# Appendix R7B: Dimensionality - Dimension Reduction

## Principal Components and Partial Least Squares Regressions

J. Alberto Espinosa

8/5/2021

## Table of Contents


Dimensionality ..... 1

Principal Components Regression (PCR) ..... 3

- PCR Loadings and Scores ..... 6
- PCR Coefficients ..... 7
- Predictions with PCR ..... 9

Partial Least Squares (PLS) Regression ..... 10

- PLSR Loadings and Scores ..... 11
- PLSR Coefficients ..... 12
- Predictions with PLSR ..... 14

Logistic and GLM Regressions with PCR and PLSR ..... 15


**Technical Note:** All procedures involving machine learning and cross-validation are based on random sampling. Therefore, results may change when you run the analysis multiple times, depending on the kind of random number generator **RNGkind** and the seed selected. The results in this script may not match the outputs in the main text of the book. Follow the chapter and this script appendix independently. Let's first set the random number generator default

```
RNGkind(sample.kind = "default") # To use the R default RNG
```

## Dimensionality

As discussed previously, large and complex models generally experience **dimensionality** issues, causing them to be overfitting and have hight variance. One of the most pervasive dimensionaity issues is multi-collinearity. **Regularized** methods like Ridge and LASSO help us reduce the detrimental effects of multi-collinearity by shrinking (i.e. biasing) the predictors. In contrast, dimension reduction methods don't just help correct for multi-collinearity, but actually take advantage of the collinearity structure of the data. In fact, dimension reduction methods like PCR and PLSR don't do much to improve the model if the multi-collinearity condition in the data is low. But it can improve things dramatically when there is severe multi-collinearity. This family of methods is particularly useful when there are too many predictors relative to the number of observations. The basic idea is this.

If two predictors are uncorrelated, a scatter plot of the two variables will show a somewhat spherical cloud of data points with no clear alignment in one direction or another. In contrast, if two predictors are highly correlated, the scatter plot of these two predictors will show a thin cloud of data points aligned in one direction. The direction of this alignment is the direction in which the variance in the data is the highest. When there are more than 2 predictors, there will be more than one direction of alignment, depending on which variable is correlated with which. If you move the origin of the plot to the mean of all variables and then rotate the axes in the direction of highest variance, second highest, etc. you would be aligning the rotated axes with **1st. Principal Component (1st PC)**, **2nd PC**, etc.

The idea behind dimension reduction is that, for **n predictors**, there are **n PCs** and these PC's have some interesting properties:

1. They are perpendicular to each other. The 2nd PC is found by taking an axis that is perpendicular to the 1st PC and rotating it around it until the direction of second largest variance is found. Same thing with the 3rd PC, and so on.

2. The resulting PCs constructed as linear combinations of the predictors are independent or uncorrelated.

3. The PCs are ordered from highest to lowest variance. This allows you to explore the PCs to find the first **m PCs** that explain a substantial amount of variance in the data.

4. When the predictors are highly correlated, the first few PCs will explain a large proportion of variance in the data, and the last PCs will explain very little.

5. So, the goal is to identify first m PC's that explain, say 70% or 80% of the variance of the predictors and disregard the remaining PCs. If **m << p** you will be achieving substantial dimension reduction, because you can now run a regression with m PC's, rather than with p predictors.

6. All m PC's are linear combinations of all p predictors, so all variables are represented in the PC's. More importantly, the PC's are uncorrelated, so the are truly independent variables.

7. PCR and PLSR will estimate a 1 PC model, 2 PC model, etc., up to a p PC model with all components. The model (either PCR or PLSR) with p PCs will yield identical results to the OLS or GLM model. As the number of PCs in the model decreases, the variance of the model decreases too, but the bias increases. Therefore, models with more PCs are better for interpretations, models with the lowest CV test deviance are best for predictions.

There a few dimension reduction methods, but the two most popular ones are: (1) **Principal Components Regression (PCR)** in which the predictors are rotated to find the PCs, without taking into account whether these dimensions help predict the outcome variable (i.e., unsupervised method); and (2) **Partial Least Squares Regression (PLSR)**, which is similar to PCR, but the axes are further rotated to improve their correlation with the response variable (i.e., supervised method). PCR and PLSR are competing methods of similar king and there is no guarantee that one method will be better than the other. It is recommended to test both and select the one with best results.

## Principal Components Regression (PCR)

I will use the **{pls}** R package for both PCR and PLSR and also the **Hitters** data set from the **{ISLR}** R package, containing baseball player salaries and 19 predictors. Note that the`pcr()` function syntax is similar to the `lm()` function syntax, with a few additional parameters.

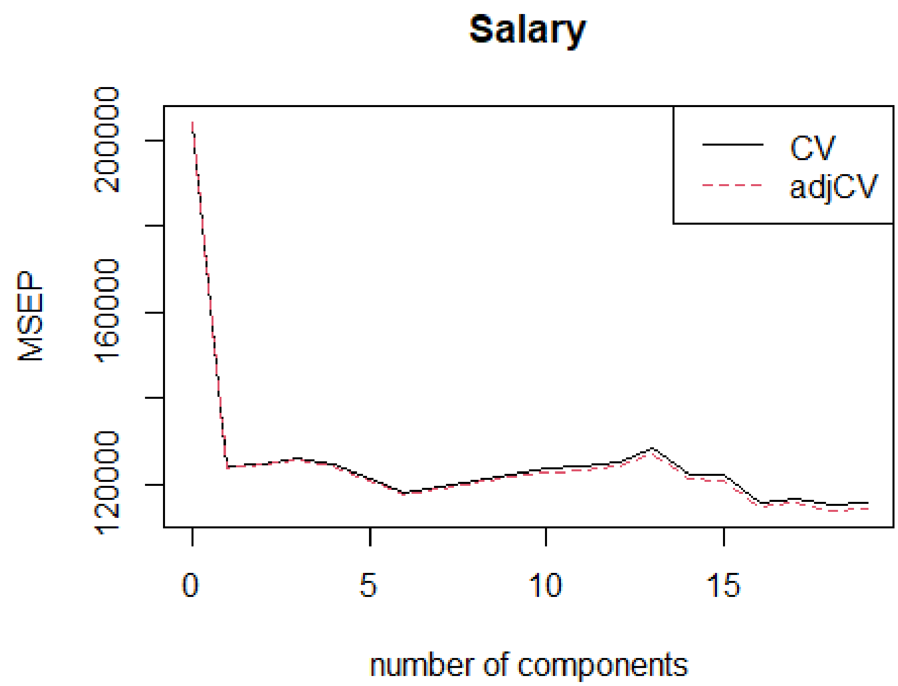
```
library(pls) # Has the Principal Components Regression pcr() function
library(ISLR) # Has the Hitters data set

Hitters <- na.omit(Hitters) # Let's remove missing values

set.seed(2) # Arbitrary
```

Predict Salary with all predictors. `scale = T` standardizes the predictors, which is recommended, and necessary when variables are in different scales (e.g., lbs, feet, etc.). Also, `validation="CV"` does 10FCV and `validation="LOO"` does LOOCV. We use all predictors in the model.

```
pcr.fit <- pcr(Salary ~ ., data = Hitters, scale = T, validation = "CV")
```

The first step is to analyze the model results visually. We do this by plotting the **Scree Plot**, which shows the MSE or RMSE for all the PC models. I'm showint the plot with **MSE** but you can use change the `val.type =` parameter to `"RMSEP"` if you prefer the RMSE. Both graphs are similar, except for the scale of the error.`

```
validationplot(pcr.fit, val.type = "MSEP",
               legendpos = "topright") # Plots the MSE
```




In the Scree plot above, one can see that the MSE drops substantially from 0 PCs (Null model) to 1 PC and it then the graph **elbows** to the right. These sharp elbows to the right are points of interest because they are where we achieve sharp error reductions, with little change afterwards. There is another sharp elbow to the right at 6 PCs and then another one at 16 PCs. These are all models of interest worth exploring further quantitatively. There is an elbow pinting upwards at 13 PCs, but this is not a point of interest, because the MSE turns downwards after that, so it is better to continue going to the right.

Let's analyze the model quantitatively with the `summary()` function.

```
summary(pcr.fit) # Take a look

## 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    351.9    353.2    355.0    352.8    348.4    343.6
## adjCV          452    351.6    352.7    354.4    352.1    347.6    342.7
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       345.5    347.7    349.6     351.4     352.1     353.5     358.2
## adjCV    344.7    346.7    348.5     350.1     350.7     352.0     356.5
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        349.7     349.4     339.9     341.6     339.2     339.6
## adjCV     348.0     347.7     338.2     339.7     337.2     337.6
```

```
##
## 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
```

A few notes about the **summary()** of **PCR** results. First, notice that there are two output panels. The first panel is the **Validation** and the second one is about **explained variance**.

**Validation Output Panel**

1. It shows CV **Square Root (MSE)** values, not MSE. Naturally, you can square these values to get the MSEs. The adjCV is a "bias-corrected" CV. It makes very little difference for our purposes, but adjCV makes some statistical adjustment that may come from sampling bias in CV testing.

2. CV RMSEs go down from 0 (Null model) to 1 component and it then goes up and down slightly until it gets to 6 components with a `CV RMSE = 343.7`. It then goes up and down again, with another low point at 16 components with a `CV RMSE = 343.4`. The absolute lowest point is at 18 components with a `CV RMSE = 341.2`.

** Variance Explained Panel**

1. The **X** row shows how much of the variance of the original predictors (X) is explained by the PCs. For example, **6 PCs** explain `88.63%` of the variance in the predictors. The **1st PC** explains `38.31%` of the variance of the predictors. The model with **2 PCs** explains `60.16%` of the variance of the predictors. This means that the **2nd PC** alone explains `60.16 - 38.31 = 21.85%` of the variance of the predictors. As expected, the 1st PC explains more variance than the 2nd PC, and so on, but cumulatively, more PCs explain increasingly more variance.

2. **Salary** row shows how much of the outcome variable variance is explained by the model. For example, a model with **6 PCs** explain `46.48%` of the variance in Salary. This is equivalent to the model's **R-Squared**.

** Selecting the Best Model**

1. If the analysis goal is predictive accuracy, I would select the model with the lowest CV RMSE or 18 PCs in this case.

2. If the analysis goal is interpretability, I would select the largest possible model with an acceptable CV RMSE, which in this case is also 18 PCs.

3. If the analysis goal is to substantially reduce the dimension of the model, I would select the model with the smallest number of PC's that explain at least `70%` of the varianec of

the predictors, or 3 PCs in this case, which explains `70.84%` of the variance in the predictors, which is a good representation of the data.

## PCR Loadings and Scores

The **pcr()** object is very complex and is full of information. You can explore this object's results with `str(pcr.fit)`. If you run this function, you will see an object attribute named **$loadings**, which is a data frame containing one column for each component model and one row for each predictors. I'm only listing the first 8 PC models for simplicity, but you can remove the `[,1:8]` index to see all of them.

```
round(pcr.fit$loadings[,1:8], digits = 3)
```

```
##            Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
## AtBat       0.198  0.384 -0.089  0.032 -0.028  0.071 -0.107  0.270
## Hits        0.196  0.377 -0.074  0.018  0.005  0.082 -0.130  0.389
## HmRun       0.204  0.237  0.216 -0.236 -0.078  0.150  0.506 -0.226
## Runs        0.198  0.378  0.017 -0.050  0.039  0.137 -0.202  0.115
## RBI         0.235  0.315  0.073 -0.139 -0.024  0.112  0.319  0.005
## Walks       0.209  0.230 -0.046 -0.131  0.032  0.019 -0.558 -0.623
## Years       0.283 -0.262 -0.035  0.095  0.010 -0.033  0.012  0.138
## CAtBat      0.330 -0.193 -0.084  0.091 -0.012 -0.024 -0.012  0.147
## CHits       0.331 -0.183 -0.086  0.084 -0.009 -0.029 -0.020  0.195
## CHmRun      0.319 -0.126  0.086 -0.074 -0.033  0.041  0.229 -0.249
## CRuns       0.338 -0.172 -0.053  0.069  0.018 -0.007 -0.063  0.085
## CRBI        0.340 -0.168 -0.015  0.007 -0.028 -0.011  0.119 -0.002
## CWalks      0.317 -0.192 -0.042  0.030  0.034 -0.034 -0.178 -0.263
## LeagueN    -0.054 -0.095 -0.548 -0.396 -0.012  0.137  0.077 -0.026
## DivisionW  -0.026 -0.037  0.016  0.043 -0.986  0.091 -0.113  0.003
## PutOuts     0.078  0.156 -0.051 -0.288 -0.106 -0.924  0.019  0.065
## Assists    -0.001  0.169 -0.398  0.524  0.011 -0.035  0.013 -0.076
## Errors     -0.008  0.201 -0.383  0.422 -0.055 -0.148  0.373 -0.301
## NewLeagueN -0.042 -0.078 -0.545 -0.418 -0.014  0.157  0.023  0.068
```

You can verify that the sum of squared components for each row or each column always equals 1. These weights can be used to convert predictors into PCs and PCs back into predictors:

```
sum(pcr.fit$loadings[1,]^2) # Sum of squared loadings for 1st predictor

## [1] 1

sum(pcr.fit$loadings[,1]^2) # Sum of squared loadings for 1st PC

## [1] 1
```

If you take the predictor values for some observations and convert them into their PC equivalent values, these values are called **PC Scores**. I'm only listing the first 6 scores of the first 10 observations, for simplicity:

```
round(pcr.fit$scores[1:10, 1:6], digits = 4)
```

```
##                     Comp 1  Comp 2  Comp 3  Comp 4  Comp 5  Comp 6
## -Alan Ashby       -0.0096 -1.8670 -1.2627 -0.9337 -1.1075 -1.2097
## -Alvin Davis       0.4107  2.4248  0.9075 -0.2637 -1.2297 -1.8231
## -Andre Dawson      3.4602 -0.8244 -0.5544 -1.6136  0.8559  1.0268
## -Andres Galarraga -2.5534  0.2305 -0.5187 -2.1721  0.8187 -1.4889
## -Alfredo Griffin   1.0257  1.5705 -1.3288  3.4874 -0.9816 -0.5127
## -Al Newman        -3.9731 -1.5044  0.1552  0.3691  1.2070 -0.0334
## -Argenis Salazar  -3.4452 -0.5988  0.6253  1.9960 -0.8055 -0.2056
## -Andres Thomas    -3.4258 -0.1133 -1.9959  0.7664 -1.0142  0.2723
## -Andre Thornton    3.8923 -1.9442  1.8170 -0.0267  1.1350  0.8195
## -Alan Trammell     3.1688  2.3878 -0.7930  2.5641  0.9455  0.0611
```

## PCR Coefficients

Reconstructing coefficients for the actual predictors from the PCs is straightforward. The **$coefficients** attribute of the **pcr()** contains these values in a list with of 3 sets of elements: **n predictors**, **1 response** variable, **m PCs**.

To get the coefficients for the 1 PC model

```
pcr.fit$coefficients[,,1] # Or, alternatively:

##       AtBat        Hits       HmRun        Runs         RBI       Walks
## 21.13207878 20.87321071 21.77988064 21.13705999 25.06279956 22.26529508
##       Years      CAtBat       CHits      CHmRun       CRuns        CRBI
## 30.11445915 35.21789413 35.24760132 33.99408860 36.04328244 36.27081015
##      CWalks     LeagueN   DivisionW     PutOuts     Assists      Errors
## 33.76212997 -5.80503669 -2.74157997  8.28029613 -0.08969488 -0.83758395
##   NewLeagueN
## -4.46643991

# coef(pcr.fit, ncomp = 1)
```

To get the coefficients for the 3 PC model:

```
pcr.fit$coefficients[,,3] # Or alternatively

##      AtBat      Hits     HmRun      Runs       RBI     Walks     Years
##  31.596172 30.841116 21.650526 28.894882 30.091792 28.345853 25.276570
##     CAtBat     CHits    CHmRun     CRuns      CRBI    CWalks   LeagueN
##  33.076799 33.388213 29.160504 33.604363 32.997441 30.624769  5.466047
##  DivisionW   PutOuts   Assists    Errors NewLeagueN
##  -3.929845 12.899211 13.245133 12.826601   7.109718

# coef(pcr.fit, ncomp = 3)
```

Coefficients for the second variable (Hits) of the 3 PC model

```
pcr.fit$coefficients[2,,3]

## [1] 30.84112
```

Coefficients for, say 2 to 4 PCR component models:

```r
pcr.fit$coefficients[,, 2:4] # Or alternatively
```

```
##              2 comps   3 comps   4 comps
## AtBat      29.438966 31.596172 30.411575
## Hits       29.039128 30.841116 30.174755
## HmRun      26.912608 21.650526 30.389560
## Runs       29.312723 28.894882 30.745533
## RBI        31.870731 30.091792 35.242082
## Walks      27.235049 28.345853 33.185988
## Years      24.434849 25.276570 21.744656
## CAtBat     31.042550 33.076799 29.700428
## CHits      31.288812 33.388213 30.284689
## CHmRun     31.260422 29.160504 31.912973
## CRuns      32.314419 33.604363 31.040899
## CRBI       32.632509 32.997441 32.750143
## CWalks     29.599532 30.624769 29.499314
## LeagueN    -7.865898  5.466047 20.140856
## DivisionW  -3.535498 -3.929845 -5.513888
## PutOuts    11.651168 12.899211 23.556308
## Assists     3.560723 13.245133 -6.174373
## Errors      3.507803 12.826601 -2.808121
## NewLeagueN -6.145712  7.109718 22.588848
```

```r
# coef(pcr.fit, ncomp = 2:4) # Same result different format
```

Coefficients for a group of models, say 3, 6 and 18 PCs

```r
pcr.fit$coefficients[,, c(3, 6, 18)] # Or alternatively
```

```
##              3 comps    6 comps     18 comps
## AtBat      31.596172  24.363042 -287.1638712
## Hits       30.841116  25.321422  330.3182702
## HmRun      21.650526  16.517824   35.8569392
## Runs       28.894882  24.483536  -55.7545172
## RBI        30.091792  26.859813  -25.4323629
## Walks      28.345853  33.873700  133.8275233
## Years      25.276570  24.422920  -15.0311528
## CAtBat     33.076799  30.534064 -425.9232643
## CHits      33.388213  31.617866  151.1036646
## CHmRun     29.160504  27.460383   -0.3535161
## CRuns      33.604363  32.497526  452.9583268
## CRBI       32.997441  31.827587  239.5044763
## CWalks     30.624769  33.605665 -206.9835246
## LeagueN     5.466047  10.910631   31.7984349
## DivisionW  -3.929845 -68.868255  -58.5994581
## PutOuts    12.899211  74.954304   78.7188346
## Assists    13.245133  -3.328012   54.5749754
## Errors     12.826601   3.191508  -22.7108234
## NewLeagueN  7.109718  11.959825  -13.0025534
```

```r
# coef(pcr.fit, ncomp = c(3, 6, 18)) # Same result different format
```

Can you spot the most biased model and the most biased coefficient in that model? Look for any coefficient that changes substantially in sign and magnitude. The most unbiased model of the 3 is the model with 18 PCs, because it is the closest to the OLS. Notice that some coefficients change in value to some extent, but the change is not much (e.g., HmRun, LeagueN), but others change wildly and even change signs (e.g., AtBat, CWalks). Listing the coefficients for various PC models is useful because you can easily spot the most problematic predictors when it comes to bias and interpretation.

## Predictions with PCR

To do predictions, you can use any data frame with values to feed to the predict() function. The values need to be in the same format as the original predictors. For this illustration I use 5% of the existing data to test our predictions. For this illustration, we use a model with 18 components for the prediction.

```
set.seed(2)
test <- sample(1:nrow(Hitters), 0.05 * nrow(Hitters))
Hitters.test <- Hitters[test,] # Test sub-sample
pcr.pred <- predict(pcr.fit, Hitters.test, ncomp = 18)
pcr.pred
```

```
## , , 18 comps
##
##                      Salary
## -Rick Dempsey      356.2034
## -Willie Upshaw    1022.4237
## -Rick Leach        247.8984
## -Mookie Wilson     649.1189
## -Eric Davis        542.5348
## -John Shelby       197.9010
## -Dale Murphy      1041.7816
## -Kevin Bass        564.6475
## -Tony Bernazard    797.5811
## -Bill Schroeder    208.4064
## -Doug DeCinces     694.5173
## -Herm Winningham   211.7767
## -Jose Cruz         872.0351
```

We can then ompute the MSE for these predictions

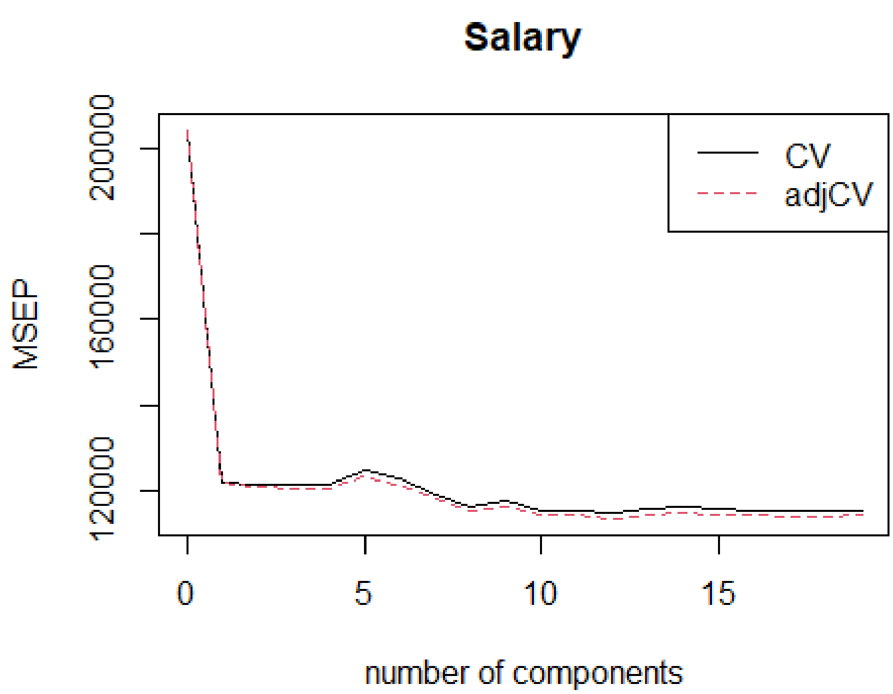
```
MSE <- round(
  mean( (pcr.pred - Hitters.test$Salary) ^ 2 ),
  digits = 2)

paste("MSE =", MSE)

## [1] "MSE = 73830.35"

RMSE <- round(
  sqrt( mean( (pcr.pred - Hitters.test$Salary) ^ 2 ) ),
```

```
  digits = 2)

paste("RMSE =", RMSE)

## [1] "RMSE = 271.72"
```

## Partial Least Squares (PLS) Regression

The process for fitting **PLSR** models is the same as for **PCR** models, except that we use **plsr()** function from the same **{pls}** library instead. The interpretation and analysis of the outputs is identical to **PCR**. The only thing that changes is how the PCs are constructed, but this is not visble in the **PLSR** output. In the next script lines, repeat the same steps above, but with the **plsr()** function.

```
library(pls)
library(ISLR) # Has the Hitters data set
set.seed(2) # Arbitrary
pls.fit <- plsr(Salary ~ ., data = Hitters, scale = T, validation = "CV")
validationplot(pls.fit, val.type = "MSEP", legendpos = "topright")
```




```
# Change to val.type="RMSEP" to use the RMSE instead
summary(pls.fit)

## Data:    X dimension: 263 19
##  Y dimension: 263 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             452    349.2    348.4    348.2    348.1    353.0    350.6
## adjCV          452    348.8    347.6    347.1    346.9    350.9    348.3
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       345.0    340.9    342.8     339.5     339.3     338.3     340.1
## adjCV    343.2    339.1    340.7     337.9     337.6     336.6     338.2
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        340.5       340     339.6     339.2     339.3     339.6
## adjCV     338.5       338     337.6     337.3     337.4     337.6
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         38.08    51.03    65.98    73.93    78.63    84.26    88.17    90.12
## Salary    43.05    46.40    47.72    48.71    50.53    51.66    52.34    53.26
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         92.92     95.00     96.68     97.68     98.22     98.55     98.98
## Salary    53.52     53.77     54.04     54.20     54.32     54.47     54.54
##         16 comps  17 comps  18 comps  19 comps
## X          99.24     99.71     99.99    100.00
## Salary     54.59     54.61     54.61     54.61
```

Notice in the output the similarities and differences between PLSR and PCR. With PLSR, the first marked elbow is also at 1 PC. But the next interesting elbow is at 8 PCs, not 6. But then, the CV MSE goes down slightly and remains relatively flat. These are the recommended models from this output:

1. If the analysis goal is predictive accuracy, I would select the model with the lowest CV RMSE or 17 PCs in this case, with `CV RMSE = 339.2`.

2. If the analysis goal is interpretability, I would select the largest possible model with an acceptable CV RMSE, which in this case is also 17 PCs. However, since the difference in CV RMSE between 17 PCs and the full 19 PCs is minuscule, it would be OK to interpret the results with the 19 PC model.

3. If the analysis goal is to substantially reduce the dimension of the model, I would select the model with the smallest number of PC's that explain at least 70% of the varianec of the predictors, or 4 PCs in this case, which explains 73.93% of the variance in the predictors, which is a good representation of the data.

## PLSR Loadings and Scores

### PLSR PC Loadings

```
round(pls.fit$loadings[,1:8], digits = 3)
```

```
##             Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
## AtBat        0.226  0.347 -0.396  0.099  0.109 -0.092 -0.199 -0.323
## Hits         0.223  0.356 -0.371  0.147  0.178 -0.016 -0.108 -0.167
```

```
## HmRun       0.218  0.091 -0.308 -0.014 -0.652  0.389  0.170  0.131
## Runs        0.225  0.337 -0.375  0.164 -0.079  0.080 -0.036 -0.077
## RBI         0.257  0.231 -0.343  0.043 -0.268  0.240  0.128 -0.008
## Walks       0.229  0.283 -0.168  0.029 -0.098  0.043 -0.050  0.317
## Years       0.266 -0.344  0.266 -0.166 -0.087 -0.240 -0.005  0.123
## CAtBat      0.320 -0.258  0.213 -0.121  0.199 -0.144  0.012  0.036
## CHits       0.321 -0.238  0.211 -0.103  0.243 -0.116  0.029  0.088
## CHmRun      0.311 -0.226  0.135 -0.025  0.020  0.252  0.068 -0.257
## CRuns       0.329 -0.234  0.203 -0.073  0.170 -0.089  0.034  0.015
## CRBI        0.331 -0.241  0.192 -0.074  0.179  0.066  0.051 -0.058
## CWalks      0.307 -0.233  0.230 -0.124 -0.042 -0.152 -0.041 -0.020
## LeagueN    -0.046  0.297  0.209 -0.860  0.421  0.102  0.033 -0.069
## DivisionW  -0.040 -0.376 -0.252 -0.293  0.491  0.164 -0.932  0.599
## PutOuts     0.100  0.467  0.095  0.236 -0.031 -0.044 -0.737  0.746
## Assists     0.010  0.131 -0.236 -0.012  0.916 -0.654  0.324 -0.244
## Errors      0.005  0.165 -0.289 -0.156  0.565 -0.626  0.522  0.305
## NewLeagueN -0.032  0.313  0.188 -0.890  0.299  0.058 -0.068 -0.076
```

**PLSR PC Scores**

```
round(pls.fit$scores[1:10, 1:6], digits = 4)

##                     Comp 1  Comp 2  Comp 3  Comp 4  Comp 5  Comp 6
## -Alan Ashby        -0.1090 -0.0879  1.1147 -1.4059 -0.6158 -1.2286
## -Alvin Davis        0.6671  0.8786 -1.0206  0.9639  0.0307  0.1497
## -Andre Dawson       3.4717  0.5270  1.2976 -0.3869  0.6279  2.0307
## -Andres Galarraga -2.1299  2.4542  2.0764  0.2078 -0.1079  0.5837
## -Alfredo Griffin    0.9771 -0.7937 -2.1395  0.4122  0.8415 -2.3038
## -Al Newman         -4.0037  0.1500  1.6439  0.7911  0.5309  0.5485
## -Argenis Salazar   -3.6685 -1.3440 -0.5804  0.9552  0.7630 -0.4132
## -Andres Thomas     -3.4262 -0.3027 -1.0077 -1.1485  0.7244 -0.4149
## -Andre Thornton     3.5184 -1.3746  1.0514  0.3408 -0.8714  0.0946
## -Alan Trammell      3.2932  0.1716 -1.7483  0.4770  0.1101 -1.6399
```

## PLSR Coefficients

To get the coefficients for the 1 PC model

```
pls.fit$coefficients[,,1] # Or, alternatively:

##        AtBat        Hits       HmRun        Runs         RBI       Walks
##   25.0420570  27.8270677  21.7597795  26.6334747  28.5110396  28.1564522
##        Years      CAtBat       CHits      CHmRun       CRuns        CRBI
##   25.4154350  33.3750764  34.8197471  33.2986538  35.6931216  35.9651267
##       CWalks     LeagueN   DivisionW     PutOuts     Assists      Errors
##   31.0715657  -0.9059591 -12.2120349  19.0607903   1.6135259  -0.3425902
##   NewLeagueN
##   -0.1798022

# coef(pls.fit, ncomp = 1)
```

To get the coefficients for the 3 PC model:

```
pls.fit$coefficients[,,3] # Or alternatively

##       AtBat        Hits       HmRun        Runs         RBI       Walks
##  11.5612560  43.0738184  -3.0788552  29.4670885  21.0426670  43.6363824
##       Years      CAtBat       CHits      CHmRun       CRuns        CRBI
##   5.4373897  28.9571188  41.9403523  35.6444168  42.6088887  43.5878120
##      CWalks     LeagueN   DivisionW     PutOuts     Assists      Errors
##  17.0901059  25.0699041 -69.4031815  74.5802997   0.8434621 -16.4113867
##  NewLeagueN
##  17.3645626

# coef(pls.fit, ncomp = 3)
```

Coefficients for the second variable (Hits) of the 3 PC model

```
pls.fit$coefficients[2,,3]

## [1] 43.07382
```

Coefficients for, say 2 to 4 PCR component models:

```
pls.fit$coefficients[,, 2:4] # Or alternatively

##               2 comps     3 comps    4 comps
## AtBat       26.988567  11.5612560  -6.652720
## Hits        43.689562  43.0738184  58.942213
## HmRun       12.561740  -3.0788552 -18.405673
## Runs        36.166065  29.4670885  31.456684
## RBI         30.845058  21.0426670  16.922246
## Walks       42.533221  43.6363824  47.896733
## Years        8.957764   5.4373897 -14.209470
## CAtBat      26.430999  28.9571188  25.046713
## CHits       33.829611  41.9403523  51.191402
## CHmRun      30.279265  35.6444168  44.690717
## CRuns       34.715277  42.6088887  51.881789
## CRBI        35.195944  43.5878120  55.348469
## CWalks      20.084235  17.0901059  -3.215364
## LeagueN     17.985777  25.0699041   9.757598
## DivisionW  -48.032748 -69.4031815 -81.335717
## PutOuts     56.282325  74.5802997  89.176182
## Assists      4.199492   0.8434621   8.658344
## Errors      -4.327875 -16.4113867 -26.720518
## NewLeagueN  15.096458  17.3645626  -8.808971

# coef(pls.fit, ncomp = 2:4) # Same result different format
```

Coefficients for a group of models, say 1, 8 and 17 PCs

```
pls.fit$coefficients[,, c(1, 8, 17)] # Or alternatively
```

```
##              1 comps    8 comps    17 comps
## AtBat       25.0420570 -268.57609 -295.439407
## Hits        27.8270677  208.90183  333.458425
## HmRun       21.7597795  -10.42637   31.880957
## Runs        26.6334747   28.55779  -51.938191
## RBI         28.5110396   26.94159  -19.261427
## Walks       28.1564522  131.95382  131.287369
## Years       25.4154350  -99.56742  -16.399659
## CAtBat      33.3750764  -23.34822 -409.837439
## CHits       34.8197471  157.73966  124.930500
## CHmRun      33.2986538   61.17685   -1.550428
## CRuns       35.6931216  160.81868  464.859131
## CRBI        35.9651267  136.84026  238.736301
## CWalks      31.0715657 -197.27214 -205.460321
## LeagueN     -0.9059591   42.70521   30.749788
## DivisionW  -12.2120349  -61.70447  -57.865552
## PutOuts     19.0607903   83.68301   79.361453
## Assists      1.6135259   39.12289   54.817784
## Errors      -0.3425902  -13.76418  -22.438269
## NewLeagueN  -0.1798022  -31.85963  -11.533475

# coef(pls.fit, ncomp = c(1, 8, 17)) # Same result different format
```

As with PCR, you can spot the most biased model and the most biased coefficient in that model. Interestingly, the models don't get as biased as with PCR when we reduce the number of components. Notice that some coefficients in the 8 PC model are different than the 17 PC model, but except for a few cases, not by much.

## Predictions with PLSR

Again, the process is the same as for PCR:

```
set.seed(2)
test <- sample(1:nrow(Hitters), 0.05 * nrow(Hitters))
Hitters.test <- Hitters[test,] # Test sub-sample
pls.pred <- predict(pls.fit, Hitters.test, ncomp = 17)
pls.pred

## , , 17 comps
##
##                     Salary
## -Rick Dempsey     349.9180
## -Willie Upshaw   1022.1080
## -Rick Leach       251.5487
## -Mookie Wilson    652.8297
## -Eric Davis       545.9409
## -John Shelby      198.3570
## -Dale Murphy     1038.4198
## -Kevin Bass       566.6128
## -Tony Bernazard   801.7325
```

```
## -Bill Schroeder   207.5167
## -Doug DeCinces    695.9014
## -Herm Winningham  209.7285
## -Jose Cruz        869.5758
```

We can then ompute the MSE for these predictions

```
MSE <- round(
  mean( (pls.pred - Hitters.test$Salary) ^ 2 ),
  digits = 2)

paste("MSE =", MSE)

## [1] "MSE = 74394.11"

RMSE <- round(
  sqrt( mean( (pls.pred - Hitters.test$Salary) ^ 2 ) ),
  digits = 2)

paste("RMSE =", RMSE)

## [1] "RMSE = 272.75"
```

## Logistic and GLM Regressions with PCR and PLSR

Running PCR and PLSR models with classification models like Logistic or even other GLM models, is similar to the quantitative methods we discussed above. The only problem is that the **{pls}** R package does not work with **GLM** models. But there are many libraries to estimate these models. In the example below I illustrate estimating a **Logistic PCR** model using the **pcr()** function from the **{compositional}** R package. As with Ridge and LASSO models, this **pcr()** function requires that we specify the model with the **X** predictor matrix and the **Y** outcome vector.

```
library(Compositional)

heart <- read.table(
  "http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data",
  sep = ",", head = T, row.names = 1)
```

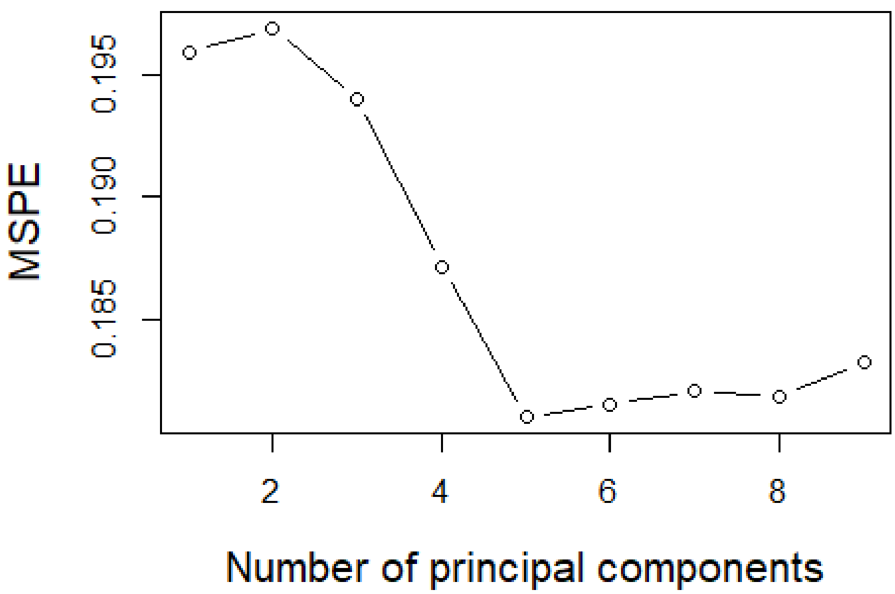
**Technical Note:** the outcome variable in the data set must be an integer. If the outcome is 0 or 1 as an integer, the function will fit a Logistic model. If it contains multiple integer values, it will fit a count data model.

Let's now compute the X matrix and Y vector:

```
y <- heart$chd
x <- model.matrix(chd ~ ., data = heart)[,-1]
```

The next step is to **tune** the model using the `pcr.tune(y, x, graph=T)` function. The default CV tuning method is **10FCV**, but you can use the `nfolds =` parameter to change the number of CV folds. The function renders a scree plot and also the mean deviance for every fold in each PC

```
heart.tune <- pcr.tune(y, x, graph=T)
```




```
round(heart.tune$msp, digits = 4)
```

```
##         [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
##  [1,] 0.2040 0.2021 0.2031 0.2018 0.1910 0.1953 0.1917 0.1934 0.1962
##  [2,] 0.1752 0.1748 0.1700 0.1645 0.1636 0.1663 0.1731 0.1743 0.1766
##  [3,] 0.2023 0.2007 0.2023 0.1727 0.1784 0.1997 0.2051 0.2076 0.2090
##  [4,] 0.2366 0.2431 0.2319 0.2183 0.2001 0.2001 0.2010 0.1997 0.2021
##  [5,] 0.1614 0.1609 0.1596 0.1461 0.1565 0.1495 0.1465 0.1464 0.1464
##  [6,] 0.2036 0.2086 0.2100 0.2050 0.1892 0.1898 0.1966 0.1956 0.1985
##  [7,] 0.2143 0.2154 0.1984 0.2102 0.2070 0.2072 0.2037 0.2009 0.2016
##  [8,] 0.1679 0.1685 0.1623 0.1488 0.1515 0.1449 0.1399 0.1396 0.1403
##  [9,] 0.2163 0.2269 0.2236 0.2226 0.2186 0.2048 0.2068 0.2056 0.2058
## [10,] 0.1769 0.1676 0.1787 0.1816 0.1544 0.1577 0.1564 0.1557 0.1560
```

To find the number of components that minimizes the CV deviance, we use the **$k** attribute:

```
best.comp <- heart.tune$k # Number of components that minimizes the deviance
best.comp # Check it out
```

```
## PC 5
##    5
```

You can now fit a PCR model with k components

```r
heart.pcr <- pcr(y, x, k = best.comp)
heart.pcr$be # Log-Odds coefficients
```

```
##                          PC5
## sbp             -0.002054484
## tobacco          0.103656894
## ldl              0.057891063
## adiposity        0.021546279
## famhistPresent   0.091868905
## typea            0.043635202
## obesity         -0.019546803
## alcohol         -0.002871749
## age              0.078381172
```

Let's display the results with log-odds and odds

```r
results <- round(cbind(heart.pcr$be, exp(heart.pcr$be)), digits = 4)
colnames(results) <- c("5 PC Log-Odds", "5 PC Odds")
results
```

```
##                 5 PC Log-Odds 5 PC Odds
## sbp                   -0.0021    0.9979
## tobacco                0.1037    1.1092
## ldl                    0.0579    1.0596
## adiposity              0.0215    1.0218
## famhistPresent         0.0919    1.0962
## typea                  0.0436    1.0446
## obesity               -0.0195    0.9806
## alcohol               -0.0029    0.9971
## age                    0.0784    1.0815
```