday, data = steam)
summary(for_model_next_next_3)$coefficients
```

```
##                 Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)  9.841732184 1.65732261  5.9383322 6.797703e-06
## temp        -0.083574482 0.01584329 -5.2750717 3.139613e-05
## fat          0.743542454 0.17543899  4.2381824 3.676151e-04
## frezday     -0.007463963 0.02710321 -0.2753904 7.857079e-01
```

None of the remaining 3 variables have p-values lower than alpha1, so we leave the model as is with fat and temp as predictors for steam, which is the same result we reached with backward selection.

Using best AIC to choose:

```r
full_model = lm(steam ~ fat + glycerine + wind + frezday + temp, data = steam)
best_subset = olsrr::ols_step_best_subset(full_model)
# which.max(best_subset$aic)
predictors.include.aic = strsplit(best_subset$predictors[which.max(best_subset$aic)], "[ ]+", perl = T)
predictors.include.aic
```

```
## [1] "temp"
```

The model with best AIC is model 1, which here only includes temp as a predictor. Using BIC:

```r
# which.max(best_subset$sbic)
predictors.include.bic = strsplit(best_subset$predictors[which.max(best_subset$sbic)], "[ ]+", perl = T)
predictors.include.bic
```

Should get temp/fat

```
## [1] "temp"
```

We have the same result for both AIC and BIC model selection - a model with only temp as a predictor for steam.

20

2. $y_i = f(x_i) + \epsilon_i$, x_i is non-random ($= \text{var}(\epsilon)$)
$E(\epsilon_i) = 0$, $E(\epsilon_i^2) = \sigma^2$, $i = 1, \dots, n$
training set $T = \{y_1, \dots, y_n\}$ to obtain $\hat{f}$, estimate for f.
In-sample test error & training error

$$Err_{in} = E\left[n^{-1} \sum_{i=1}^{n} \{\tilde{y}_i - \hat{f}(x_i)\}^2 \mid T\right] \quad \text{(test MSE)}$$

$$\overline{err} = n^{-1} \sum_{i=1}^{n} \{y_i - \hat{f}(x_i)\}^2 \quad \text{(training MSE)}$$

(a) show that

$$E(Err_{in}) = n^{-1} \sum_{i=1}^{n} \left([\text{Bias}\{\hat{f}(x_i)\}]^2 + \text{var}\{\hat{f}(x_i)\}\right) + \sigma^2$$

From lecture, we know that

$$\text{bias}\{\hat{f}(x)\} = E[\hat{f}(x)] - f(x)$$

and $$\text{var}\{\hat{f}(x)\} = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

we can write $E(Err_{in})$ as:

$$E[E[\tfrac{1}{n} \sum_{i=1}^{n} \{\tilde{y}_i - \hat{f}(x_i)\}^2 \mid T] = E[(\tilde{y} - \hat{f}(x))^2]$$

and if $y = f(x) + \epsilon$, $\tilde{y} = f(x) + \epsilon$ because they have the same distribution...

$$= E[(f(x) + \epsilon - \hat{f}(x))^2]$$
$$= E[(f(x) - \hat{f}(x))^2] + E(\epsilon^2) + 2E[(f(x) - \hat{f}(x))\epsilon]$$
$$= E[(f(x) - \hat{f}(x))^2] + E(\epsilon^2) + 2E[f(x) - \hat{f}(x)]E(\epsilon)$$

we know $E(\epsilon) = 0$ and $E(\epsilon^2) = \sigma^2$

$$= E[(f(x) - \hat{f}(x))^2] + \sigma^2 + 0$$

$$E[(f(x) - \hat{f}(x))^2] = E[((f(x) - E[\hat{f}(x)] - (\hat{f}(x) - E[\hat{f}(x)]))^2]$$
$$= E[(E[\hat{f}(x)] - f(x))^2] + E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$
$$- 2E[(f(x) - E[\hat{f}(x)])(\hat{f}(x) - E[\hat{f}(x)])]$$
$$= \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{var}} + \underbrace{0}_{\text{independence}}$$

so over all,

$$E(Err_{in}) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma^2$$
$$= \tfrac{1}{n} \sum_{i=1}^{n} \left([\text{Bias}(\hat{f}(x_i))]^2 + \text{var}(\hat{f}(x_i))\right) + \sigma^2$$

→

2, continued ⓐ

This is equivalent to minimizing the average MSE $\frac{1}{n}\sum_{i=1}^{n} MSE_i$, $MSE_i = E[\{\hat{f}(x_i) - f(x_i)\}^2]$ because MSE determines the distance of a prediction from its true value, and in optimizing the bias-variance tradeoff with respect to the training set T we are in effect minimizing MSE, as seen in the second half of the proof.

ⓑ Show that expected optimism, $\omega = Err_{in} - \overline{err}$, is

$$E(\omega) = \frac{2}{n}\sum_{i=1}^{n} cov\{y_i, \hat{f}(x_i)\}$$

If y_i & $\tilde{y}_i$ have the same dist, let's use y_i & say $\hat{y}_i = \hat{f}(x_i)$

$$E(\omega) = E[Err_{in} - \overline{err}]$$

$$= E[Err_{in}] - E[\overline{err}]$$

$$= E\left[\frac{1}{n}\sum_{i=1}^{n} E[(y_i - \hat{y}_i)^2]\right] - E\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} E\left[E[(y_i - \hat{y}_i)^2]\right] - E[(y_i - \hat{y}_i)^2]$$

$$= \frac{1}{n}\sum_{i=1}^{n} E[y_i^2] + E[\hat{y}_i^2] - 2E[y_i]E[\hat{y}_i] - E[y_i^2] - E[\hat{y}_i^2] + 2E[y_i\hat{y}_i]$$

$$= 2\cdot\frac{1}{n}\sum_{i=1}^{n} E[y_i\hat{y}_i] - E[y_i]E[\hat{y}_i]$$

$$= \frac{2}{n}\sum_{i=1}^{n} cov\{y_i, \hat{y}_i\} = \frac{2}{n}\sum_{i=1}^{n} cov\{y_i, \hat{f}(x_i)\}$$

This shows us that the training error usually underestimates the test (prediction) error because we can rearrange the optimism to have:

$$E(Err_{in}) = E(\overline{err}) + \underbrace{\frac{2}{n}\sum_{i=1}^{n} cov\{y_i, \hat{f}(x_i)\}}_{\omega \text{ is } > 0}$$

and see that when ω is > 0, the training error underestimates the test error.

→

40

3. Suppose $X = (x_1, \ldots, x_n)^T \in \mathbb{R}^{n \times p}$, $Y = (y_1, \ldots, y_n)^T \in \mathbb{R}^n$, $\lambda \geq 0$

(a) Show that $X^TX + \lambda I_p$ is invertible for all $\lambda > 0$, regardless of X being full-rank.

Remember that $\hat{\beta}^{OLS}$ minimizes $\sum_{i=1}^{n} (y_i - x_i b)^2$
and for full-rank X, $\hat{\beta}^{OLS} = (X^TX)^{-1}X^TY$
Ridge estimator $\hat{\beta}_\lambda$ solves: $\underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i\beta)^2 + \lambda \sum_{j=1}^{p} b_j^2$
for $\lambda > 0$

where the solution is $\hat{\beta}_\lambda = (X^TX + \lambda I_p)^{-1} X^T Y$
from: $(y - X\beta)^T(y - X\beta) + \lambda \beta^T\beta$
the first derivative $\rightarrow \nabla_\lambda = 0$
where $-2X^T(y - X\beta) + 2\lambda\beta = 0$
and $(X^TX + \lambda I_p)\beta = X^TY$
where $(X^TX + \lambda I_p)$ is positive definite
for any $\lambda > 0$ because for any vector
$a \neq 0$, $a^T(X^TX + \lambda I_p)a = (Xa)^T(Xa) + \lambda a^Ta$
$= \sum_{i=1}^{n} (x_i a)^2 + \lambda \sum_{j=1}^{p} a_j^2 > 0$
($\sum (x_i a)^2$: squared, so ≥ 0; $\lambda \sum a_j^2$: squared, so ≥ 0 → always > 0)

and even when $x_i a = 0$, for $\forall i$, $a_k^2 > 0$
making the matrix $(X^TX + \lambda I_p)$ full rank
and invertible. we don't have to assume
X is full-rank, the ridge estimator
$\hat{\beta}^{(ridge)}$ exists either way.

3, continued

(b) $\text{cov}(y_i, y_j) = \sigma^2 1\{i=j\}$

$\hat{y}_\lambda^{(ridge)} = X\hat{\beta}^{(ridge)}(\lambda)$ → $\hat{\beta}^{(ridge)}(\lambda)$ is ridge estimate for β

(i) Find $n \times n$ matrix H_λ that only depends on X and λ such that $\hat{y}_\lambda^{(ridge)} = X\hat{\beta}^{(ridge)}(\lambda) = H_\lambda Y$.
Let df_λ = degrees of freedom in ridge reg. estimator with λ. Conclude $df_\lambda = Tr(H_\lambda)$

We know that $\hat{\beta}^{(ridge)}(\lambda) = (X^TX + \lambda I_p)^{-1} X^TY$

$\hat{y}_\lambda^{(ridge)} = X\hat{\beta}^{(ridge)}(\lambda) = X(X^TX + \lambda I_p)^{-1}X^TY = H_\lambda Y$

making the hat matrix:

$H_\lambda = X(X^TX + \lambda I_p)^{-1}X^T$

From the midterm,

$df = \frac{1}{\sigma^2}\sum_{i=1}^{n} \text{cov}\{\hat{f}(x_i), y_i\}$

$df_\lambda = \frac{1}{\sigma^2}\sum_{i=1}^{n} \text{cov}\{\hat{y}_\lambda^{(ridge)}, y_i\}$

$= \frac{1}{\sigma^2} tr[\text{cov}(\hat{y}_\lambda^{(ridge)}, y_i)]$

$= \frac{1}{\sigma^2} tr[\text{cov}(H_\lambda Y, Y)]$

$= \frac{1}{\sigma^2} tr[H_\lambda \text{var}(Y)]$

$= \frac{1}{\sigma^2} \cdot \sigma^2 \cdot tr(H_\lambda)$

$= tr(H_\lambda)$

(ii) Show that for $0 < \lambda_1 < \lambda_2$, $df_{\lambda_2} < df_{\lambda_1} < \text{rank}(X)$.

Use eigen-decomposition

$X = UDV^T$

$X^TX = VD^2V^T$

$H_\lambda = X(X^TX + \lambda I_p)^{-1}X^T$

$= UDV^T(VD^2V^T + \lambda I_p)^{-1}(UDV^T)^T$

$= UD(D^2 + \lambda I_p)^{-1}DU^T$

From (i) $df_\lambda = tr(H_\lambda)$, here depends on $(D^2 + \lambda I_p)$ → $tr(H_\lambda) = \sum \frac{d_j^2}{\lambda + d_j^2}$

where d_j are diagonal of D

Since $tr(H_\lambda)$ decreases as λ increases,

for $\lambda_1 < \lambda_2$, $df_{\lambda_2} < df_{\lambda_1}$

↳ $tr(H_{\lambda_2}) < tr(H_{\lambda_1})$ →

3, continued

© Show that $PRESS(\lambda) = \sum_{i=1}^{n} \left\{ \frac{y_i - \hat{y}_i^{(\lambda)}}{1 - h_{ii}^{(\lambda)}} \right\}^2$

If $PRESS(\lambda) = \sum_{i=1}^{n} \{ y_i - x_i^T \hat{\beta}_{(-i)}^{(ridge)}(\lambda) \}^2$

and $\hat{y}_i^{(\lambda)}$ is i^{th} element of $\hat{Y}_\lambda^{(ridge)}$

$h_{ii}^{(\lambda)}$ is i^{th} diag of H_λ

$$\{ y_i - x_i^T \hat{\beta}_{(-i)}^{(ridge)}(\lambda) \}^2$$

$$= \{ y_i - x_i^T [\hat{\beta}(\lambda) + \frac{A^{-1} x_i}{1 - h_{ii}} (\hat{y}_i - y_i)] \}^2$$

where $A = X^T X + \lambda I_p$

$$= \{ y_i - \hat{y}_i - \frac{h_{ii}}{1 - h_{ii}} (\hat{y}_i - y_i) \}^2$$

$$= \{ (y_i - \hat{y}_i)(1 + \frac{h_{ii}}{1 - h_{ii}}) \}^2$$

$$= \left\{ \frac{y_i - \hat{y}_i}{1 - h_{ii}} \right\}^2$$

$$\rightarrow \sum_{i=1}^{n} \left\{ \frac{y_i - \hat{y}_i^{(\lambda)}}{1 - h_{ii}^{(\lambda)}} \right\}^2$$

(d) Define corresponding generalized cross validation estimator for λ.

By the answer from ©, the CV estimator for λ would minimize PRESS: $\operatorname{argmin} \sum_{i=1}^{n} \left\{ \frac{y_i - \hat{y}_i^{(\lambda)}}{1 - h_{ii}^{(\lambda)}} \right\}^2$

15

Problem 4

Consider the data Fat.txt. Remove every tenth observation from the data for use as a test sample. Use the remaining data to fit (i.e. train) the following models where % body fat, siri, is the response and all other variables are predictors:

(i) A simple linear model.

(ii) Ridge regression, where the tuning parameter lambda is chosen with generalized cross validation. See RidgeExample.R for an example of how to do this in R. Plot the generalized cross validation value as a function of lambda so that the minimum value is clearly visible.

Models (i) and (ii):

```
# remove every 10th observation
test = fat[seq(0, nrow(fat), 10), ] # save every 10th obs as training data
train = fat[-seq(0, nrow(fat), 10), ] # save data without every 10th obs as testing data

x_train = model.matrix(siri ~ ., data = train)[, -1]
x_test = model.matrix(siri ~ ., data = test)[, -1]

y_train = train$siri
y_test = test$siri

# simple linear model
model_slr = lm(siri ~ ., data = train)
summary(model_slr)
```

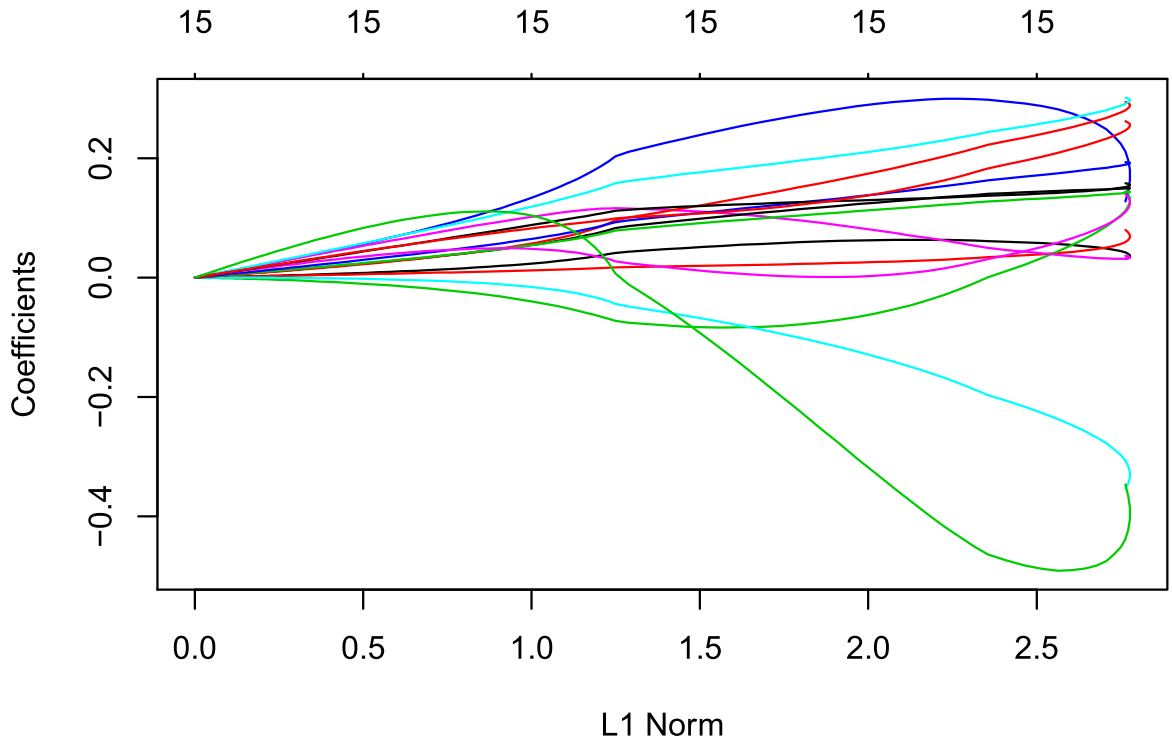
```
##
## Call:
## lm(formula = siri ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8314 -0.6722  0.1828  0.9150  6.6619
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -12.591885   6.448868  -1.953 0.052193 .
## age           0.007978   0.012320   0.648 0.517983
## weight        0.362999   0.023314  15.570  < 2e-16 ***
## height        0.049026   0.040315   1.216 0.225315
## adipos       -0.514032   0.114074  -4.506 1.09e-05 ***
## free         -0.564773   0.014889 -37.933  < 2e-16 ***
## neck          0.016525   0.089863   0.184 0.854272
## chest         0.120219   0.039590   3.037 0.002694 **
## abdom         0.140108   0.042186   3.321 0.001056 **
## hip           0.006197   0.056101   0.110 0.912148
## thigh         0.195057   0.054460   3.582 0.000424 ***
## knee          0.106637   0.093534   1.140 0.255542
## ankle         0.125118   0.081303   1.539 0.125325
## biceps        0.096199   0.064656   1.488 0.138278
## forearm       0.230775   0.073332   3.147 0.001888 **
## wrist         0.139279   0.206804   0.673 0.501378
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.55 on 211 degrees of freedom
## Multiple R-squared:  0.9692, Adjusted R-squared:  0.967
## F-statistic: 442.5 on 15 and 211 DF,  p-value: < 2.2e-16
```

```r
# training error for slr model
pred_slr = predict(model_slr, train)
mean((y_train - pred_slr)^2)
```

```
## [1] 2.232976
```

```r
# ridge regression
ridge_modt = glmnet(x_train, y_train, alpha = 0) # no designated lambda
plot(ridge_modt)
```



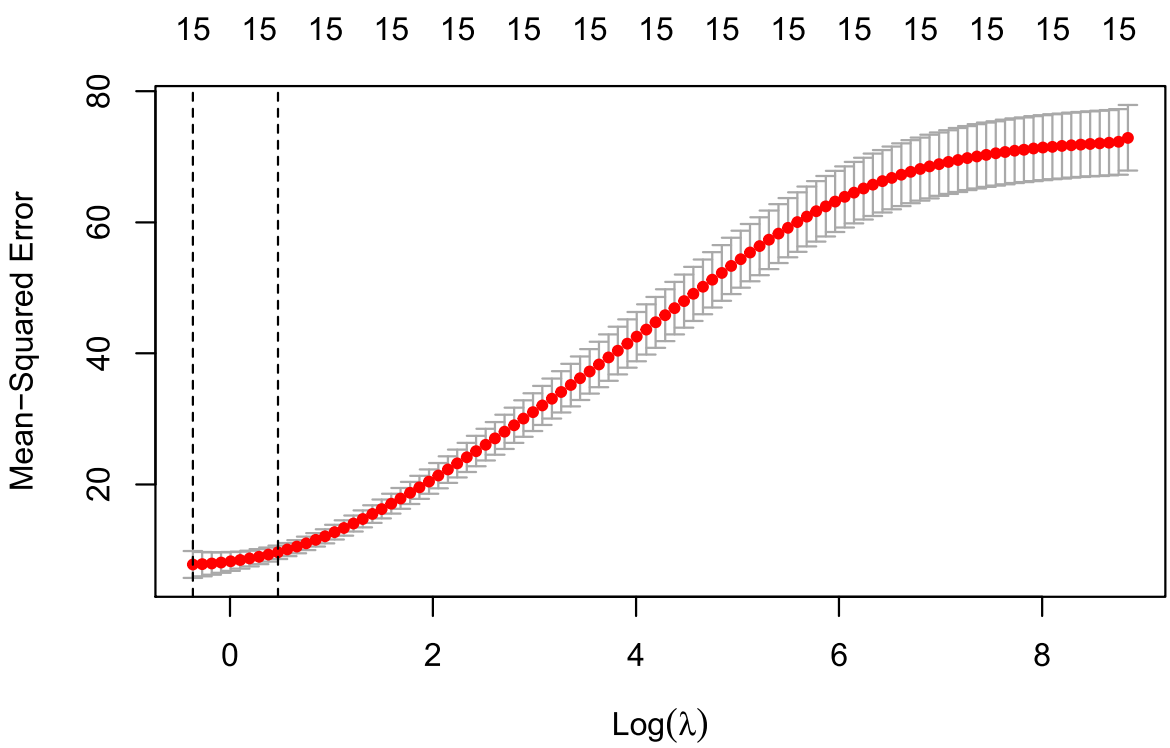

```r
# use cv.glmnet to find optimal lambda
cv.out = cv.glmnet(x_train, y_train, alpha = 0)
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.6941839
```

Use lm.ridge function - find lambda to be 0.046

Linear model produces smaller training error, but largest test error

```r
plot(cv.out) # see optimal lambda
```




```r
ridge_modl = glmnet(x_train, y_train, lambda = bestlam) # bring in best lambda for ridge model fit
summary(ridge_modl)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      15     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       4     -none-    call
## nobs       1     -none-    numeric
```

```r
# training error for ridge model
pred_ridge = predict(ridge_modl, newx = x_train)
mean((y_train - pred_ridge)^2)
```

```
## [1] 10.91856
```

(a) Which model has the smaller training error (see problem 2)? Why is training error a poor judge of how well the model will predict future data?

The simple linear regression model has a smaller training error. From problem 2, we saw that the training error usually underestimates the test/prediction error, so it is not a great judge of accuracy for predicting future data.

(b) Now use the models you fit in (i) and (ii) to predict the held out data. Which model performs better? Clearly indicate the metric (i.e. loss function) you used to judge model performance. (Hint: since you are using squared loss to choose lambda, you might want to use squared loss to judge prediction...)

Since we are comparing the model prediction against the test data (held out), we want to calculate the test MSE for each model. We are looking for the smaller MSE to tell us which model performs better at predicting % body fat (siri).

```
# SLR model
pred_test = predict(model_slr, test)
mean((y_test - pred_test) ^ 2) # test MSE
```

```
## [1] 1.280357
```

```
# RIDGE model
ridge_pred_test = predict(ridge_modl, s = bestlam, newx = x_test)
mean((y_test - ridge_pred_test ) ^ 2) # test MSE
```

```
## [1] 8.247045
```

With the lower MSE belonging to the SLR model, we can conclude that the simple linear regression model performs better when fitting to predict % body fat (siri).