Post01-More on Principal Component Analaysis and PCA Regression

JackMoorer

10/27/2017

Introduction

This post will explore Principal Component Analysis, the mathematics behind it, and how we can expand on this idea of Principal Component Analysis to form a supervised learning technique, Principal Components Regression. In the post I will do an analysis in R of the built in mtcars data set in order to provide a visual aid to my post.

In class we briefly learned about Principal Component Analysis and its role in data visualization and analysis. The main idea of Principal Component Analysis (PCA) is to reduce the dimensionality of a data set that consists of many interrelated variables, while retaining as much information about the variation present in the data set. In order to do this, the variables are transformed to a new set of variables, the Principal Components (PCs). These Principal Components are created in such a way where they are uncorrelated and ordered so that the first few principal components contain most of the variation present in all of the original variables. In this way, we generally seek a number of PCs less than the number of variables used, allowing us to re-express the data in a lower dimensional space.

The mathematics behind PCA is based around creating a representation of data by projecting the data onto a lower dimensional space spanned by the Principal Components. If our data set has p variables and n individuals we can view the data set as consisting of n points in a p dimensional space. Using our Principal Components, we can project these n individuals in a lower dimensional space. For example, if I decided to use 2 PCs, we would be projecting our individuals on a plane that best represents the dispersion in our data. Each Principal Component is generated as a linear combination of our variables based on a related vector of loadings. These vectors of loadings are orthonormal right singular vectors of the n by p data matrix of predictors, and are eigenvectors of the sample correlation matrix. If you have not taken either Math 54 or Math 110, what I just wrote might not make any sense to you and because I know linear algebra is not a perquisite for this class I will try to explain PCA without going too much into detail on the linear algebra going on under the hood.

Using statistical learning methods we seek to find some kind of model, which in simple terms can be thought of as a function, that can be used to predict based on given inputs. In order to do that we use a training data set, which can be thought of as a matrix of predictors. For example, say I think the sales for my company are based on the amount we spend on television and radio advertising. I can look at the amount of sales based on the amount I spend in advertising, and create a model that predicts the amount of sales I can expect based on a certain television and radio advertising budget. In the field of statistical learning Principal Components analysis is considered an unsupervised learning method. The difference between supervised and unsupervised learning is that in supervised learning methods I use a data set, usually a vector, of responses in order to train my model. In the example I gave this response vector would be my vector of sales. In PCA we are using an unsupervised statistical learning method, so we do no base our analysis on a vector of responses. Unsupervised learning methods are often much more challenging, and unsupervised methods like PCA are often performed as a part of exploratory data analysis, rather that just being used for prediction. However, as I will discuss later, we can combine ideas from Principal Component Analysis with ideas from Linear Regression, a supervised learning technique, in order to create a supervised learning technique using PCA, called Principal Components Regression (PCR).

Principal Components

Setup

In order to understand PCA we first need to understand what Principal Components (PCs) are and how they are generated. First let's use the Motor Trend Car Road Tests, mtcars, data set to think of how we set up our data. I have printed the first 6 rows of mtcars. Each row of this data frame can be thought of as an individual, and each column as a variable. In mtcars we see 11 columns of different values that describe each individual car. Mathematically, we can view each car as a point in an 11-dimensional space, with each dimension representing a different variable

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

In addition, later I will need a response vector, y, and a matrix of predictors, X, so I will create that now by making y the miles per gallon of the car, and X the 32 by 10 matrix of remaining variables

```
#response vector
y <- mtcars$mpg
```

```
#predictors
X <- mtcars[,-1]
```

In PCA there is also the concept of active vs supplementary individuals, and active vs supplementary variables. The idea of active vs supplementary individuals or variables is that active individuals and active variables are used to generate the Principal Components. The supplementary individuals can be represented by projecting them on the Principal Components, and supplementary variables can be used to examine the correlation between the Principal Components and variables. I am going to use the first 20 rows as my active individuals, and use cyl, disp, hp, drat, wt, and qsex as my active variables.

```
#generate actives
active <- X[1:20, 1:6]
sup_inds <- X[21:32, 1:6]
```

```
#generate supplementaries
active_y <- y[1:20]
supp_y <- y[21:32]
```

Before we talk about what Principal Components are, I should mention that PCA, and later PCR, is not scale invariant, meaning that multiplying a certain column by a scalar will effect our analysis. Because of this it is always important to mean center and standardize our data. This can be done using the scale() function in R.

```
#scale our data
data <- scale(active)
resp <- scale(active_y)
```

Finding PCs

Now we can move on to what our principal components are and how to find them. Principal Components can be thought of in several different ways, but the core idea of them is that PCs are uncorrelated and capture the desperation or variability of the data. One way to view this is to imagine each Principal Component as a new axis, or line in space, that best represents the data. So the first PC is the line that best represents the dispersion of the active data, the first 2 PCs generate the best plane that represents the dispersion of the active data, and so on. In this way we see how PCs are used to reduce the dimensions of our data.

How do we generate these PCs though? We generate our Principal Components via a linear combination of the active individuals and an associated loading vector. A PC is the projection of the active individuals on the corresponding loading vector. But what are these loading vectors, and how do we find them so that they best describe the dispersion of data while generating uncorrelated PCs?

Finding Loading Vectors

In order to find the loading vectors, we turn to matrix decompositions from linear algebra. Matrix decompositions seek to represent a matrix into a product of often-simpler matrices. The two important matrix decompositions in relation to PCA are singular value decomposition and Eigen decomposition (or eigenvalue decomposition).

Eigen decomposition is one of the most famous matrix decompositions. In order to understand it, I first need to talk about eigenvalues and eigenvectors. For a given matrix M, x is an eigenvector if when you multiply x by M you get x times a constant, known as an eigenvalue. For example if Mx = ux, for any scalar (number) u, u is an eigenvalue and x is an eigenvector. In eigen decomposition, we decompose an n by n matrix into a product of a n by p matrix with columns of eigenvectors, V, and a p by p matrix with eigenvalues on the diagonal indices of the matrix and 0s elsewhere, A, (so a diagonal matrix). We get the equation M = VA(V)^T, where (V)^T is the trace of the matrix V. I have added this horrible word-text version of these equations only because on github the LaTeX will not appear, but download the html version of this post to see the below LaTeX equations describing this paragraph.

Defition of Eigenvalue and Eigenvector

\[If \ M*x = \lambda*x \ then \ x \ is \ an \ eigenvector \ of \ M \ and \ \lambda \ is \ the \ associated \ eigenvalue \] Eigen Decomposition \[M = V*A*V^T \ where \ V \ is \ a \ matrix \ of \ eigenvectors \ and \ A \ is \ diagonal \ matrix \ of \ eigenvalues \] Luckily, R has a function eigen() that computers Eigen Decomposition for us. However, note that our matrix must be a square matrix, meaning it has the same number of rows as columns. In order to perform PCA on R using Eigen Decomposition, we use eigen() on the Sample Correlation Matrix of X, R. Below is the eigenvalue decomposition from our example of active variables and individuals.

```
#R is the sample correlation matrix
n <- nrow(data)
R <- (1/(n-1))*t(data)%*%data
#Perform EVD
EVD <- eigen(R)
EVD
```

```
## eigen() decomposition
## $values
## [1] 4.61689812 0.90421803 0.27808900 0.09310695 0.06627787 0.04141003
##
## $vectors
##            [,1]        [,2]       [,3]       [,4]        [,5]       [,6]
## [1,] -0.4479240 -0.15440089  0.1657366  0.3623664 -0.51998979  0.5885210
## [2,] -0.4473560  0.12799900 -0.2564708 -0.2009273  0.66865560  0.4798328
## [3,] -0.4476782 -0.09417191 -0.2130723  0.6539422  0.20893901 -0.5234694
## [4,]  0.3891241 -0.36923504 -0.7864680  0.1739928 -0.08777439  0.2360892
## [5,] -0.4160682  0.33389733 -0.4805090 -0.4384809 -0.48028872 -0.2481295
## [6,]  0.2725631  0.83850243 -0.1092405  0.4220656 -0.02175332  0.1791004
```

We can see the variable EVD produces a list of eigenvalues and a matrix of eigenvectors.

```
#Eigenvalues
EVD$values
```

```
## [1] 4.61689812 0.90421803 0.27808900 0.09310695 0.06627787 0.04141003
```

```
#Eigenvectors
EVD$vectors
```

```
##            [,1]        [,2]       [,3]       [,4]        [,5]       [,6]
## [1,] -0.4479240 -0.15440089  0.1657366  0.3623664 -0.51998979  0.5885210
## [2,] -0.4473560  0.12799900 -0.2564708 -0.2009273  0.66865560  0.4798328
## [3,] -0.4476782 -0.09417191 -0.2130723  0.6539422  0.20893901 -0.5234694
## [4,]  0.3891241 -0.36923504 -0.7864680  0.1739928 -0.08777439  0.2360892
## [5,] -0.4160682  0.33389733 -0.4805090 -0.4384809 -0.48028872 -0.2481295
## [6,]  0.2725631  0.83850243 -0.1092405  0.4220656 -0.02175332  0.1791004
```

The matrix of diagonal eigenvalues can be generated by using the diag() function.

```
#Matrix of eigenvalues
A <- diag(EVD$values)
A
```

```
##          [,1]     [,2]     [,3]       [,4]       [,5]       [,6]
## [1,] 4.616898 0.000000 0.000000 0.00000000 0.00000000 0.00000000
## [2,] 0.000000 0.904218 0.000000 0.00000000 0.00000000 0.00000000
## [3,] 0.000000 0.000000 0.278089 0.00000000 0.00000000 0.00000000
## [4,] 0.000000 0.000000 0.000000 0.09310695 0.00000000 0.00000000
## [5,] 0.000000 0.000000 0.000000 0.00000000 0.06627787 0.00000000
## [6,] 0.000000 0.000000 0.000000 0.00000000 0.00000000 0.04141003
```

So what does this have to do with PCA? Well, the eigenvectors of R are the loading vectors associated to our PCs. Here we can see how to compute the first Principal Component.

```
#v1 is the first loading vector
v1 <- EVD$vectors[,1]
#z1 is the first PC
z1 <- data%*%v1
z1
```

```
##                           [,1]
## Mazda RX4             0.71769193
## Mazda RX4 Wag         0.71325496
## Datsun 710            2.00152175
## Hornet 4 Drive        0.08540947
## Hornet Sportabout    -1.73305949
## Valiant               0.05281502
## Duster 360           -2.46672340
## Merc 240D             1.88356419
## Merc 230              2.32860525
## Merc 280              0.60057963
## Merc 280C             0.70316800
## Merc 450SE           -1.71599242
## Merc 450SL           -1.54821481
## Merc 450SLC          -1.49946690
## Cadillac Fleetwood   -3.04509369
## Lincoln Continental  -3.12530614
## Chrysler Imperial    -3.04590100
## Fiat 128              2.65818506
## Honda Civic           3.42906194
## Toyota Corolla        3.00590063
```

We can also generate all Principal Components at once. Z is a matrix of Principal Components, where each column is a principal component.

```
#V is the matrix of loadings
V <- EVD$vectors
#Z is the matrix of PCs
Z <- data%*%V
Z
```

```
##                           [,1]        [,2]         [,3]        [,4]
## Mazda RX4            0.71769193 -1.53488926  0.212892988 -0.30510688
## Mazda RX4 Wag        0.71325496 -1.15993039  0.058814740 -0.26242170
## Datsun 710           2.00152175 -0.30952648  0.241064687 -0.15655121
## Hornet 4 Drive       0.08540947  0.85631653  0.691257189 -0.16917028
## Hornet Sportabout   -1.73305949 -0.57353646  0.417623154  0.09330899
## Valiant              0.05281502  1.52916745  1.057927339 -0.16622160
## Duster 360          -2.46672340 -1.30253460  0.106902649  0.50869729
## Merc 240D            1.88356419  0.88815131  0.008061844 -0.59656393
## Merc 230             2.32860525  2.19214760 -0.599873089  0.62795581
## Merc 280             0.60057963 -0.33451071 -0.374552181 -0.02140914
## Merc 280C            0.70316800 -0.01891190 -0.415668540  0.13744955
## Merc 450SE          -1.71599242 -0.21394667  0.367519845  0.09409670
## Merc 450SL          -1.54821481 -0.21594706  0.508085031  0.28782684
## Merc 450SLC         -1.49946690  0.01021684  0.457987266  0.37303009
## Cadillac Fleetwood  -3.04509369  0.71081692 -0.491008342 -0.31768315
## Lincoln Continental -3.12530614  0.60790657 -0.668594088 -0.28269656
## Chrysler Imperial   -3.04590100  0.17806303 -0.940696773 -0.08998062
## Fiat 128             2.65818506 -0.03284640  0.068906736 -0.05659017
## Honda Civic          3.42906194 -1.25648961 -0.734450542  0.04934615
## Toyota Corolla       3.00590063 -0.01971670  0.027800090  0.25268381
##                           [,5]         [,6]
## Mazda RX4           -0.08923933 -0.006765484
## Mazda RX4 Wag       -0.21253103 -0.003596751
## Datsun 710           0.30638780 -0.451778326
## Hornet 4 Drive       0.23686373  0.209061507
## Hornet Sportabout    0.30244937  0.417059136
## Valiant             -0.02356969  0.025596580
## Duster 360           0.49402400 -0.331583357
## Merc 240D            0.01069160 -0.152060150
## Merc 230             0.03765419 -0.030169360
## Merc 280            -0.40465045 -0.068903973
## Merc 280C           -0.41283804 -0.001493467
## Merc 450SE          -0.39547756 -0.078642171
## Merc 450SL          -0.24400682  0.023491650
## Merc 450SLC         -0.27214168  0.056716744
## Cadillac Fleetwood   0.18849803  0.163683966
## Lincoln Continental  0.07339263  0.002402273
## Chrysler Imperial    0.02721520 -0.132765291
## Fiat 128             0.06707015 -0.103918935
## Honda Civic          0.14825515  0.394510228
## Toyota Corolla       0.16195274  0.069155178
```

However, we saw one big issue with using eigen decomposition in that we had to use a square matrix of full column rank, generally a symetric matrix. Full column rank means that for a n by n matrix the rank of the matrix, or number of linearly indpendent columns, is equal to n. However, every matrix does have a singular value decomposition (SVD). For a given n by p matrix M , the SVD of M = UD(V)^T where for k being the rank of M, meaning the number of linear independent columns of M, U is the n by k matrix of “left singular vectors,” V is the p by k matrix of “right singular values” and D is a k by k diagonal matrix of singular values of M. Below is a LaTeX code of an SVD equation, to view download the html version of this post.

\[M = U*D*V^T \] Using our active individuals we can once again see the SVD of our data using the built in svd() function.

```
#SVD of our data
svd <- svd(data)
svd
```

```
## $d
## [1] 9.3659524 4.1448936 2.2986281 1.3300496 1.1221762 0.8870121
##
## $u
##               [,1]         [,2]         [,3]        [,4]         [,5]
##  [1,] -0.076627758 -0.370308486  0.092617414  0.22939510  0.079523451
##  [2,] -0.076154023 -0.279845639  0.025586888  0.19730219  0.189391850
##  [3,] -0.213701892 -0.074676580  0.104873289  0.11770328 -0.273030022
##  [4,] -0.009119145  0.206595541  0.300725984  0.12719095 -0.211075338
##  [5,]  0.185038255 -0.138371819  0.181683657 -0.07015452 -0.269520385
##  [6,] -0.005639044  0.368928038  0.460242938  0.12497398  0.021003556
##  [7,]  0.263371337 -0.314250434  0.046507154 -0.38246489 -0.440237445
##  [8,] -0.201107599  0.214276024  0.003507241  0.44852757 -0.009527562
##  [9,] -0.248624502  0.528879105 -0.260970053 -0.47212960 -0.033554613
## [10,] -0.064123712 -0.080704293 -0.162945970  0.01609649  0.360594380
## [11,] -0.075077042 -0.004562700 -0.180833318 -0.10334167  0.367890558
## [12,]  0.183216010 -0.051616928  0.159886608 -0.07074676  0.352420182
## [13,]  0.165302443 -0.052099543  0.221038382 -0.21640308  0.217440732
## [14,]  0.160097643  0.002464923  0.199243745 -0.28046328  0.242512426
## [15,]  0.325123764  0.171492200 -0.213609304  0.23885060 -0.167975420
## [16,]  0.333688023  0.146663975 -0.290866581  0.21254587 -0.065402057
## [17,]  0.325209959  0.042959616 -0.409242706  0.06765207 -0.024252164
## [18,] -0.283813642 -0.007924547  0.029977331  0.04254741 -0.059767930
## [19,] -0.366119941 -0.303141587 -0.319516911 -0.03710099 -0.132113964
## [20,] -0.320939132 -0.004756865  0.012094210 -0.18998073 -0.144320233
##               [,6]
##  [1,] -0.007627273
##  [2,] -0.004054906
##  [3,] -0.509325974
##  [4,]  0.235691819
##  [5,]  0.470184244
##  [6,]  0.028857080
##  [7,] -0.373820537
##  [8,] -0.171429613
##  [9,] -0.034012341
## [10,] -0.077680980
## [11,] -0.001683705
## [12,] -0.088659633
## [13,]  0.026484023
## [14,]  0.063941339
## [15,]  0.184534075
## [16,]  0.002708275
## [17,] -0.149676970
## [18,] -0.117156157
## [19,]  0.444763049
## [20,]  0.077964184
##
## $v
##            [,1]        [,2]       [,3]       [,4]        [,5]       [,6]
## [1,]  0.4479240 -0.15440089  0.1657366 -0.3623664  0.51998979  0.5885210
## [2,]  0.4473560  0.12799900 -0.2564708  0.2009273 -0.66865560  0.4798328
## [3,]  0.4476782 -0.09417191 -0.2130723 -0.6539422 -0.20893901 -0.5234694
## [4,] -0.3891241 -0.36923504 -0.7864680 -0.1739928  0.08777439  0.2360892
## [5,]  0.4160682  0.33389733 -0.4805090  0.4384809  0.48028872 -0.2481295
## [6,] -0.2725631  0.83850243 -0.1092405 -0.4220656  0.02175332  0.1791004
```

Notice that V for SVD is the same as V for Eigen Decomposition. This means that Principal Components can also be constructed via SVD.

```r
#v1 is the first loadings vector
v1 <- svd$v[,1]
#z1 is the first PC
z1 <- data%*%v1
z1
```

```
##                           [,1]
## Mazda RX4           -0.71769193
## Mazda RX4 Wag       -0.71325496
## Datsun 710          -2.00152175
## Hornet 4 Drive      -0.08540947
## Hornet Sportabout    1.73305949
## Valiant             -0.05281502
## Duster 360           2.46672340
## Merc 240D           -1.88356419
## Merc 230            -2.32860525
## Merc 280            -0.60057963
## Merc 280C           -0.70316800
## Merc 450SE           1.71599242
## Merc 450SL           1.54821481
## Merc 450SLC          1.49946690
## Cadillac Fleetwood   3.04509369
## Lincoln Continental  3.12530614
## Chrysler Imperial    3.04590100
## Fiat 128            -2.65818506
## Honda Civic         -3.42906194
## Toyota Corolla      -3.00590063
```

```
#V is matrix of loadings
V <- svd$v
#Z is matrix of PCs
Z <- data%*%V
Z
```

```
##                           [,1]        [,2]         [,3]        [,4]
## Mazda RX4           -0.71769193 -1.53488926  0.212892988  0.30510688
## Mazda RX4 Wag       -0.71325496 -1.15993039  0.058814740  0.26242170
## Datsun 710          -2.00152175 -0.30952648  0.241064687  0.15655121
## Hornet 4 Drive      -0.08540947  0.85631653  0.691257189  0.16917028
## Hornet Sportabout    1.73305949 -0.57353646  0.417623154 -0.09330899
## Valiant             -0.05281502  1.52916745  1.057927339  0.16622160
## Duster 360           2.46672340 -1.30253460  0.106902649 -0.50869729
## Merc 240D           -1.88356419  0.88815131  0.008061844  0.59656393
## Merc 230            -2.32860525  2.19214760 -0.599873089 -0.62795581
## Merc 280            -0.60057963 -0.33451071 -0.374552181  0.02140914
## Merc 280C           -0.70316800 -0.01891190 -0.415668540 -0.13744955
## Merc 450SE           1.71599242 -0.21394667  0.367519845 -0.09409670
## Merc 450SL           1.54821481 -0.21594706  0.508085031 -0.28782684
## Merc 450SLC          1.49946690  0.01021684  0.457987266 -0.37303009
## Cadillac Fleetwood   3.04509369  0.71081692 -0.491008342  0.31768315
## Lincoln Continental  3.12530614  0.60790657 -0.668594088  0.28269656
## Chrysler Imperial    3.04590100  0.17806303 -0.940696773  0.08998062
## Fiat 128            -2.65818506 -0.03284640  0.068906736  0.05659017
## Honda Civic         -3.42906194 -1.25648961 -0.734450542 -0.04934615
## Toyota Corolla      -3.00590063 -0.01971670  0.027800090 -0.25268381
##                           [,5]         [,6]
## Mazda RX4            0.08923933 -0.006765484
## Mazda RX4 Wag        0.21253103 -0.003596751
## Datsun 710          -0.30638780 -0.451778326
## Hornet 4 Drive      -0.23686373  0.209061507
## Hornet Sportabout   -0.30244937  0.417059136
## Valiant              0.02356969  0.025596580
## Duster 360          -0.49402400 -0.331583357
## Merc 240D           -0.01069160 -0.152060150
## Merc 230            -0.03765419 -0.030169360
## Merc 280             0.40465045 -0.068903973
## Merc 280C            0.41283804 -0.001493467
## Merc 450SE           0.39547756 -0.078642171
## Merc 450SL           0.24400682  0.023491650
## Merc 450SLC          0.27214168  0.056716744
## Cadillac Fleetwood  -0.18849803  0.163683966
## Lincoln Continental -0.07339263  0.002402273
## Chrysler Imperial   -0.02721520 -0.132765291
## Fiat 128            -0.06707015 -0.103918935
## Honda Civic         -0.14825515  0.394510228
## Toyota Corolla      -0.16195274  0.069155178
```

Notice that the results from the above r code are the same as when we found the Principal Components using Eigen Decomposition, except, for some changes in sign. This is fine, as a change in sign for a Principal Component does not change our analysis or what the PC represents.

Finding PCs using built in R functions.

While finding PCs by hand is great, we could have used R's built in functions prcomp() and princomp(). There is also a function PCA() in the library "FactoMineR" but I will not cover it.

```
#PCA with prcomp
pca1 <- prcomp(data)
```

The loadings can be found using rotation.

```
#Loadings
pca1$rotation
```

```
##             PC1         PC2        PC3        PC4         PC5        PC6
## cyl   0.4479240 -0.15440089  0.1657366 -0.3623664  0.51998979  0.5885210
## disp  0.4473560  0.12799900 -0.2564708  0.2009273 -0.66865560  0.4798328
## hp    0.4476782 -0.09417191 -0.2130723 -0.6539422 -0.20893901 -0.5234694
## drat -0.3891241 -0.36923504 -0.7864680 -0.1739928  0.08777439  0.2360892
## wt    0.4160682  0.33389733 -0.4805090  0.4384809  0.48028872 -0.2481295
## qsec -0.2725631  0.83850243 -0.1092405 -0.4220656  0.02175332  0.1791004
```

And the PCs using x

```
#PCs
pca1$x
```

```
##                           PC1         PC2          PC3         PC4
## Mazda RX4           -0.71769193 -1.53488926  0.212892988  0.30510688
## Mazda RX4 Wag       -0.71325496 -1.15993039  0.058814740  0.26242170
## Datsun 710          -2.00152175 -0.30952648  0.241064687  0.15655121
## Hornet 4 Drive      -0.08540947  0.85631653  0.691257189  0.16917028
## Hornet Sportabout    1.73305949 -0.57353646  0.417623154 -0.09330899
## Valiant             -0.05281502  1.52916745  1.057927339  0.16622160
## Duster 360           2.46672340 -1.30253460  0.106902649 -0.50869729
## Merc 240D           -1.88356419  0.88815131  0.008061844  0.59656393
## Merc 230            -2.32860525  2.19214760 -0.599873089 -0.62795581
## Merc 280            -0.60057963 -0.33451071 -0.374552181  0.02140914
## Merc 280C           -0.70316800 -0.01891190 -0.415668540 -0.13744955
## Merc 450SE           1.71599242 -0.21394667  0.367519845 -0.09409670
## Merc 450SL           1.54821481 -0.21594706  0.508085031 -0.28782684
## Merc 450SLC          1.49946690  0.01021684  0.457987266 -0.37303009
## Cadillac Fleetwood   3.04509369  0.71081692 -0.491008342  0.31768315
## Lincoln Continental  3.12530614  0.60790657 -0.668594088  0.28269656
## Chrysler Imperial    3.04590100  0.17806303 -0.940696773  0.08998062
## Fiat 128            -2.65818506 -0.03284640  0.068906736  0.05659017
## Honda Civic         -3.42906194 -1.25648961 -0.734450542 -0.04934615
## Toyota Corolla      -3.00590063 -0.01971670  0.027800090 -0.25268381
##                           PC5          PC6
## Mazda RX4            0.08923933 -0.006765484
## Mazda RX4 Wag        0.21253103 -0.003596751
## Datsun 710          -0.30638780 -0.451778326
## Hornet 4 Drive      -0.23686373  0.209061507
## Hornet Sportabout   -0.30244937  0.417059136
## Valiant              0.02356969  0.025596580
## Duster 360          -0.49402400 -0.331583357
## Merc 240D           -0.01069160 -0.152060150
## Merc 230            -0.03765419 -0.030169360
## Merc 280             0.40465045 -0.068903973
## Merc 280C            0.41283804 -0.001493467
## Merc 450SE           0.39547756 -0.078642171
## Merc 450SL           0.24400682  0.023491650
## Merc 450SLC          0.27214168  0.056716744
## Cadillac Fleetwood  -0.18849803  0.163683966
## Lincoln Continental -0.07339263  0.002402273
## Chrysler Imperial   -0.02721520 -0.132765291
## Fiat 128            -0.06707015 -0.103918935
## Honda Civic         -0.14825515  0.394510228
## Toyota Corolla      -0.16195274  0.069155178
```

Notice we get the same result as when we did SVD.

We could also instead use princomp():

```
#PCA weith princomp
pca2 <- princomp(data)
```

We can find the loadings using loadings:

```
#Loadings
unclass(pca2$loadings)
```

```
##          Comp.1      Comp.2     Comp.3     Comp.4      Comp.5     Comp.6
## cyl  -0.4479240 -0.15440089  0.1657366 -0.3623664 -0.51998979  0.5885210
## disp -0.4473560  0.12799900 -0.2564708  0.2009273  0.66865560  0.4798328
## hp   -0.4476782 -0.09417191 -0.2130723 -0.6539422  0.20893901 -0.5234694
## drat  0.3891241 -0.36923504 -0.7864680 -0.1739928 -0.08777439  0.2360892
## wt   -0.4160682  0.33389733 -0.4805090  0.4384809 -0.48028872 -0.2481295
## qsec  0.2725631  0.83850243 -0.1092405 -0.4220656 -0.02175332  0.1791004
```

And can see the PCs via scores:

```
#PCs
pca2$scores
```

```
##                          Comp.1      Comp.2       Comp.3      Comp.4
## Mazda RX4             0.71769193 -1.53488926  0.212892988  0.30510688
## Mazda RX4 Wag         0.71325496 -1.15993039  0.058814740  0.26242170
## Datsun 710            2.00152175 -0.30952648  0.241064687  0.15655121
## Hornet 4 Drive        0.08540947  0.85631653  0.691257189  0.16917028
## Hornet Sportabout    -1.73305949 -0.57353646  0.417623154 -0.09330899
## Valiant               0.05281502  1.52916745  1.057927339  0.16622160
## Duster 360           -2.46672340 -1.30253460  0.106902649 -0.50869729
## Merc 240D             1.88356419  0.88815131  0.008061844  0.59656393
## Merc 230              2.32860525  2.19214760 -0.599873089 -0.62795581
## Merc 280              0.60057963 -0.33451071 -0.374552181  0.02140914
## Merc 280C             0.70316800 -0.01891190 -0.415668540 -0.13744955
## Merc 450SE           -1.71599242 -0.21394667  0.367519845 -0.09409670
## Merc 450SL           -1.54821481 -0.21594706  0.508085031 -0.28782684
## Merc 450SLC          -1.49946690  0.01021684  0.457987266 -0.37303009
## Cadillac Fleetwood   -3.04509369  0.71081692 -0.491008342  0.31768315
## Lincoln Continental  -3.12530614  0.60790657 -0.668594088  0.28269656
## Chrysler Imperial    -3.04590100  0.17806303 -0.940696773  0.08998062
## Fiat 128              2.65818506 -0.03284640  0.068906736  0.05659017
## Honda Civic           3.42906194 -1.25648961 -0.734450542 -0.04934615
## Toyota Corolla        3.00590063 -0.01971670  0.027800090 -0.25268381
##                          Comp.5       Comp.6
## Mazda RX4            -0.08923933 -0.006765484
## Mazda RX4 Wag        -0.21253103 -0.003596751
## Datsun 710            0.30638780 -0.451778326
## Hornet 4 Drive        0.23686373  0.209061507
## Hornet Sportabout     0.30244937  0.417059136
## Valiant              -0.02356969  0.025596580
## Duster 360            0.49402400 -0.331583357
## Merc 240D             0.01069160 -0.152060150
## Merc 230              0.03765419 -0.030169360
## Merc 280             -0.40465045 -0.068903973
## Merc 280C            -0.41283804 -0.001493467
## Merc 450SE           -0.39547756 -0.078642171
## Merc 450SL           -0.24400682  0.023491650
## Merc 450SLC          -0.27214168  0.056716744
## Cadillac Fleetwood    0.18849803  0.163683966
## Lincoln Continental   0.07339263  0.002402273
## Chrysler Imperial     0.02721520 -0.132765291
## Fiat 128              0.06707015 -0.103918935
## Honda Civic           0.14825515  0.394510228
## Toyota Corolla        0.16195274  0.069155178
```

Notice this is the same result as our Eigen Decomposition

Picking the Number of PCs to use

Before I mentioned that PCA was a dimension reduction technique. Using the first k PCs I can generate the best k-dimensional representation of our data. But how do we pick this value, k? There are several ways to do this.

Kaiser's Rule

One way to pick how many PCs to use is by using Kaiser's rule. Remember, based on our Eigen decomposition, we saw that PCs are linked to eigenvectors, and hence eigenvalues. PCs are ordered by how large their eigenvalues are, so one can make the case that once eigenvalues become too small, the associated PC will no longer contribute enough information about the dispersion of the data to add a dimension. Kaiser's rule states that we only use PCs with associated eigenvalues greater than 1.

Here is a table of the eigenevalues associated with each PC, the proportion of variance described by each PC, and the cumulative proportion of variance described by all PCs up to that PC.

```
#eigenvalues
eigenvalues <- EVD$values
#proportion of variance explained
proportion <- EVD$values/sum(EVD$values)
#cumulative proportion
cum_prop <- rep(0, 6)
for (i in 1:6) {
  if (i == 1) {
    cum_prop[i] <- proportion[1]
  }
  else {
    cum_prop[i] <- cum_prop[i - 1] + proportion[i]
  }
}
#data frame
table <- data.frame(eigenvalues, proportion, cum_prop)
#labeling the PCs
rownames(table) <- c("Comp 1", "Comp 2", "Comp 3", "Comp 4", "Comp 5", "Comp 6")
table
```

```
##        eigenvalues  proportion  cum_prop
## Comp 1  4.61689812 0.769483020 0.7694830
## Comp 2  0.90421803 0.150703005 0.9201860
## Comp 3  0.27808900 0.046348167 0.9665342
## Comp 4  0.09310695 0.015517825 0.9820520
## Comp 5  0.06627787 0.011046312 0.9930983
## Comp 6  0.04141003 0.006901672 1.0000000
```

Based on Kaiser's rule we would only accept the first PC.

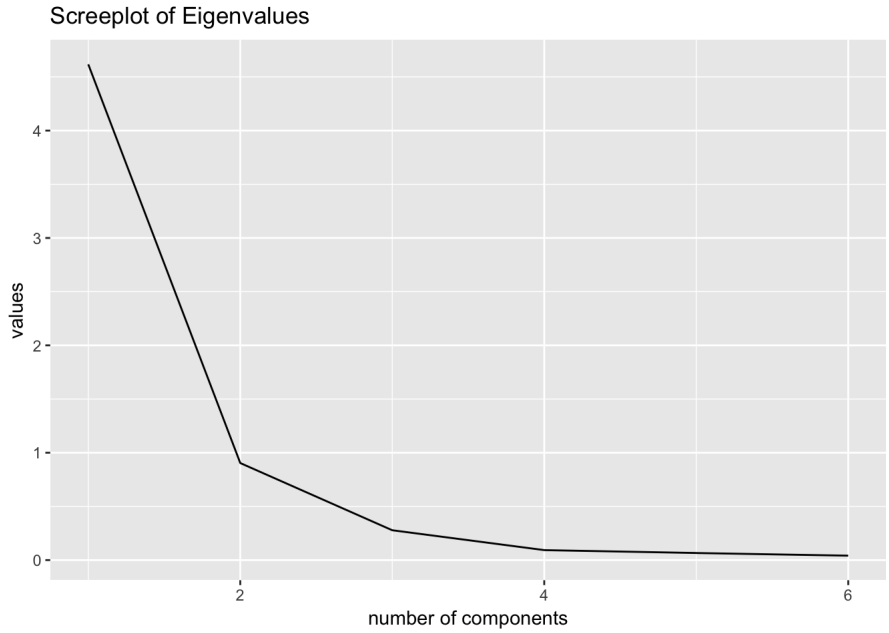
Jollife's Rule

Jollife's rule is an alternative to Kaiser's rule. We might think keeping only the PCs with eigenvalues greater than 1 is too strict. Jollife's rule instead determins the number of PCs by keeping any PC with eigenvalue great than 0.7. In our example above we would keep the first 2 PCs.

Screeplot or predetermined amount of variation

We can also pick the number of PCs based on a predetermined amount of variation, so cumulative proportion, associated with each PC. For a visual way to determine the number of PCs, we can look at a Scree Plot of the eigenvalues based on each component and look for an "elbow" shape to stop at that number of components.

```
library(ggplot2)
table$num_comps <- 1:6
ggplot(data = table, aes(x = num_comps, y = eigenvalues)) + geom_line() + ggtitle("Screeplot of Eigenvalues") + xl
ab("number of components") + ylab("values")
```



Based on the plot we should pick 2 or 3 components. These methods are much more objective than Kaiser's Jollife's Rule, but may be better in that you can examine what choice would be appropriate based on visuals.

PCA

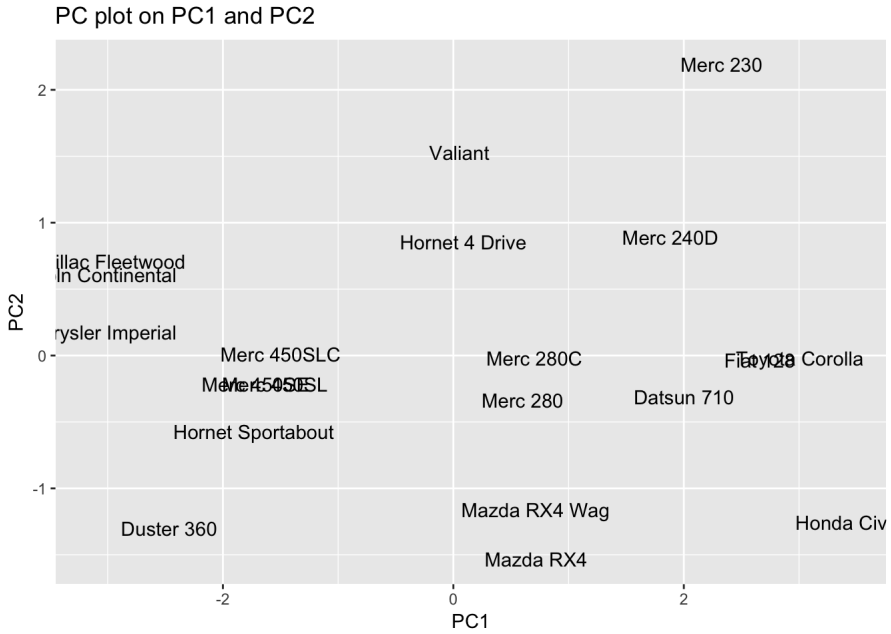
Visualizing Individuals

Now that we have generated our PCs we can do a bit of visual analysis. First, I will pick 2 PCs based on our previous section for this analysis.

In order to study the individuals we can look at a scatterplot of our individuals based on PC1 and PC2. This will give us a 2 dimensional view of the dispertion in our data.

```
#Generate a Data Frame of PCs
PCs <- data.frame(pca2$scores)

#Plot our 2D view of our active Individuals
plot <- ggplot(data = PCs, aes(x = Comp.1, y = Comp.2, label = rownames(PCs)))
plot + geom_text() + ggtitle("PC plot on PC1 and PC2") + xlab("PC1") + ylab("PC2")
```



Above we can see the 2D plot of our active individuals based on our first 2 PCs. This is the best projection of the individuals on a 2 dimensional space.

Remember before we split our data into active and supplementary individuals. We can now represent these individuals by projecting them on the PCs, representing our prediction of where the supplementary individuals would lie based on the PCs generated from the active individuals. First we need to standarize the data based on the mean and standard deviation of the active individuals, then project these onto the PCs.

```
#means of active individuals
active_mean <- colMeans(active)
#sd of actives
active_sd <- apply(active, 2, sd)
#mean center
sup_inds <- sweep(sup_inds, 2, active_mean)
#scale by sd
sup_inds <- sweep(sup_inds, 2, active_sd, FUN = "/")
```

Now we project the supplementary individuals with V, the vector of loadings.

```
V <- EVD$vectors
```

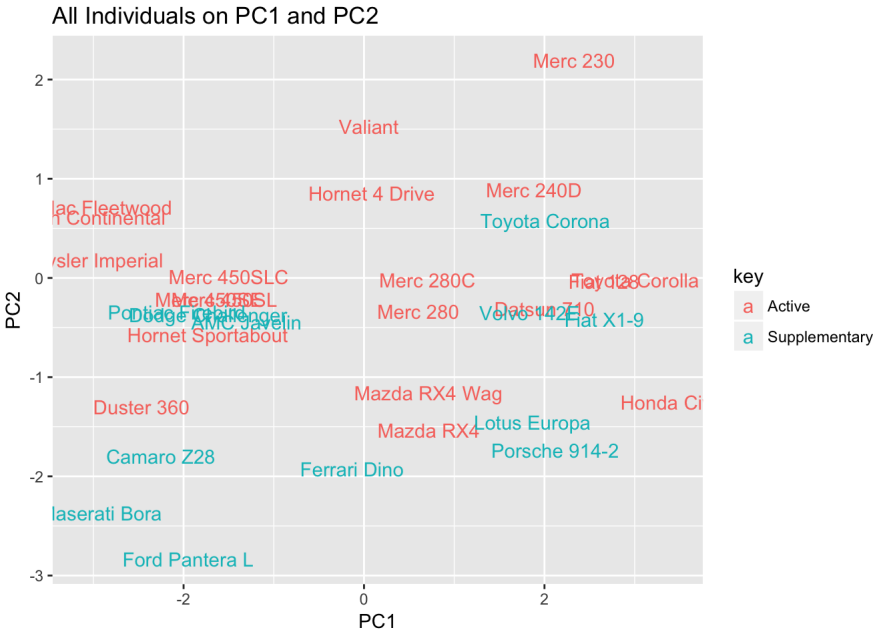
```
#project Supplementary Individuals
supPCs <- as.matrix(sup_inds)%*%V
colnames(supPCs) <- c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6")
colnames(PCs) <- c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6")
```

We can now vizualize these projections, along side of our active individuals.

```
#create a data frame of all PCs
all_PCs <- rbind(PCs, supPCs)
```

```
all_PCs <- data.frame(all_PCs)
#create a vector of keys for active or supp
key <- rep(c("Active", "Supplementary"), times <- c(nrow(PCs), nrow(supPCs)))
key <- factor(key)
#Add column for key factor
all_PCs$key <- key

#plot active and supp individuals on PC1 and PC2
plot <- ggplot(data = all_PCs, aes(x = PC1, y = PC2, color = key, label = rownames(all_PCs)))
plot + geom_text() + ggtitle("All Individuals on PC1 and PC2")
```



In the above plot we see the the active individuals in red and supplementary individuals in blue.

Visualizing Variables

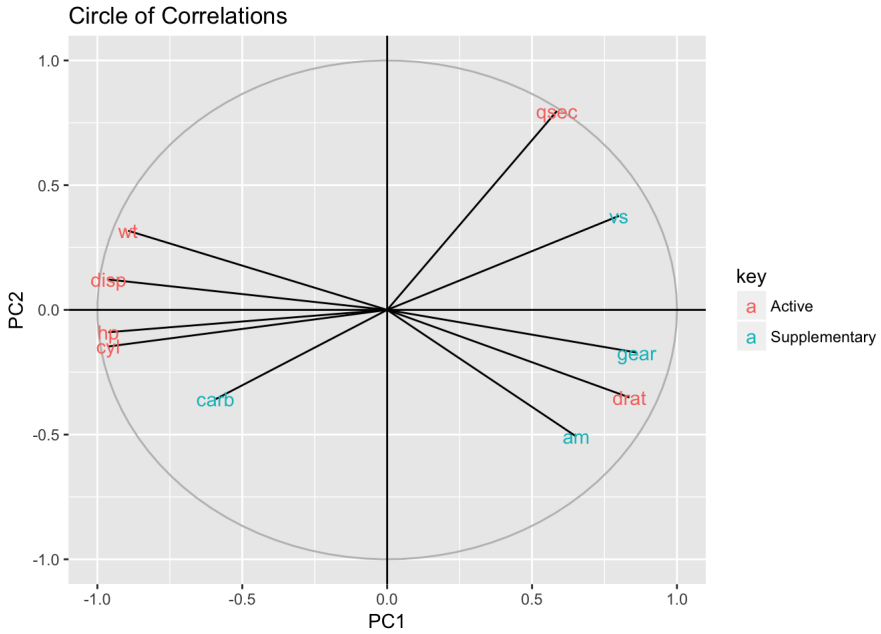
When we created our Principal Components we esentially created new uncorrelated variables using a linear combination of the original variables. What if we want to understand how our choosen PCs relate to our original variables. One way to do this is to look at the correlations between our original variables and PCs. Just like last section, we view both active and supplementary variables.

```
#examine all variables
ind_all_vars <- mtcars[1:20, -1]
#generate correlation_matrix
cor_mat <- cor(ind_all_vars, PCs)
```

```
#Plot Circle of Correlations
cor_mat <- data.frame(cor_mat)
colnames(cor_mat) <- c("PC1","PC2","PC3","PC4","PC5","PC6")
#create a vector of keys for active or supp
key <- rep(c("Active", "Supplementary"), times <- c(6, 4))
key <- factor(key)
#add column
cor_mat$key <- key

#Generate Plot
circle <- function(center = c(0,0), npoints = 100) {
  r = 1
  tt = seq(0, 2 * pi, length = npoints)
  xx = center[1] + r *cos(tt)
  yy = center[1] + r * sin(tt)
  return(data.frame(x = xx, y = yy))
}
corcir = circle(c(0,0), npoints = 100)
x1 <- rep(0, 10)
y1 <- rep(0, 10)
arrows = data.frame(x1, y1, x2 = cor_mat$PC1, y2 = cor_mat$PC2)

plot <- ggplot(cor_mat, aes(color= key))
plot <- plot + geom_path(data = corcir, aes(x = x, y = y), colour = "gray")
plot <- plot + geom_segment(data = arrows, aes(x = x1, y = y1, xend = x2, yend = y2), color = "Black")
plot <- plot+ geom_text(data = cor_mat, aes(x = PC1, y = PC2, label = rownames(cor_mat)))
plot <- plot + geom_hline(yintercept = 0, colour = "black")
plot <- plot + geom_vline(xintercept = 0, colour = "black")
plot <- plot + xlab("PC1") + ylab("PC2")
plot <- plot+ ggtitle("Circle of Correlations")
plot
```



Here we can see how correlated each variabe is with PC1 and PC2. In addition, we can see the active variables in red and supplementary variables in blue.

PCR

So far I have discussed PCA, an unsupervised learning method. However, the issue with unsupervised learning is that it is hard to understand what our analysis means, and it is hard to use it as a predictive tool. Recall I separated miles per gallon, mpg, as a vector of responses, and wondered whether our matrix of active variables can be used to predict mpg. So far it is hard to see how we can do this using PCA, but in reality we can use the PCs to regress our active variables on our response.

First I'll quickly go over multiple least squares regression. We have already seen simple linear regression in our HWs for class. Multiple linear regression is very similar in structure to simple linear regression. Say we have a vector of responses, Y, of length n, and an n by p matrix of predictors, X, where the ith column of X is Xi. If we assume there is a linear relationship, we can fit a model Y = B0 + B1X1 + B2X2 + … + … BpsXp + E, with a vector of residuals E. I will include the LaTeX for this below. Using matrix algebra beyond the scope of this course we minimize what is known as the residual sum of squares, which is the sum of squared differences in real value and predicted value of y, to estimate our coefficients such that Y = XB.

\[Y = B_0 + \sum_{i = 1}^{p}B_iX_i + \epsilon \] \[We \ want \ to \ estimate \hat{B_0}, \hat{B_1}...,\hat{B_p} \ so \ \hat{Y} = \hat{B_0} + \sum_{i = 1}^{p}\hat{B_i}X_i \]

One of the largest issues beyond the scope of this class in supervised learning methods is multicollinearity. In simple terms, this is an issue when our predictors are correlated, which causes the estimated coefficients to have high variance, and hense bad predictors. However, recall, that Principal Components are created so that each PC is uncorrelated with all other PCs. Instead of using our matrix of predictors, X, in our regression model, we can create a model using our PCs. Look at the LaTeX code below if you are interested.

\[\hat{Y} = \sum_{i = 1}^{p}\hat{B_i}Z_i \] I am not going into the math of how we estimate our coefficients, but it's ok, because R will do PCR for us with a function called PCR in the package "pls". However, what I will go over is how we pick the correct number of principal components to use in our regression model. One way to do this is through Cross Validation.

When we fit a regression model we are minimizing something called the residual sum of squares, or RSS. One of the most common ways to see how well a model is performing is to look at the Mean Squared Error, MSE, of a model, which is the RSS divided by the number of predicted values. However, if we look at just the MSE on the data we used to train the model, we won't get a very honest representation of how the model is doing, since we are essentially minimizing the MSE it to find the estimated coefficients. Instead, what we do is we split the data into a training set and test set. We fit the model on the training set, predict the values of the test set using that model, and then find the MSE using the predicted values in the test set and actual values in the test set. In order to improve our MSEs even more, we use something called cross validation, where we randomly split the data into subsets, for example 10 subsets. We then can use each subset as the test subset, re-training our model for each subset left out, and then find the average MSE based on the MSEs of each subset left out. In order to find the best number of components to use in our model we can do cross validation for each number of components, and then find our estimated coefficients based on the model with the best Cross Validation MSE. Luckily the pcr() function does this for us with the "validation" input.

I will now use cross validation to find the best number of components to use.

```
#load pls
library(pls)
```

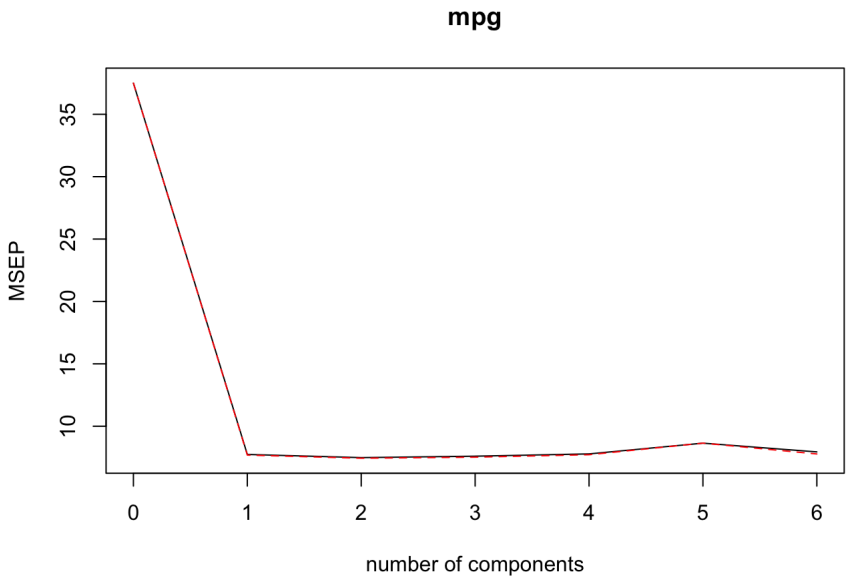
```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```r
#set a random set
set.seed(1000)
#Use the first 7 variables
data <- mtcars[,1:7]
#Fit our PCR model using MPG has the response using Cross Validation
pcr <- pcr(mpg ~ ., data = data, scale = TRUE, validation = "CV")
```

I can now fit a validation plot to look at the cross-validation MSEP for each number of components.

```r
validationplot(pcr, val.type = "MSEP")
```




```r
#find the best number of components based on our corss validation
number_of_components <- which.min(pcr$validation$PRESS)
number_of_components
```

```
## [1] 2
```

Based on our cross validation, we should use 2 PCs to fit our model.

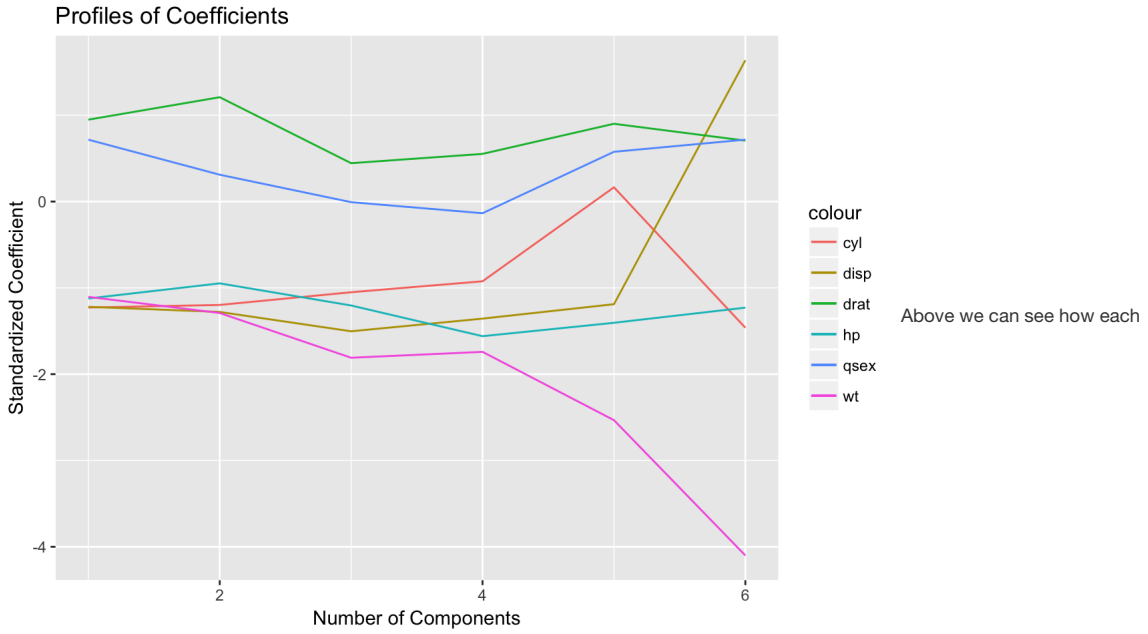
```r
#Estimated coefficients used the best number of components
pcr$coefficients[,,number_of_components]
```

```
##        cyl       disp         hp       drat         wt       qsec
## -1.1977118 -1.2789372 -0.9476601  1.2091189 -1.2911823  0.3110321
```

We can also examine how the estimated coefficients change based on the number of coefficients by looking at a paths of coefficients plot.

```r
#create a matrix of coefficients for different component values
comp_vs_coef <- rbind(pcr$coefficients[,,1], pcr$coefficients[,,2], pcr$coefficients[,,3], pcr$coefficients[,,4],
pcr$coefficients[,,5], pcr$coefficients[,,6])
#turn it into a data farme
comp_vs_coef <- data.frame(comp_vs_coef)
#add a column for number of components
comp_vs_coef$Number_Of_Comopents <- 1:6

#plot
p <- ggplot(data = comp_vs_coef, aes(x = Number_Of_Comopents))
p <- p + geom_line(aes(y = cyl, color = "cyl"))
p <- p + geom_line(aes(y = disp, color = "disp"))
p <- p + geom_line(aes(y = hp, color = "hp"))
p <- p + geom_line(aes(y = drat, color = "drat"))
p <- p + geom_line(aes(y = wt, color = "wt"))
p <- p + geom_line(aes(y = qsec, color = "qsex"))
p <- p + xlab("Number of Components")
p <- p + ylab("Standardized Coefficient")
p <- p + ggtitle("Profiles of Coefficients")
p
```



Above we can see how each estimated coefficient changes as the number of PCs changes.

Conclusion

Principal Components Analysis is a dimension reduction unsupervised statistical learning technique. When one first sees PCA, it can seem confusing what is going on. In this post I went over some of the linear algebra behind PCA, how we can use PCA visualize our individuals and variables, and finally, how we can extend PCA to a superversied learning technique in Principal Components Regression.

References

Abdi, Hervé. “The Eigen-Decomposition: Eigenvalues and Eigenvectors.” UTDallas.edu, University of Texas - Dallas, 2007, www.utdallas.edu/~herve/Abdi-EVD2007-pretty.pdf

Hastie, Trevor J., et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2017.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning: With Applications in R. New York: Springer, 2014. Print.

Jia, Yan-Bin. “Singular Value Decomposition.” Iowa State University Department of Computer Science, Iowa State University, 12 Sept. 2017, web.cs.iastate.edu/~cs577/handouts/svd.pdf.

Jollife, Ian T. Principal Component Analysis. Springer, 2002.

Motor Trend Car Road Tests." ETH Zurich Department of Mathematics. ETH Zurich. Web.

Sanchez, Gaston. “Data Analysis Visually Enforced | 5 Functions to Do Principal Components Analysis in R .” Gastonsanchez.com, 17 June 2012, www.gastonsanchez.com/visually-enforced/how-to/2012/06/17/PCA-in-R/.

Links to references

Abdi, Hervé

Hastie, Trevor J., et al

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani

Jia, Yan-Bin

Jollife, Ian T

Motor Trend Car Road Tests

Sanchez, Gaston