Principal Component Analysis

Question 9.1

In [1]:

```
# Read raw data
df <- read.table('uscrime.txt',
                 stringsAsFactors=FALSE,
                 header=TRUE)
final.test <-data.frame(M = 14.0,So = 0, Ed = 10.0, Po1 = 12.0, Po2 = 15.5,
                  LF = 0.640, M.F = 94.0, Pop = 150, NW = 1.1, U1 = 0.120,
                  U2 = 3.6, Wealth = 3200, Ineq = 20.1, Prob = 0.040,Time = 39.0)
dim(df)
```

47 16

Part 1: Introduction

PCA essentially creates new features by finding vector orthogonal to the original data set's variance. Basically, we can extract the useful information, the variance per predictor, and only use those to build a model. Since the new features are orthogonal, we do not have to worry abobut multicollinearity. This, coupled with having a "simpler" model, can help with overfitting.

It is important to note that the predictors used post-PCA are **new** features. This means that to look for any form of interpretation, we must reverse the transformation process.

Let us try out a simple example using the first 5 principal components for our data set and compare the results of using PCA transformations.

Without PCA

In [2]:

```
# Split train and test
sample <- sample(nrow(df), nrow(df)*0.75)
df.train <- df[sample,]
df.test <- df[-sample,]
```

In [3]:

```
# Predicting on all data
lm1 <- lm(Crime~.,
         data=df.train)

pred <- predict(lm1, df.test[,1:15])
ans <- df.test[,16]
sqrt(mean(pred - ans)^2)
```

77.6949876023082

In [4]:

```
lm2 <- lm(Crime ~  M + Ed + Po1 + U2 + Ineq,
          data = df.train)

pred2 <- predict(lm2, df.test[,c('M', 'Ed', 'Po1', 'U2', 'Ineq')])
ans2 <- df.test[,16]
sqrt(mean(pred2 - ans2)^2)
```

26.4953053318143

In [5]:

```
summary(lm1)
```

```
Call:
lm(formula = Crime ~ ., data = df.train)

Residuals:
    Min      1Q  Median      3Q     Max 
-410.78  -89.24  -11.56   98.04  415.16 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)   
(Intercept) -7.586e+03  2.501e+03  -3.033  0.00685 **
M            9.766e+01  5.385e+01   1.813  0.08558 . 
So           5.047e+01  1.826e+02   0.276  0.78518   
Ed           1.917e+02  7.702e+01   2.488  0.02228 * 
Po1          1.684e+02  1.338e+02   1.259  0.22337   
Po2         -7.208e+01  1.485e+02  -0.485  0.63295   
LF          -1.992e+03  1.832e+03  -1.087  0.29047   
M.F          3.980e+01  2.716e+01   1.466  0.15914   
Pop          1.242e+00  2.424e+00   0.512  0.61427   
NW          -1.844e+00  9.191e+00  -0.201  0.84311   
U1          -7.124e+03  5.624e+03  -1.267  0.22053   
U2           1.467e+02  1.107e+02   1.325  0.20082   
Wealth       7.153e-02  1.242e-01   0.576  0.57138   
Ineq         7.386e+01  2.887e+01   2.558  0.01921 * 
Prob        -3.907e+03  2.684e+03  -1.456  0.16185   
Time        -5.312e-02  8.759e+00  -0.006  0.99522   
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 226.6 on 19 degrees of freedom
Multiple R-squared:  0.793,	Adjusted R-squared:  0.6295 
F-statistic: 4.852 on 15 and 19 DF,  p-value: 0.0008221
```

In [6]:

```
summary(lm2)
```

```
Call:
lm(formula = Crime ~ M + Ed + Po1 + U2 + Ineq, data = df.train)

Residuals:
    Min      1Q  Median      3Q     Max 
-463.82 -102.11   -8.17   98.72  552.73 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -4906.19    1239.62  -3.958 0.000449 ***
M              87.13      39.76   2.191 0.036620 *  
Ed            172.61      57.98   2.977 0.005822 ** 
Po1           139.88      18.39   7.607 2.19e-08 ***
U2             79.59      50.75   1.568 0.127675    
Ineq           68.45      18.30   3.740 0.000807 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 216.6 on 29 degrees of freedom
Multiple R-squared:  0.7112,	Adjusted R-squared:  0.6614 
F-statistic: 14.28 on 5 and 29 DF,  p-value: 4.42e-07
```

In all fairness, I should note that the lm2 model gives worse testing results than lm1 in some iterations. Here, a cross-validation approach could have been more useful as shown in Solutions for Homeowrk 5. However, the point remains that the Adjusted R-Squared for lm2 is much better than for lm1.

Let us move on to using PCA.

With PCA

In [7]:

```
# Apply PCA to training data
pca <- prcomp(df.train[,1:15], scale.=TRUE)
```

In [8]:

```
# Split train and test
# Keep only 5 principal components
train3 <- data.frame(cbind(pca$x[,1:5], df.train[,16]))
test3 <- data.frame(predict(pca, df.test[,1:15]))[,1:5]
```

In [9]:

```
# Build model
lm3 <- lm(V6~.,
         data=train3)
```

In [10]:

```
pred3 <- predict(lm3, test3)
```

In [11]:

```
sqrt(mean(pred3 - df.test[,16])^2)
```

28.962570355243

In [12]:

```
summary(lm3)
```

```
Call:
lm(formula = V6 ~ ., data = train3)

Residuals:
    Min      1Q  Median      3Q     Max 
-434.29 -189.83   32.25  179.02  421.15 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   865.89      42.33  20.457  < 2e-16 ***
PC1            64.51      18.05   3.574  0.00125 ** 
PC2           -55.25      25.77  -2.144  0.04054 *  
PC3           -62.25      29.99  -2.076  0.04688 *  
PC4           -65.35      36.38  -1.797  0.08282 .  
PC5          -199.22      43.23  -4.608 7.52e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 250.4 on 29 degrees of freedom
Multiple R-squared:  0.6141,	Adjusted R-squared:  0.5475 
F-statistic: 9.229 on 5 and 29 DF,  p-value: 2.425e-05
```

Here, we see the affects of PCA first hand. We not only get a good testing score, but we have an ever lower R^2. While having a super low R^2 value is not what we are really looking for, we can see how the model saved from some overfitting whiel retaining as much information as possible for only have 5 "predictors".

It is important to note that refreshing and running the code leads to vastly different results with each iteration. Sometimes, the "BEST" score is from the overfit model - this is probably because there is not a lot of data at all. We will see an example later on why cross-validating would be a much better option, and how a single test case cannot be a good validation for such small data sets.

In [13]:

```
predict(lm1, final.test)
```

1: 480.078597718236

In [14]:

```
predict(lm2, final.test)
```

1: 1380.52758066187

In [15]:

```
predict(lm3, data.frame(predict(pca, final.test)))
```

1: 1282.52390742588

We can see how the PCA method also seems to help improve model accuracy in the way that looking for significant features does.

Part 2: Model Building with PCA

Now that we see how PCA works, let us build a model using all of the data since the models before were made without a lot in the first place. Also, we can use cross-validation for the same reason.

We will try two things here: PCA using all of the components, and PCA using only the first 5 as shown above.

In [16]:

```
if (!require("DAAG")) install.packages("DAAG")
library(DAAG)
```

```
Loading required package: DAAG
Loading required package: lattice
```

In [17]:

```
# PCA Transform on full data
df.x <- df[,1:15]
df.y <- df[,16]

# Use first 5 components, like before
pca2 <- prcomp(df.x, scale.=TRUE)
pca.x <- pca2$x
df.pca <- data.frame(cbind(pca.x, df.y))

df.pca.train <- df.pca[,c(1:5, 16)]
```

All 15 Principal Components

In [18]:

```
# Build linear model
final.lm.example <- lm(df.y~.,
              df.pca)
```

In [19]:

```
# Arbitrarily chose 4 since other models also split data into 4 slices (75% train | 25% test)
cv.example <- cv.lm(df.pca, final.lm.example, m=4,
           plotit=FALSE)
```

```
Analysis of Variance Table

Response: df.y
          Df  Sum Sq Mean Sq F value  Pr(>F)
PC1        1 1177568 1177568   26.94 1.2e-05 ***
PC2        1  633037  633037   14.48 0.00062 ***
PC3        1   58541   58541    1.34 0.25599
PC4        1  257832  257832    5.90 0.02114 *
```

```
PC5         1 2312556 2312556   52.91 3.5e-08 ***
PC6         1   92261   92261    2.11 0.15631
PC7         1  203535  203535    4.66 0.03879 *
PC8         1   11661   11661    0.27 0.60916
PC9         1   14950   14950    0.34 0.56289
PC10        1   29162   29162    0.67 0.42026
PC11        1    7564    7564    0.17 0.68027
PC12        1  494595  494595   11.32 0.00206 **
PC13        1   21336   21336    0.49 0.48996
PC14        1  129212  129212    2.96 0.09552 .
PC15        1   82173   82173    1.88 0.18017
Residuals 31 1354946   43708
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


fold 1
Observations in test set: 11
               2    6  12  18  24    25   26    27     28    32    39
Predicted   1474  793 722 844 869 605.9 1977 279.5 1258.5 807.8 839.3
cvpred      1402  786 745 699 818 605.5 1744 331.9 1200.2 818.5 867.6
df.y        1635  682 849 929 968 523.0 1993 342.0 1216.0 754.0 826.0
CV residual  233 -104 104 230 150 -82.5  249  10.1   15.8 -64.5 -41.6

Sum of squares = 226656    Mean square = 20605    n = 11

fold 2
Observations in test set: 12
                1   9    10   11    17   22   23   29   35   40  42   45
Predicted   755.0 689 736.5 1161 393.4  657  958 1287  738 1131 326  617
cvpred      764.7 674 796.2  980 505.2  814  655 1411  920 1310 293  738
df.y        791.0 856 705.0 1674 539.0  439 1216 1043  653 1151 542  455
CV residual  26.3 182 -91.2  694  33.8 -375  561 -368 -267 -159 249 -283

Sum of squares = 1354557    Mean square = 112880    n = 12

fold 3
Observations in test set: 12
               5   7   14   15   20     21   33   37    38   44   46   47
Predicted   1167 934  780  903 1228 774.85  841  971 562.7 1121  827  992
cvpred       991 798  855 1172 1474 736.35  862 1511 642.9 1166  936 1317
df.y        1234 963  664  798 1225 742.00 1072  831 566.0 1030  508  849
CV residual  243 165 -191 -374 -249   5.65  210 -680 -76.9 -136 -428 -468

Sum of squares = 1256692    Mean square = 104724    n = 12

fold 4
Observations in test set: 12
              3    4    8   13     16   19    30  31   34     36  41   43
Predicted   322 1791 1362  733 1005.7 1146 702.7 388  971 1137.6 824 1134
cvpred      159 1844 1238  762  932.7 1245 733.7 257 1026 1223.9 916 1246
df.y        578 1969 1555  511  946.0  750 696.0 373  923 1272.0 880  823
CV residual 419  125  317 -251   13.3 -495 -37.7 116 -103   48.1 -36 -423

Sum of squares = 808153    Mean square = 67346    n = 12

Overall (Sum over all 12 folds)
   ms
77576
```

Notice that although the overall error is 77576. Not the best, but also not too bad for the amount of data we have for our model. What is weird is how the mean square for EACH of the folds is so vastly different - the range is multiple times the minimum!

We can move on to building it as we had before, with 5 principal components.

In [20]:

```
# Build linear model
final.lm <- lm(df.y~.,
              df.pca.train)

# Arbitrarily chose 4 since other models also split data into 4 slices (75% train | 25% test)
cv <- cv.lm(df.pca, final.lm, m=4,
           plotit=FALSE)
```

```
Analysis of Variance Table

Response: df.y
          Df  Sum Sq Mean Sq F value  Pr(>F)    
PC1        1 1177568 1177568   19.78 6.5e-05 ***
PC2        1  633037  633037   10.63  0.0022 ** 
PC3        1   58541   58541    0.98  0.3272    
PC4        1  257832  257832    4.33  0.0437 *  
PC5        1 2312556 2312556   38.84 2.0e-07 ***
Residuals 41 2441394   59546                    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


fold 1 
Observations in test set: 11 
                2    6     12   18  24   25   26   27   28   32  39
Predicted    1196  901 831.74 1098 929  604 1846  480 1015  970 628
cvpred       1088 1025 849.78 1178 793  657 1612  614  908  982 671
df.y         1635  682 849.00  929 968  523 1993  342 1216  754 826
CV residual   547 -343  -0.78 -249 175 -134  381 -272  308 -228 155

Sum of squares = 916770    Mean square = 83343    n = 11 

fold 2 
Observations in test set: 12 
               1     9   10   11  17   22   23   29   35     40  42    45
Predicted    714 862.7  906 1310 468  770  768 1464  915 1069.9 272 425.5
cvpred       731 885.4  848 1402 390  810  769 1588  988 1117.2 159 432.1
df.y         791 856.0  705 1674 539  439 1216 1043  653 1151.0 542 455.0
CV residual   60 -29.4 -143  272 149 -371  447 -545 -335   33.8 383  22.9

Sum of squares = 1016775    Mean square = 84731    n = 12 

fold 3 
Observations in test set: 12 
                5   7    14  15     20     21   33   37    38   44   46   47
Predicted    1004 818 653.8 663 1238.8 805.8  723 1212 604.3 1126  927 1139
cvpred       1038 795 665.3 669 1263.5 841.7  734 1334 649.6 1185  933 1224
df.y         1234 963 664.0 798 1225.0 742.0 1072  831 566.0 1030  508  849
CV residual   196 168  -1.3 129  -38.5 -99.7  338 -503 -83.6 -155 -425 -375

Sum of squares = 814840    Mean square = 67903    n = 12 

fold 4 
Observations in test set: 12 
                 3    4    8   13    16   19   30   31    34   36    41   43
Predicted    506.4 1745 1158  669 933.8  975  802  688 841.7  978 841.5 1043
cvpred       541.4 1636 1101  694 913.1  917  798  752 839.6  954 861.8 1021
df.y         578.0 1969 1555  511 946.0  750  696  373 923.0 1272 880.0  823
CV residual   36.6  333  454 -183  32.9 -167 -102 -379  83.4  318  18.2 -198

Sum of squares = 682326    Mean square = 56860    n = 12 

Overall (Sum over all 12 folds) 
   ms 
72994 
```

Notice that while the overall ms is very similar, the folds are all much closer to one another. I believe that this shows robustness in our model, compared to the model before. Since all of the folds performed similarly, I think it is fair to say that the model is **consistent** - not the mathematical definition, but in actual English. We can assume that the model will perform consistently for other data sets.

Part 3: Results

In [21]:

```
SStot <- sum((df.y - mean(df.y))^2)
SSres_model1 <- sum(final.lm.example$residuals^2)
SSres_model2 <- sum(final.lm$residuals^2)

SSres_c <- 72994*length(df.y)

1-SSres_model1/SStot
```

```
1-SSres_model2/SStot
1-SSres_c/SStot
```

0.803086758316909

0.645194053656692

0.501416354054191

Using the R^2 calculation method from Hw5 Solutions, the results are not any different than what we saw then.
Basically, we see how using a bunch of components overfits - this applies even when using PCA rather than just the original variables. However, I do want to point out that the R^Squared for the PCA with 15 components is slightly lower than when using 15 variables, in the Hw 5 Solutions.

What we really need to note here is looking at these R^2 values but also in combination with the testing scores. What does it say about a model if it has a great R^2 but a horrible testing score? It is overfitting.

Let us look at the actual predictions made with our models.

In [22]:

```
pred5 <- predict(final.lm, data.frame(predict(pca2, final.test)))
pred15 <- predict(final.lm.example, data.frame(predict(pca2, final.test)))
```

In [23]:

```
results <- data.frame( c(pred5), c(pred15))
colnames(results) = c('5 Comps', '15 Comps')
results
```

5 Comps	15 Comps
1389	155

The explanation from before is justified. The model using 15 principal components does horribly. This is significant because even though we got rid of multicollinearity, we are still using too many features for such a small data set. They are just adding noise or random error to the model, and our model is forced to fit to them.

Part 4: Reverting Model to Original Variables

Now, we are just getting the post-PCA model and translating back into the language of Crime Data.

Principal Component so far has just applied a linear transformation to our data set. Basically, each principal component is a linear combination of the original crime predictors. (y = a1x1 + a2x2 ... means y is a linear combo of x, as long as a1 a2 etc are scalars).

Since the model is a linear combination of the principal components (hence, the name linear model) we can substitute in values (basically just multiply) and then add them all up to get the model back into the language of our original data. First, however, we need to unscale the Principal Component Analysis data.

In [24]:

```
final.lm
```

```
Call:
lm(formula = df.y ~ ., data = df.pca.train)

Coefficients:
(Intercept)          PC1          PC2          PC3          PC4          PC5
      905.1         65.2        -70.1         25.2         69.4       -229.0
```

In [25]:

```
# PCA transformation values
pca2$rotation[,1:5]
```

	PC1	PC2	PC3	PC4	PC5
M	-0.3037	0.06280	0.172420	-0.0204	-0.3583
So	-0.3309	-0.15837	0.015543	0.2925	-0.1206
Ed	0.3396	0.21461	0.067740	0.0797	-0.0244
Po1	0.3086	-0.26982	0.050646	0.3333	-0.2353
Po2	0.3110	-0.26396	0.053065	0.3519	-0.2047
LF	0.1762	0.31943	0.271530	-0.1433	-0.3941
M.F	0.1164	0.39434	-0.203162	0.0105	-0.5788
Pop	0.1131	-0.46723	0.077021	-0.0321	-0.0832
NW	-0.2936	-0.22801	0.078816	0.2393	-0.3608
U1	0.0405	0.00807	-0.659029	-0.1828	-0.1314
U2	0.0181	-0.27971	-0.578501	-0.0689	-0.1350
Wealth	0.3797	-0.07719	0.010065	0.1178	0.0117
Ineq	-0.3658	-0.02752	-0.000294	-0.0807	-0.2167
Prob	-0.2589	0.15832	-0.117673	0.4930	0.1656
Time	-0.0206	-0.38015	0.223566	-0.5406	-0.1476

In [26]:

```
# Get linear model coef without intercept
as.matrix(final.lm$coef)[-1]
```

65.215930138666 -70.0831185497856 25.1940780425773 69.4460307968387 -229.042822001687

In [27]:

```
X <- pca2$rotation[,1:5]
Y <- final.lm$coef[-1]
```

In [28]:

```
X %*% Y
```

M	60.79
So	37.85
Ed	19.95
Po1	117.34
Po2	111.45
LF	76.25
M.F	108.13
Pop	58.88
NW	98.07
U1	2.87
U2	32.35
Wealth	35.93
Ineq	22.10
Prob	-34.64
Time	27.21

With an intercept of 905.1, the values in the dataframe above show the coefficients for our final linear model using the 5 Principal Components.