2131 HW10

88/100

Orly Olbum

Problem 1

Consider the data set "NIR" in the R package 'chemometrics', which contains the first derivatives (with respect to wavelength) of near infrared spectroscopy (NIR) absorbance values at p = 235 wavelengths between 1115-2285nm. The goal is to use these covariates to predict the glucose concentration in n = 166 alcoholic fermentation mashes of feedstock. The columns in the covariate matrix NIR$xNIR are arranged in order of increasing wavelength.

10 *(a) Concentration is typically right skewed, and often times must be transformed to meet linear modeling assumptions. Use ordinary least squares to regress glucose concentration onto ten randomly chosen predictors. Using the results from this regression, do you think a transformation is warranted? Explain. (Hint: To make results as interpretable as possible, it is usually best to avoid complex transformations if possible.)*

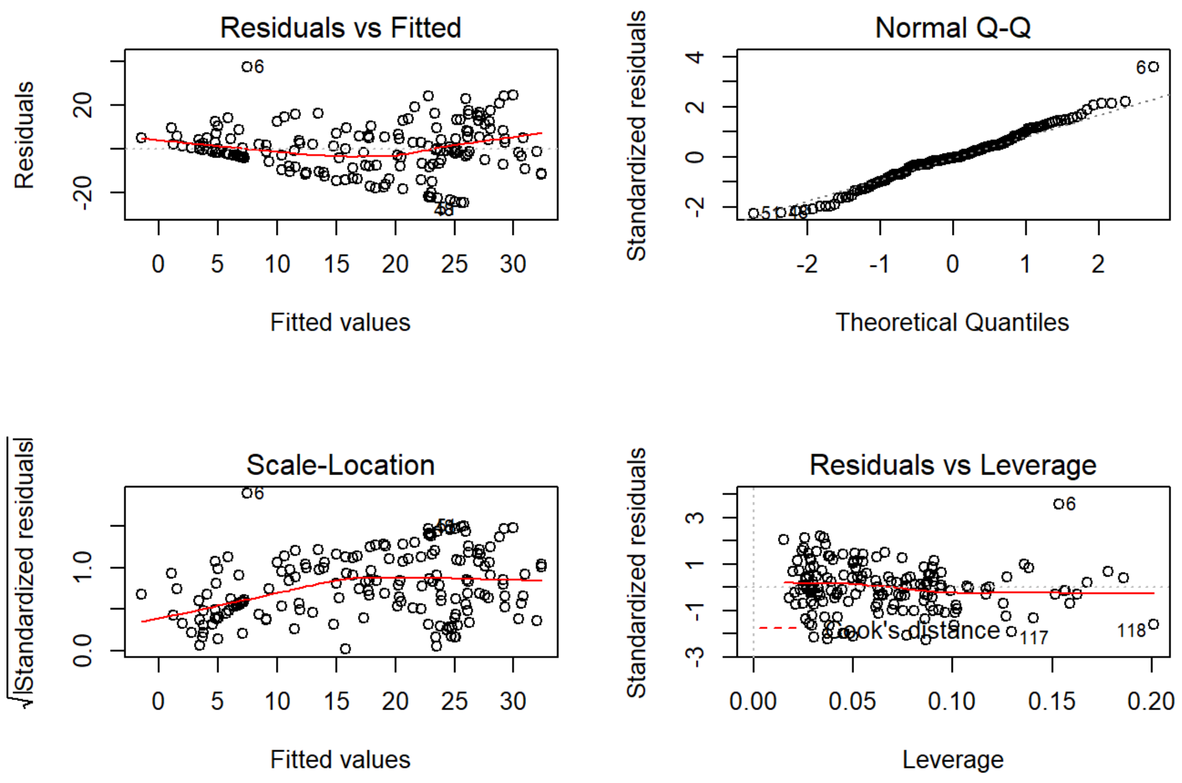
```
data(NIR)
glucose = NIR$yGlcEtOH$Glucose
pred1 = NIR$xNIR$X1115.0; pred2 = NIR$xNIR$X1125.0; pred3 = NIR$xNIR$X1135.0
pred4 = NIR$xNIR$X1145.0; pred5 = NIR$xNIR$X1155.0; pred6 = NIR$xNIR$X1165.0
pred7 = NIR$xNIR$X1175.0; pred8 = NIR$xNIR$X1185.0; pred9 = NIR$xNIR$X1195.0
pred10 = NIR$xNIR$X1205.0

newdata = data.frame(glucose, pred1, pred2, pred3, pred4, pred5, pred6, pred7, pred8,
                     pred9, pred10)

model = lm(glucose ~ ., data = newdata)
summary(model)$r.squared
```

```
## [1] 0.3970771
```

```
par(mfrow = c(2, 2))
plot(model)
```



There are some indications in the residual plots that transforming the predictors might make the model a bit smoother, and considering the R^2 is so low, we may want to do this. Additionally, the residual plots show *potential* outliers, but they look close enough that we would worry too much. None of the plots are alarming enough to jump to this conclusion, and we may be okay with this linear model.

10

(b) Let yi be the glucose concentration in fermentation mash i. Can ordinary least squares be used to estimate the parameters in the model (below) where Aj is the first derivative of the absorbance spectrum at wavelength j? If not, can you suggest four other methods that we've looked at in class that might be used to estimate B1, … , Bp in this model?

$$y_i = \sum_{j=1}^{p} \beta_j A_j + \epsilon_i, \quad i = 1, \ldots, n$$

OLS cannot be used because p > n, which additionally cancels out other options for estimating B such as leave-one-out CV.

Four alternate methods for estimating B1, … , Bp:

1. Kernel Estimation: take into account leverage scores of points that appear further away
2. Bootstrap: we still have mostly constand variance, so we can use boostrap estimation
3. Ridge Regression: if the model isn't fitting because of multicollinearity in the predictors, we might want to use ridge to take care of this - and it is in fact possible, given the type of predictor we have. Alternatively, using lasso would allow coefficients to be reduced to zero, which ridge does not allow
4. LARS: take into account OLS nature but also the residuals, a combo of keeping the lm and also taking into account the residuals we aren't quite happy with yet

In the following questions, permute the observations by using the seed "1968", and then use the first 126 values for training and the last 40 values for testing.

20

(c) Use the training set and principal component regression, using 9-fold cross validation to estimate the number of components, to estimate B from Model (1). That is, choose the number of components ^K to be (below) where xi is the ith row of the 126 x 235 covariate matrix and ^B(k)(-f) is PCR's estimate for B with k components using data from folds 1, … , f-1, f+1,

…, #folds = 9. (Remember that you need to account for the intercept!)

$$\mathcal{L}(k) = \frac{1}{126}\sum_{f=1}^{9}\sum_{i\in\text{fold } f}(y_i - x_i^T\hat{\beta}_{(-f)}^{(k)})^2$$

$$\hat{K} = \underset{k\in\{0,\ldots,\min(n-1,p)\}}{\arg\min}\ \mathcal{L}(k),$$

(i) Plot L(k) as a function of k. What is ^K ?

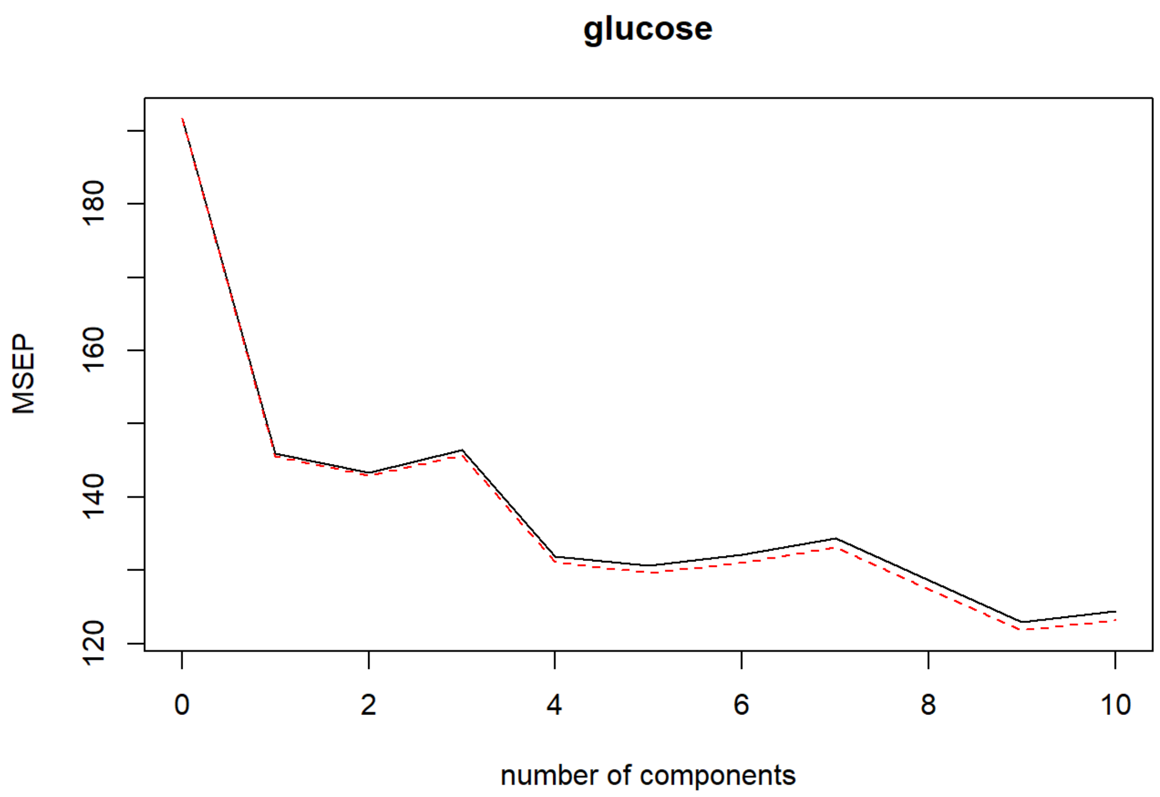
```
par(mfrow = c(1, 1))
# shuffle rows
set.seed(1968)
rows = sample(nrow(newdata))
newdata = newdata[rows, ]

# extract first 126 rows as training data, last 40 as testing data
train = newdata[1:126,]
test = newdata[127:166,]
x_train = model.matrix(glucose ~ ., train)[, -1]
x_test = model.matrix(glucose ~ ., test)[, -1]
y_train = train[,1]
y_test = test[,1]

# CV to estimate beta using training data
pcr_model = pcr(glucose ~ ., data = train, scale = TRUE, validation = "CV")
summary(pcr_model)
```

```
## Data:    X dimension: 126 10
##  Y dimension: 126 1
## Fit method: svdpc
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           13.84    12.08    11.97    12.10    11.48    11.43    11.50
## adjCV        13.84    12.07    11.96    12.07    11.45    11.39    11.45
##        7 comps  8 comps  9 comps  10 comps
## CV       11.59    11.34    11.09     11.15
## adjCV    11.54    11.29    11.04     11.10
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          82.02    93.89    98.99    99.54    99.94    99.98    99.99   100.00
## glucose    25.32    27.70    29.18    36.95    40.63    40.64    40.69    43.38
##          9 comps  10 comps
## X         100.00    100.00
## glucose    46.18     46.33
```

```
validationplot(pcr_model, val.type = "MSEP")
```



```
# min error occurs at comps = 9

pcr_pred = predict(pcr_model, x_test, ncomp = 9)
mean((pcr_pred - y_test)^2)
```

```
## [1] 207.967
```

We can see that the CV error halts when 10 components are used, which is everything. We can also see that the lowest error occurs at M = 9 components, and that using all components increases % of variance to 100. We now have k, which is the minumum of the function above, 9 - where the error fails when PCR is fitted onto the model.

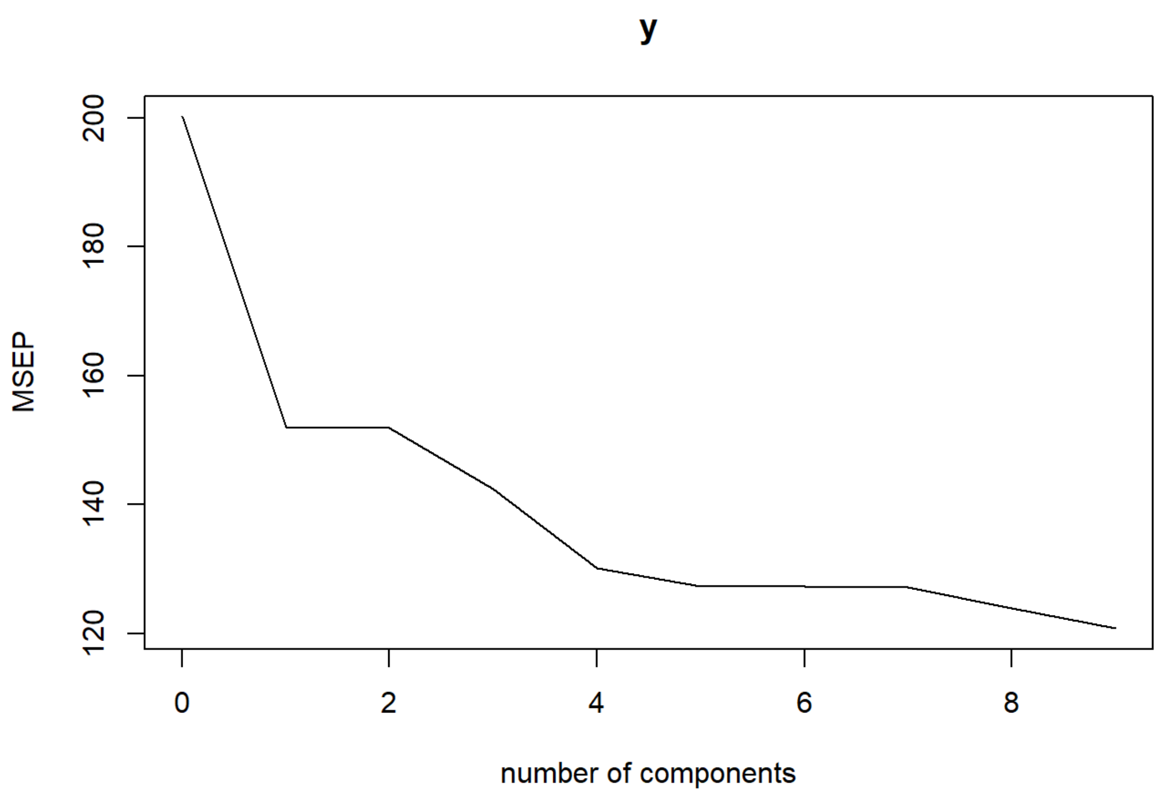
(ii) Plot your estimate for B as a function of wavelength lambda. What do you conclude?

```
x = model.matrix(glucose ~ ., newdata)[, -1]
y = newdata[, 1]

pcr_fit = pcr(y ~ x, scale = TRUE, ncomp = 9)
summary(pcr_fit)
```

```
## Data:    X dimension: 166 10
##  Y dimension: 166 1
## Fit method: svdpc
## Number of components considered: 9
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    80.88    92.53    98.94    99.51    99.94    99.98    99.99   100.00
## y    24.03    24.12    28.82    35.00    36.41    36.41    36.43    38.08
##    9 comps
## X   100.00
## y    39.65
```

```
validationplot(pcr_fit, val.type = "MSEP")
```



When we apply the results from (i) to the full dataset, we see that the error does fail at 9 components.

(iii) Repeat part (i) using leave one out cross validation instead of 9-fold cross validation. How does the loss compare to part (i)?

```
pcr_model_loo = pcr(glucose ~ ., data = train, scale = TRUE, validation = "LOO")
pcr_pred_loo = predict(pcr_model_loo, x_test)
mean((pcr_pred_loo - y_test)^2)
```

```
## [1] 197.374
```

The loss is lower than that of (i), which would indicate LOO is a more accurate prediction method than k-fold CV above.

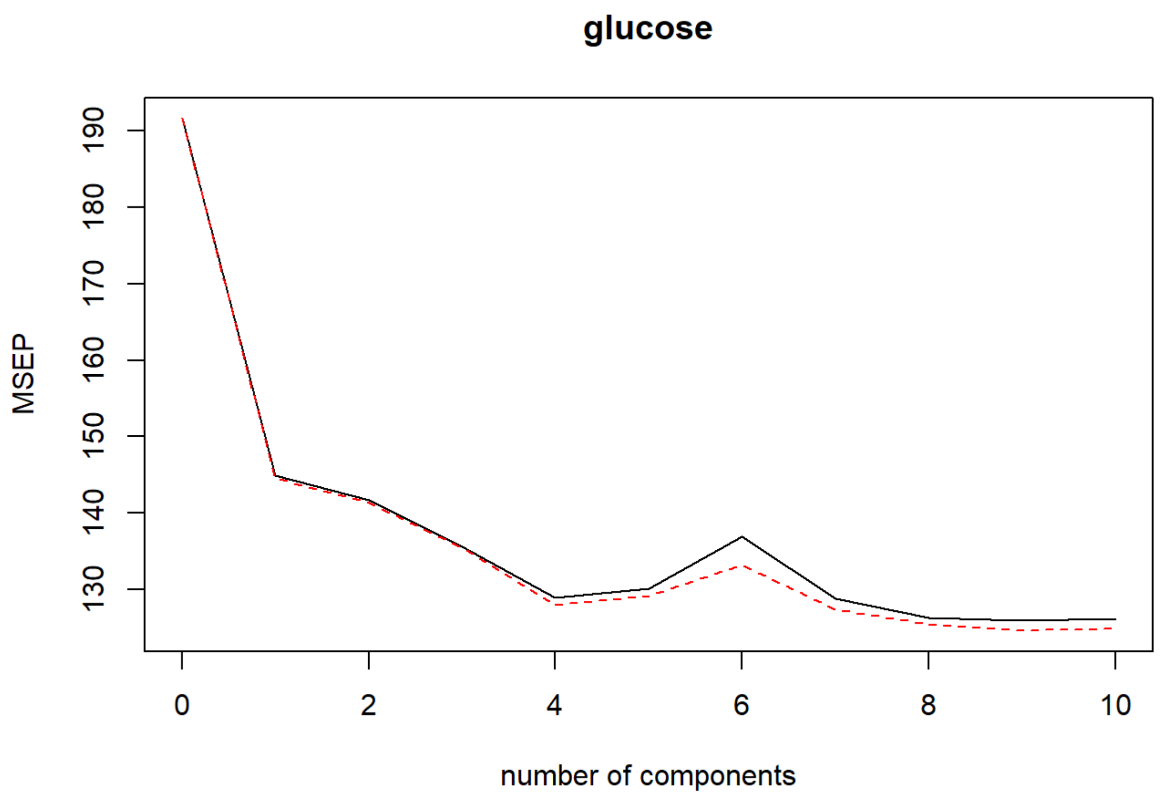
(d) Repeat (c), but with partial least squares.

2D

```
pls_fit = plsr(glucose ~ ., data = train, scale = TRUE, validation = "CV")
summary(pls_fit)
```

```
## Data:    X dimension: 126 10
##  Y dimension: 126 1
## Fit method: kernelpls
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           13.84    12.04    11.90    11.64    11.35    11.41    11.70
## adjCV        13.84    12.02    11.89    11.64    11.31    11.37    11.54
##        7 comps  8 comps  9 comps  10 comps
## CV       11.35    11.24    11.22     11.23
## adjCV    11.28    11.20    11.16     11.17
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          81.87    92.97    98.06    99.51    99.94    99.95    99.98    100.0
## glucose    26.29    30.13    33.83    40.54    40.78    44.00    44.89     45.4
##          9 comps  10 comps
## X          100.0    100.00
## glucose     46.3     46.33
```

```
validationplot(pls_fit, val.type = "MSEP")
```



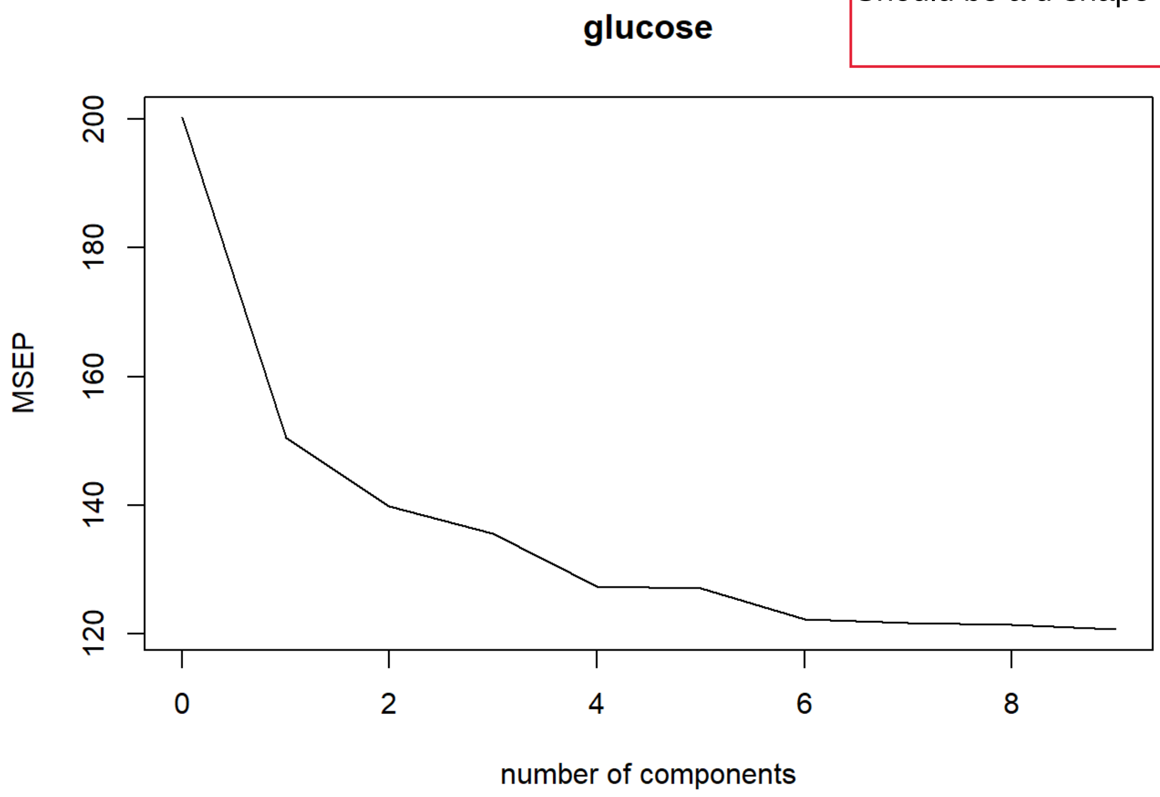
With partial least squares (PLS), the lowest cross-validated error occurs when M = 9.

```
pls_fit2 = plsr(glucose ~ ., data = newdata, scale = TRUE, ncomp = 9)
summary(pls_fit2)
```

```
## Data:    X dimension: 166 10
##  Y dimension: 166 1
## Fit method: kernelpls
## Number of components considered: 9
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          80.79    87.49    96.16    99.49    99.94    99.94    99.96    99.99
## glucose    24.88    30.16    32.26    36.39    36.49    38.87    39.16    39.29
##          9 comps
## X         100.00
## glucose    39.69
```

```
validationplot(pls_fit2, val.type = "MSEP")
```

Should be a u-shape

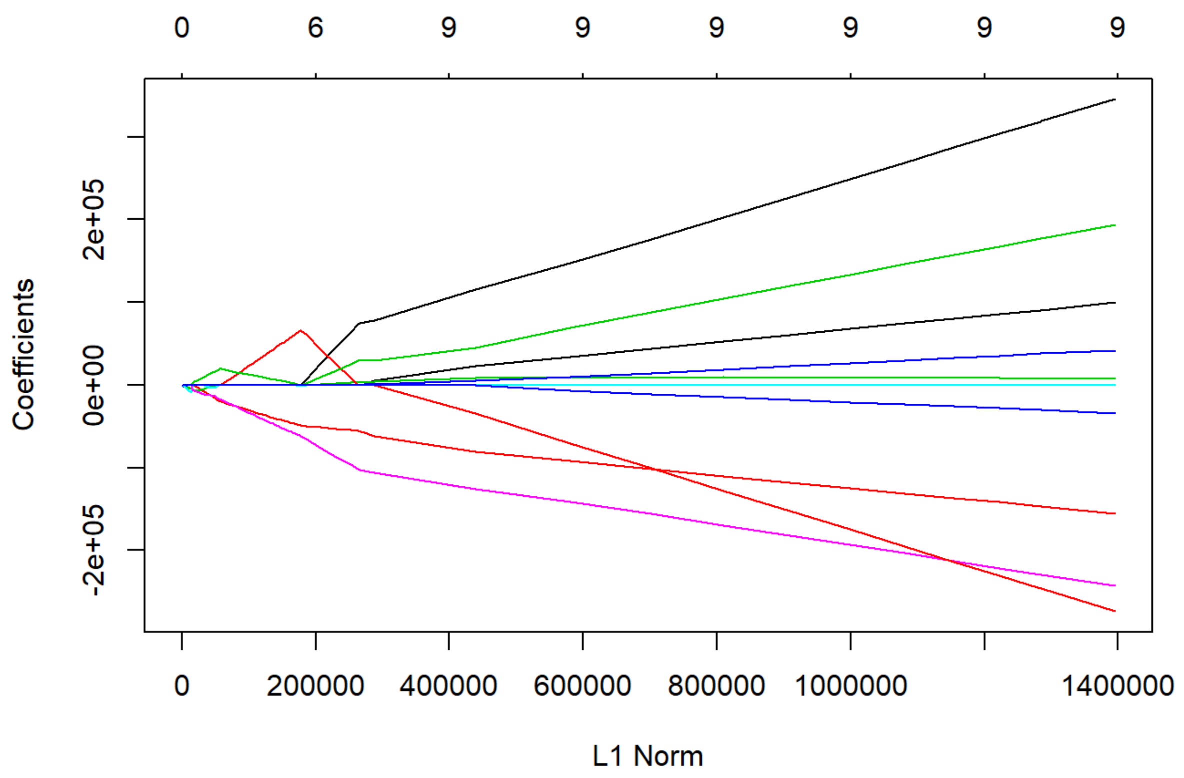


The plot of the components of the main dataset shows the error fail at M = 9 for PLS.

(e) Now use LASSO to estimate B with lambda chosen with 9-fold cross validation. Plot L(lambda) as a function of log(lambda).

10

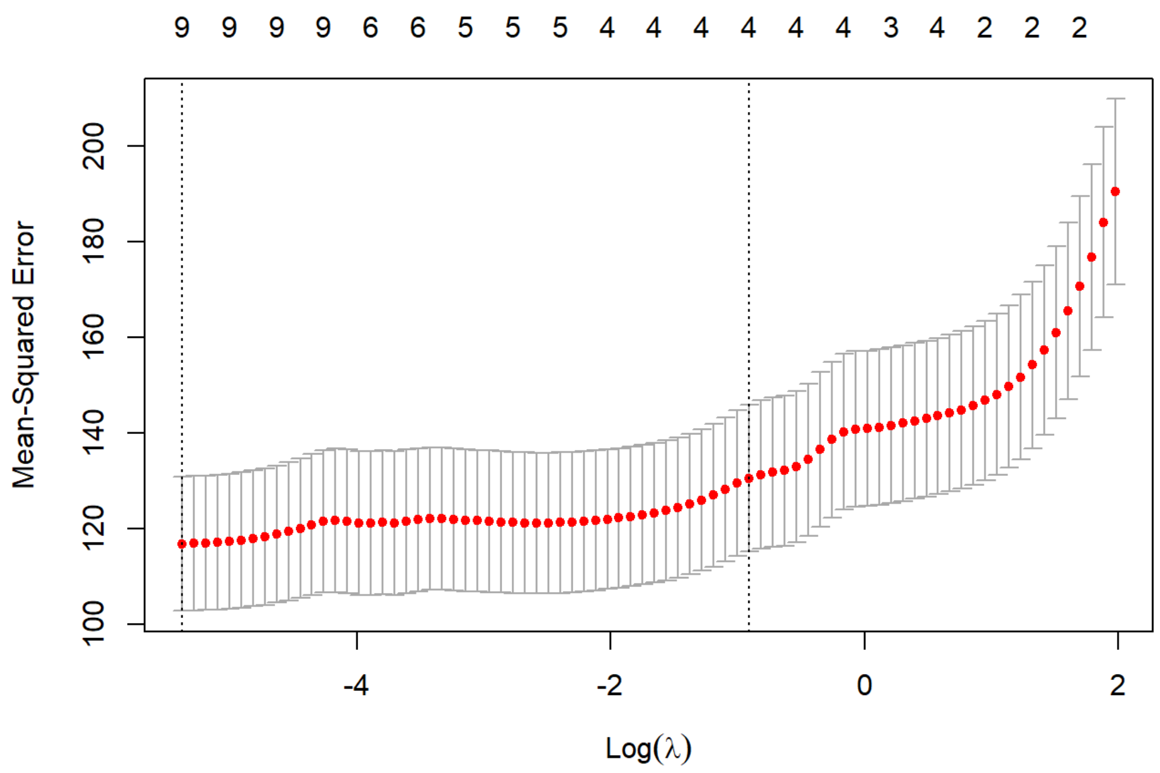
```
lasso_mod = glmnet(x_train, y_train, alpha = 1)
plot(lasso_mod)
```



```
# use cv.glmnet to find optimal lambda
cv.out = cv.glmnet(x_train, y_train, alpha = 1)
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.004629302
```

```
plot(cv.out)
```



```
lasso_mod_2 = glmnet(x_train, y_train, lambda = bestlam, alpha = 1)
summary(lasso_mod_2)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      10     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       5     -none-    call
## nobs       1     -none-    numeric
```

```
# fit lasso on full model
lasso_full = glmnet(x, y, alpha = 1, lambda = bestlam)
lasso_coef = predict(lasso_full, type = "coefficients", s = bestlam)
lasso_coef
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)      321.8417
## pred1          52495.8913
## pred2        -108498.1143
## pred3          23412.5787
## pred4          15695.4447
## pred5               .
## pred6        -151307.5565
## pred7         225233.2379
## pred8        -219852.3458
## pred9         204728.9661
## pred10        -43133.1777
```

The plot shows log(lambda) and we have an optimal lambda shown in the code above.

8

(f) Use the test data to evaluate PCR's, PLS's, and LASSO's predictive performance on this dataset. Comment on L's ability to estimate the testing error.

```
# PCR test error
pcr_pred = predict(pcr_model, x_test, ncomp = 4)
mean((pcr_pred - y_test)^2)
```

```
## [1] 170.5175
```

```
# PLS test error
pls_pred = predict(pls_fit, x_test, ncomp = 2)
mean((pls_pred - y_test)^2)
```

```
## [1] 204.2948
```

```
# lasso test error
lasso_pred = predict(lasso_full, x_test)
mean((lasso_pred - y_test)^2)
```

```
## [1] 160.4873
```

It looks like the best predictive performance belongs to the lasso estimate of beta, with the lowest test error, indicating that L was not the best prediction estimate.

Lasso also used L

With L function, test error is underestimated