Chapter 6 in ISLR: Part 3 Dimension Reduction Methods

Laura Kapitula

11/12/2020

```
library(ISLR) #contains the credit data
library(tidyverse)
library(pls)
theme_set(theme_classic())
```

In section 6.3 of an "Introduction to Statistical Learning with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, Dimension Reduction methods are discussed.

Dimension Reduction Methods

- The methods that we have discussed so far in this chapter have involved fitting linear regression models, via least squares or a shrunken approach, using the original predictors, $X_1, X_2, ..., X_p$.
- Next we explore a class of approaches that transform the predictors and then fit a least squares model using the transformed variables. These techniques are referred to as dimension reduction methods.
- Let $Z_1, Z_2, ...Z_M$ represent $M < p$ linear combinations of our original p predictors. That is,

$$Z_m = \sum_{j=1}^{p} \phi_{mj} X_j$$

for some constants $\phi_{m1}, \phi_{m2}, ..., \phi_{mp}$.

- We can then fit the linear regression model,

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, i = 1, ..., n$$

using ordinary least squares.

- Note that in model above, the regression coefficients are given by $\theta_0, \theta_1, ..., \theta_M$. If the constants $\phi_{m1}, ..., \phi_{mp}$ are chosen wisely, then such dimension reduction approaches can often outperform OLS regression.
- Note that since the Z_m are linear functions of the X_j the model above can be thought of as a special case of the original linear regression model.
- If we use all p of the Z_m the model is equivalent to the OLS model. We just took a transformation of the original X variables.
- The dimension reduction comes in when we use $M < p$ this reduction in dimension is another way to constrain the estimated β_j coefficients and thus reduce variance of the estimated model. (ie win the variance-bias trade-off)

Principal Components Regression

- We can apply principal components analysis (PCA) (discussed in Chapter 10 of ISLR) to define the linear combinations of the predictors, for use in our regression.
- The first principal component is that (normalized) linear combination of the variables with the largest variance.
- The second principal component has largest variance, subject to being uncorrelated with the first.
- And so on.
- The idea is that when we have a lot of correlated original variables, we replace them with a small set of principal components (new variables) that capture their joint variation, that are all uncorrelated with each other.

We will start by giving this a try on the Credit data.

```
data(Credit, package = "ISLR")
Credit=Credit[2:12]
```

The syntax for the `pcr()` function is similar to that for `lm()`, with a few additional options. Setting `scale=TRUE` has the effect of standardizing each predictor prior to generating the principal components, so that the scale on which each variable is measured will not have an effect. Setting `validation="CV"` causes `pcr()` to compute the ten-fold cross-validation error for each possible value of M, the number of principal components used. As usual, we'll set a random seed for consistency:

```
set.seed(2)
pcr_fit = pcr(Balance~., data = Credit, scale = TRUE, validation = "CV")
```

The resulting fit can be examined using the `summary()` function:

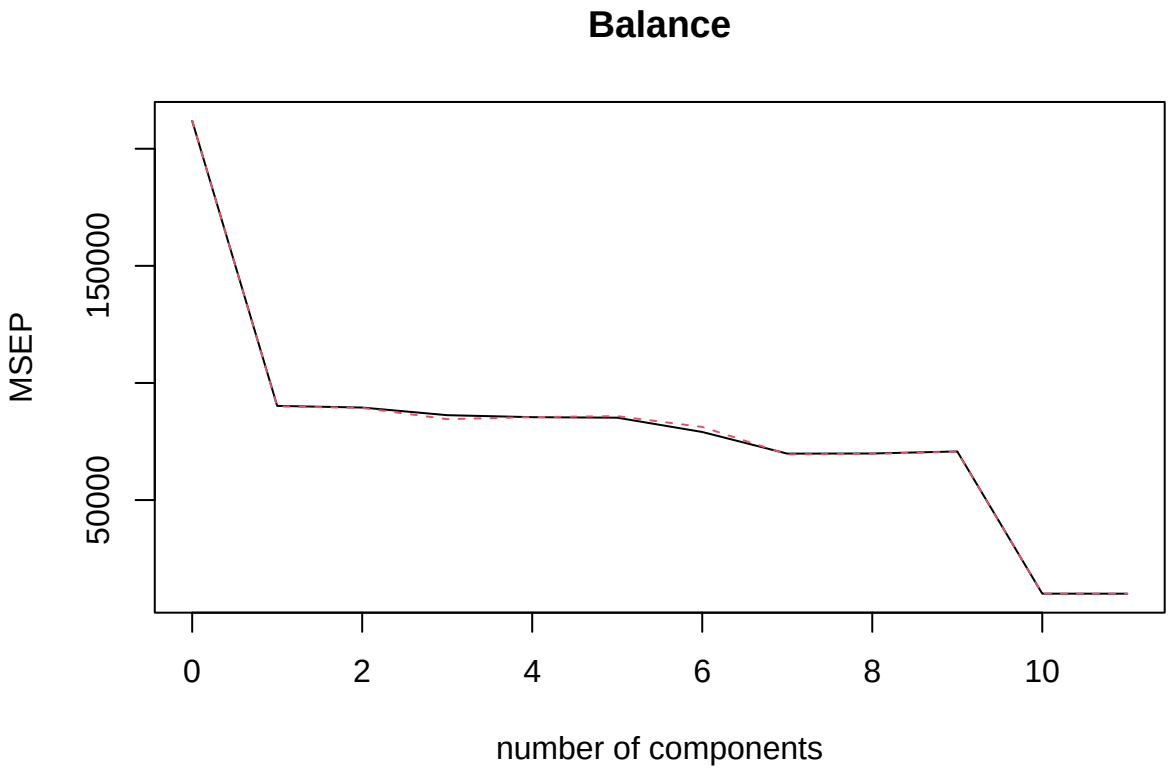
```
summary(pcr_fit)
```

```
## Data:    X dimension: 400 11
##  Y dimension: 400 1
## Fit method: svdpc
## Number of components considered: 11
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           460.3    300.3    299.2    293.7    292.2    291.8    281.2
## adjCV        460.3    300.0    298.9    290.8    292.3    293.0    285.0
##        7 comps  8 comps  9 comps  10 comps  11 comps
## CV       264.2    264.4    266.0     100.2    100.11
## adjCV    263.6    264.0    265.8     100.1     99.97
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          25.05    39.64    49.73    59.74    68.89    77.73    86.43    93.91
## Balance    58.07    58.37    60.78    60.90    61.46    63.11    68.70    68.71
##          9 comps  10 comps  11 comps
## X          97.60     99.98    100.00
## Balance    68.72     95.47     95.51
```

The CV score is provided for each possible number of components, ranging from $M = 0$ onwards. Note that `pcr()` reports the **root mean squared error**; in order to obtain the usual MSE, we must square this quantity.

One can also plot the cross-validation scores using the `validationplot()` function. Using `val.type="MSEP"` will cause the cross-validation MSE to be plotted:

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



We see that the smallest cross-validation error occurs when $M = 10$ components are used. This is barely fewer than $M = 11$, which amounts to simply performing least squares, because when all of the components are used in PCR no dimension reduction occurs.

You might have noticed that the `summary()` function also provides the percentage of variance explained in the predictors and in the response using different numbers of components.We can think of this as the amount of information about the predictors or the response that is captured using M principal components. For example, setting $M = 1$ only captures 25% of all the variance, or information, in the predictors. In contrast, using $M = 6$ increases the value to 77.7%. If we were to use all $M = p = 11$ components, this would increase to 100%.

Partial Least Squares (PLS)

- PCR identifies linear combinations, or directions, that best represent the predictors $X_1, ..., X_p$
- These directions are identified in an unsupervised way, since the response Y is not used to help determine the principal component directions.
- That is, the response does not supervise the identification of the principal components.
- Consequently, PCR suffers from a potentially serious drawback: there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.
- Like PCR, PLS is a dimension reduction method, which first identifies a new set of features $Z_1, ...Z_M$ that are linear combinations of the original features, and then fits a linear model via OLS using these M new features.
- But unlike PCR, PLS identifies these new features in a supervised way { that is, it makes use of the response Y in order to identify new features that not only approximate the old features well, but also that are related to the response.

- Roughly speaking, the PLS approach attempts to find directions that help explain both the response and the predictors.

Details of PLS

- After standardizing the p predictors, PLS computes the first direction Z_1 by setting each ϕ_{1j} in

$$Z_m = \sum_{j=1}^{p} \phi_{mj} X_j$$

equal to the coeficient from the simple linear regression of Y onto X_j .

- One can show that this coefficient is proportional to the correlation between Y and X_j .
- Hence, in computing

$$Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j$$

, PLS places the highest weight on the variables that are most strongly related to the response.
- Subsequent directions are found by taking residuals and then repeating the above prescription.

The rest of this handout is a lab on PCS and PLS in R comes from p. 256-259 of "Introduction to Statistical Learning with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Which was then reimplemented in the `tidyverse` format by Amelia McNamara and R. Jordan Crouser at Smith College. http://www.science.smith.edu/~jcrouser/SDS293/labs/lab11.Rmd

6.7.1 Principal Components Regression with Hitters Data

Principal components regression (PCR) can be performed using the `pcr()` function, which is part of the `pls` library. In this lab, we'll apply PCR to the `Hitters` data, in order to predict `Salary`. As in previous labs, we'll start by ensuring that the missing values have been removed from the data:

```
Hitters = na.omit(Hitters) # Omit empty rows
```

The syntax for the `pcr()` function is similar to that for `lm()`, with a few additional options. Setting `scale=TRUE` has the effect of standardizing each predictor prior to generating the principal components, so that the scale on which each variable is measured will not have an effect. Setting `validation="CV"` causes `pcr()` to compute the ten-fold cross-validation error for each possible value of M, the number of principal components used. As usual, we'll set a random seed for consistency:

```
set.seed(1)
pcr_fit = pcr(Salary~., data = Hitters, scale = TRUE, validation = "CV")
```

The resulting fit can be examined using the `summary()` function:

```
summary(pcr_fit)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV             452    352.5    351.6    352.3    350.7    346.1    345.5
## adjCV          452    352.1    351.2    351.8    350.1    345.5    344.6
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
```

```
## CV       345.4    348.5    350.4     353.2     354.5     357.5     360.3
## adjCV    344.5    347.5    349.3     351.8     353.0     355.8     358.5
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        352.4     354.3     345.6     346.7     346.6     349.4
## adjCV     350.2     352.3     343.6     344.5     344.3     346.9
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26    94.96
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69    46.75
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         96.28     97.26     97.98     98.65     99.15     99.47     99.75
## Salary    46.86     47.76     47.82     47.85     48.10     50.40     50.55
##         16 comps  17 comps  18 comps  19 comps
## X          99.89     99.97     99.99    100.00
## Salary     53.01     53.85     54.61     54.61
```

The CV score is provided for each possible number of components, ranging from $M = 0$ onwards. Note that `pcr()` reports the **root mean squared error**; in order to obtain the usual MSE, we must square this quantity. For instance, a root mean squared error of 352.8 corresponds to an MSE of $352.82 = 124{,}468$.

One can also plot the cross-validation scores using the `validationplot()` function. Using `val.type="MSEP"` will cause the cross-validation MSE to be plotted:

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

We see that the smallest cross-validation error occurs when $M = 16$ components are used. This is barely fewer than $M = 19$, which amounts to simply performing least squares, because when all of the components are used in PCR no dimension reduction occurs. However, from the plot we also see that the cross-validation error is roughly the same when only five component is included in the model. This suggests that a model that uses a smaller number of components might suffice.

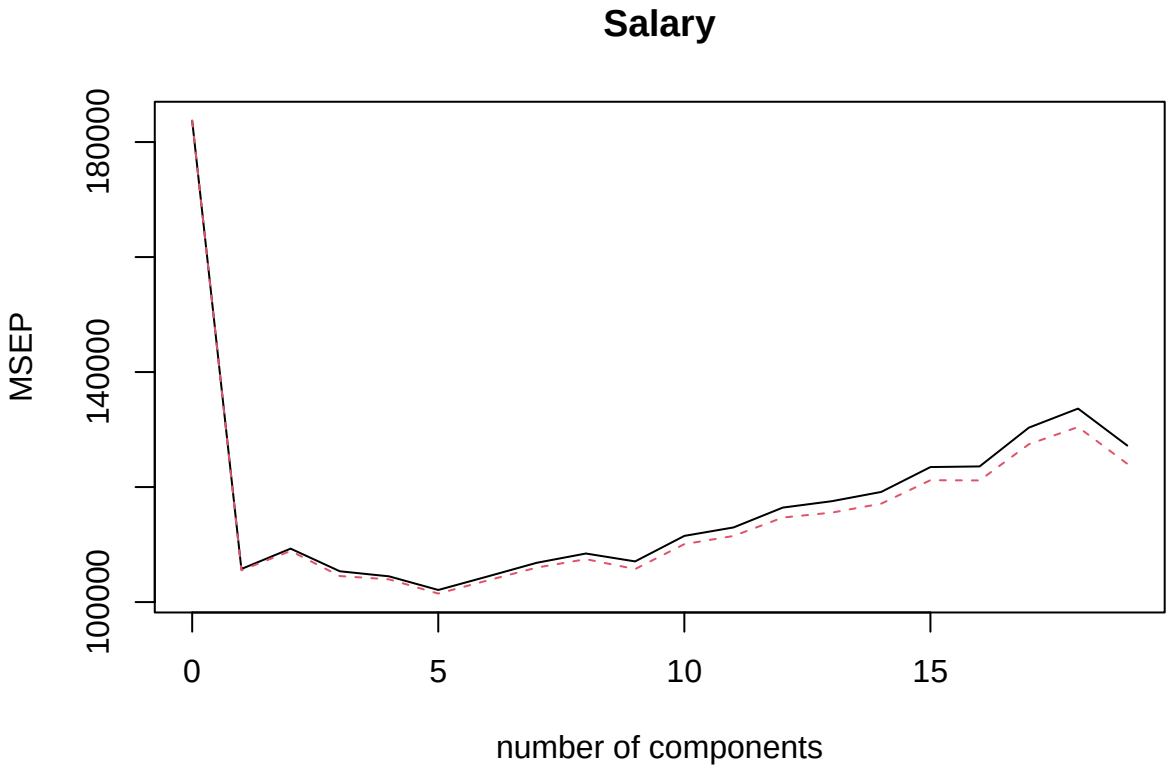
Now let's perform PCR on the training data and evaluate its test set performance:

```
set.seed(1)

train = Hitters %>%
  sample_frac(0.5)

test = Hitters %>%
  setdiff(train)

pcr_fit2 = pcr(Salary~., data = train, scale = TRUE, validation = "CV")
validationplot(pcr_fit2, val.type = "MSEP")
```



We find that the lowest cross-validation error occurs when $M = 5$ components are used. We compute the test MSE as follows:

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

y_train = train %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()

y_test = test %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()

pcr_pred = predict(pcr_fit2, x_test, ncomp=7)
mean((pcr_pred-y_test)^2)
```

```
## [1] 142350.1
```

This test set MSE is competitive with the results obtained using ridge regression and the lasso. However, as a result of the way PCR is implemented, the final model is more difficult to interpret because it does not perform any kind of variable selection or even directly produce coefficient estimates.

Finally, we fit PCR on the full data set using $M = 5$, the number of components identified by cross-validation:

```
x = model.matrix(Salary~., Hitters)[,-1]

y = Hitters %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()

pcr_fit2 = pcr(y~x, scale = TRUE, ncomp = 5)
summary(pcr_fit2)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 5
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps
## X    38.31    60.16    70.84    79.03    84.29
## y    40.63    41.58    42.17    43.22    44.90
```

6.7.2 Partial Least Squares

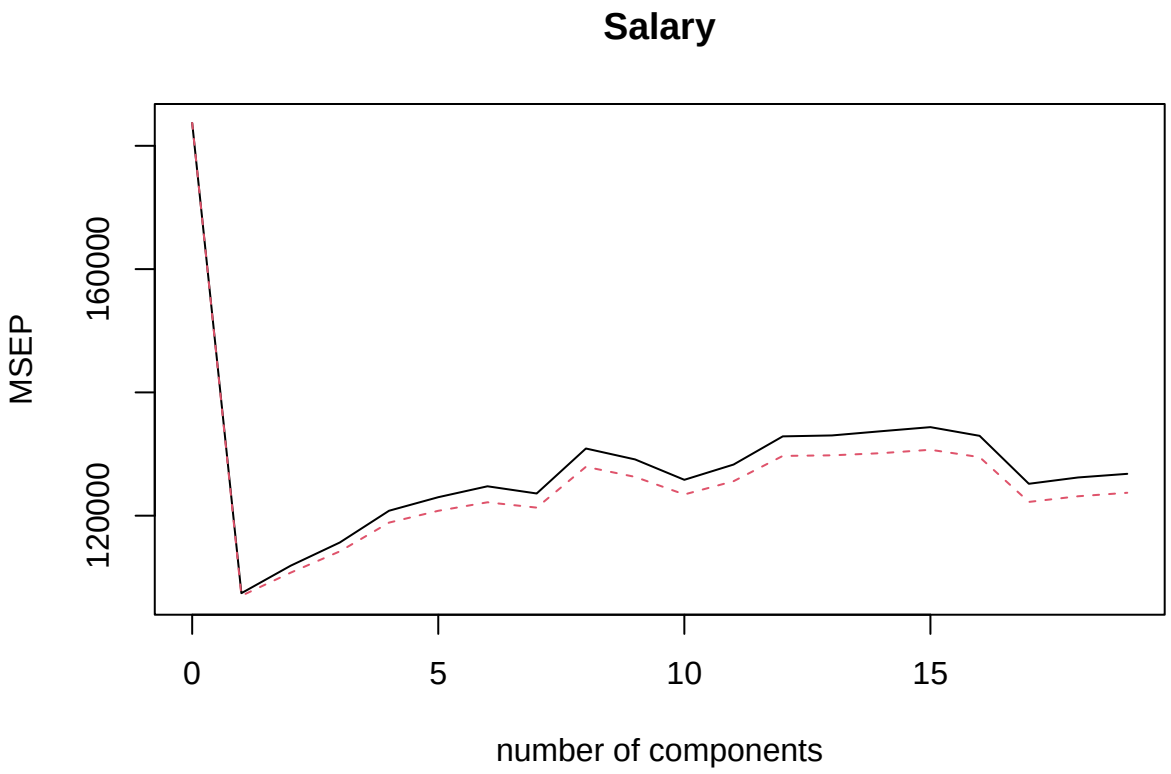
Next we'll implement partial least squares (PLS) using the `plsr()` function, also in the `pls` library. The syntax is just like that of the `pcr()` function:

```
set.seed(1)
pls_fit = plsr(Salary~., data = train, scale = TRUE, validation = "CV")
summary(pls_fit)
```

```
## Data:    X dimension: 132 19
##  Y dimension: 132 1
## Fit method: kernelpls
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           428.6    327.8    334.5    340.1    347.6    350.7    353.2
## adjCV        428.6    327.1    332.8    338.0    344.8    347.6    349.5
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       351.6    361.8    359.3     354.7     358.2     364.5     364.7
## adjCV    348.3    357.7    355.4     351.4     354.4     360.1     360.3
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        365.7     366.6     364.6     353.8     355.2     356.1
## adjCV     360.8     361.5     359.9     349.6     350.9     351.7
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
```

```
## X         39.03    48.78    60.06    74.97    78.19    80.72    87.53    90.46
## Salary    46.53    50.52    51.81    52.52    53.61    54.20    54.47    54.97
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         93.29     95.72     97.11     97.77     98.51     98.80     99.21
## Salary    55.19     55.44     55.82     56.10     56.47     57.33     57.58
##        16 comps  17 comps  18 comps  19 comps
## X         99.60     99.67     99.94    100.00
## Salary    57.72     58.27     58.36     58.43
```

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



The lowest cross-validation error occurs when only $M = 1$ partial least squares dimensions are used. We now evaluate the corresponding test set MSE:

```
pls_pred = predict(pls_fit, x_test, ncomp = 1)
mean((pls_pred - y_test)^2)
```

```
## [1] 153443.7
```

The test MSE is comparable to, but slightly higher than, the test MSE obtained using ridge regression, the lasso, and PCR.

Finally, we perform PLS using the full data set using $M = 1$, the number of components identified by cross-validation:

```
pls_fit1 = plsr(Salary~., data = Hitters, scale = TRUE, ncomp = 1)
summary(pls_fit1)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: kernelpls
## Number of components considered: 1
## TRAINING: % variance explained
```

```
##          1 comps
## X          38.08
## Salary     43.05
```