PSTAT 134: Homework 3

TJ Sipin

2022-07-20

Problem 1 (BigMac data)

The data bicmac.txt includes economic data on 45 world cities from the period 1900-1991. The Economist magazine has published a Big Mac party index, which compares the costs of a Big Mac in various places, as a measure of inefficiency in currency exchange. The data contains following variables:

- `BicMac` (Y): Minutes of labor required by an average worker to buy a Bic Mac and French fries
- `Bread` (X_1): Minutes of labor required to buy one kilogram of bread
- `BusFare` (X_2): The lowest cost of a ten-kilometer bus, train, or subway ticket, in U.S. dollars
- `EngSal` (X_3): The average annual salary of an electrical engineer, in thousands of U.S. dollars
- `EngTax` (X_4): The average Tax rate paid by engineers
- `Service` (X_5): Annual cost of 19 services, primarily relevant to Europe and North America
- `TeachSal` (X_6): The average annual salary of a primary school teacher, in thousands of U.S. dollars
- `TeachTax` (X_7): The average tax rate paid by primary teachers
- `VacDays` (X_8): Average days of vacation per year
- `WorkHrs` (X_9): Average hours worked per year
- `City` (X_10): Name of city We want to study how the cost of a Big Mac varies with economic indicators that describe each city.

a. First we apply a logarithmic transformation to every variable except X_9 and X_10 to make a linear relationship between the response and predictors.

```
mac <- read.table(file = 'bigmac.txt', header = T)
mac[1:9] <- log(mac[1:9])
mac$WorkHrs <- as.numeric(mac$WorkHrs)
```

b. Now we run a multiple linear regression model on our response on the other predictors (except `City`).

```
lm.mac <- lm(BigMac ~ . - City, data = mac)
summary(lm.mac)
```

```
##
## Call:
## lm(formula = BigMac ~ . - City, data = mac)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42483 -0.16488  0.00118  0.12128  0.65605
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.2554517  1.5101987   1.493   0.1443
## Bread        0.1052823  0.0910522   1.156   0.2554
## BusFare     -0.2383930  0.1075679  -2.216   0.0333 *
## EngSal      -0.3493607  0.1881340  -1.857   0.0717 .
## EngTax       0.1493621  0.1929697   0.774   0.4441
## Service      0.3525023  0.1772127   1.989   0.0545 .
## TeachSal    -0.3025287  0.1593094  -1.899   0.0658 .
## TeachTax     0.2602043  0.1780313   1.462   0.1528
## VacDays      0.0363043  0.1566651   0.232   0.8181
## WorkHrs     -0.0001588  0.0003976  -0.400   0.6919
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.276 on 35 degrees of freedom
## Multiple R-squared:  0.8648, Adjusted R-squared:  0.8301
## F-statistic: 24.88 on 9 and 35 DF,  p-value: 1.122e-12
```

The only predictor that is significant at the 95% level is `BusFare`. The other significant predictors are `EngSal`, `Service`, and `TeachSal` at the 90% level.

The fact that the highest significance level is 95% is not expected, as it seems that while there is a slight effect of economic predictors on the relative cost of a Big Mac and fries, one might think that there is a slightly higher effect. The predictors that are higher than then 90% level do make sense that they are more significant than the others. The typical salaries of the engineer and the teacher comprise the upper and lower end of salaries in an economy. Seeing that they are more or less equally significant in predicting the time of labor to buy a Big Mac and fries is plausible. Cost of (the 19) services is also plausible as a statistically significant predictor, since that intuitively seems like a broad indicator of the state of the local economy.

c. Use the `princomp` function to conduct PCA on our predictors then run a multiple regression of the response on all PCs.

```
mac.pca <- princomp(mac[2:10], scores = T)

mac.pca.lm <- lm(mac$BigMac ~ ., data = data.frame(mac.pca$scores))
summary(mac.pca.lm)
```

```
##
## Call:
## lm(formula = mac$BigMac ~ ., data = data.frame(mac.pca$scores))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42483 -0.16488  0.00118  0.12128  0.65605
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.7260532  0.0411429  90.564  < 2e-16 ***
## Comp.1       0.0011592  0.0002377   4.877 2.32e-05 ***
## Comp.2       0.3389775  0.0255319  13.277 3.17e-15 ***
## Comp.3       0.1354727  0.0699742   1.936  0.06097 .
## Comp.4      -0.2595021  0.0848919  -3.057  0.00427 **
## Comp.5       0.0047216  0.1108945   0.043  0.96628
## Comp.6      -0.3107160  0.1521543  -2.042  0.04873 *
## Comp.7      -0.3980072  0.1648045  -2.415  0.02110 *
## Comp.8      -0.1340445  0.1854722  -0.723  0.47465
## Comp.9      -0.1534198  0.3094385  -0.496  0.62313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.276 on 35 degrees of freedom
## Multiple R-squared:  0.8648, Adjusted R-squared:  0.8301
## F-statistic: 24.88 on 9 and 35 DF,  p-value: 1.122e-12
```

The multiple R^2 is the same as that of (b). This is likely since we did not reduce any dimensions because we kept all PCs. Something else that caught my eye is that there are more statistically significant coefficients in this model.

```
lm(BigMac ~ Bread, data = mac) %>% summary()
```

```
##
## Call:
## lm(formula = BigMac ~ Bread, data = mac)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.84895 -0.31817 -0.07832  0.30280  1.25856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.88728    0.27834   6.780 2.68e-08 ***
## Bread        0.64076    0.09388   6.825 2.31e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4692 on 43 degrees of freedom
## Multiple R-squared:   0.52,  Adjusted R-squared:  0.5088
## F-statistic: 46.58 on 1 and 43 DF,  p-value: 2.306e-08
```

```
lm(mac$BigMac ~ Comp.1, data = data.frame(mac.pca$scores)) %>% summary()
```

```
##
## Call:
## lm(formula = mac$BigMac ~ Comp.1, data = data.frame(mac.pca$scores))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.11834 -0.53637 -0.09618  0.46345  1.66277
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.7260532  0.0962097  38.728   <2e-16 ***
## Comp.1      0.0011592  0.0005558   2.086    0.043 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6454 on 43 degrees of freedom
## Multiple R-squared:  0.09187,    Adjusted R-squared:  0.07075
## F-statistic:  4.35 on 1 and 43 DF,  p-value: 0.04298
```

The coefficient for `Bread` changes drastically: it goes from 0.105 to 0.641. The coefficient for `Comp.1` stays exactly the same at 0.001. This is likely due to the nature of principle components re-arranging variables using all the variables without loss of information. That is, when using `Comp.1` as the sole predictor, there is no loss of information, whereas when using `Bread` as the sole predictor, there is a loss of information. This then results in the adjustment for the coefficient for `Bread` and no adjustment for the coefficient for `Comp.1`.

e. Using the R function provided in lecture, apply the Contour Regression of the response on the other predictors.

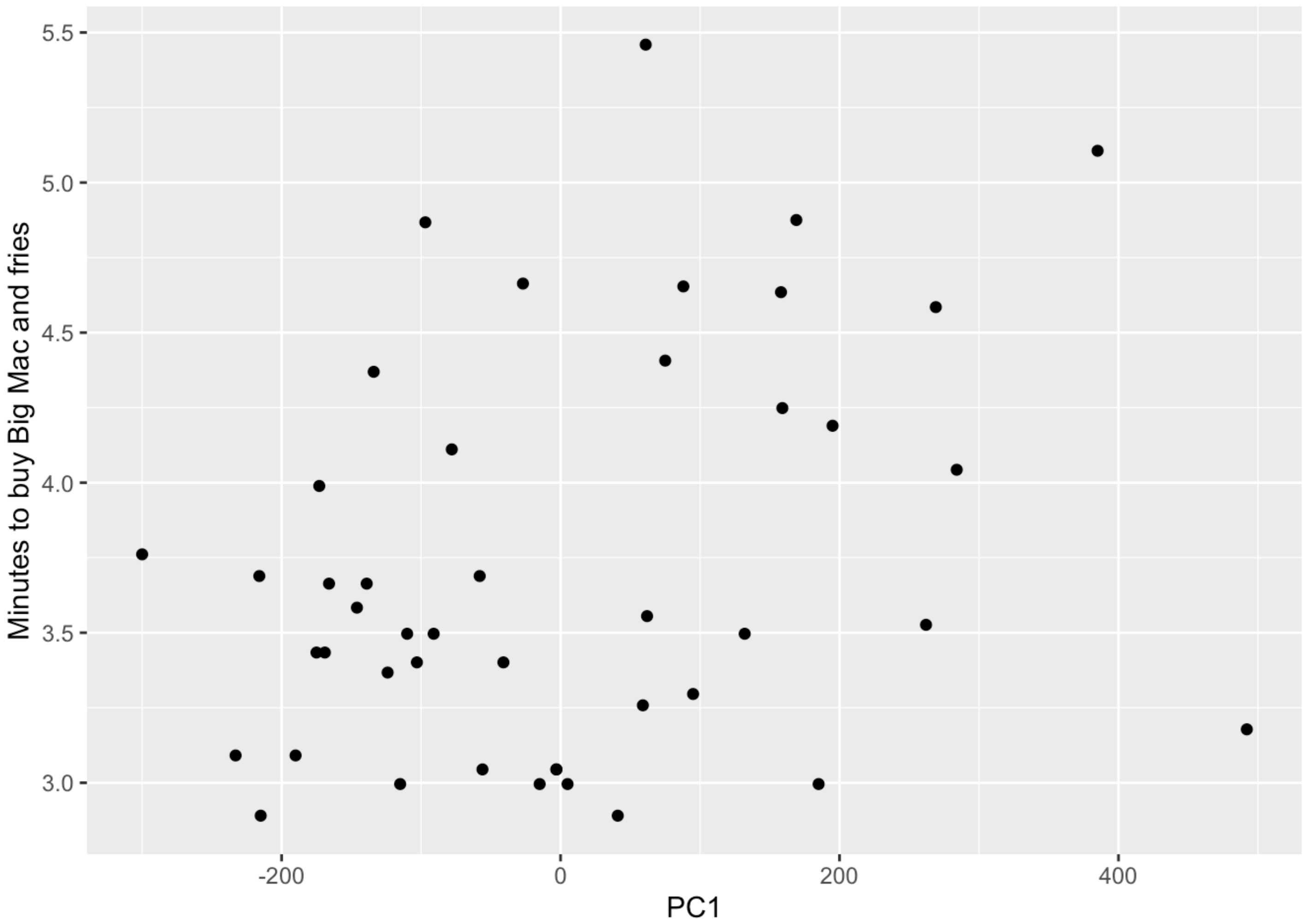
```
## This function is from [Sufficient Dimension Reduction - Methods and Applications with R]
## By Bing Li
## Function Set-up
matpower = function(a, alpha){
  a = round((a + t(a))/2, 7)
  tmp = eigen(a)
  return(tmp$vectors %*% diag((tmp$values)^alpha) %*% t(tmp$vectors))
}

discretize = function(y, h){
  n = length(y)
  m = floor(n/h)
  y = y + .00001 * mean(y) * rnorm(n)
  yord = y[order(y)]
  divpt = numeric()
  for (i in 1:(h-1)) divpt = c(divpt, yord[i*m+1])
  y1 = rep(0, n)
  y1[y<divpt[1]] = 1
  y1[y>=divpt[h-1]] = h
  for (i in 2:(h-1)) y1[(y>=divpt[i-1]) & (y<divpt[i])] = i
  return(y1)
}



cr = function(x,y,percent){

  tradeindex12 = function(k,n){
    j = ceiling(k/n)
    i = k - (j-1)*n
    return(c(i,j))
  }
  mu=apply(x,2,mean);signrt=matpower(var(x),-1/2)
  z=t(t(x)-mu)%*%signrt
  n=dim(x)[1];p = dim(x)[2]

  ymat=matrix(y,n,n)
  deltay=c(abs(ymat - t(ymat)))
  singleindex=(1:n^2)[deltay < percent*mean(deltay)]

  contourmat=diag(1,p)
  for(k in singleindex){

    doubleindex=tradeindex12(k,n)
    deltaz=z[doubleindex[1],]-z[doubleindex[2],]
    contourmat=contourmat-deltaz %*% t(deltaz)

  }
  signrt=matpower(var(x),-1/2)
  return(signrt%*%eigen(contourmat)$vectors)
}
```

First, we apply the `cr()` function on the response and the other predictors.

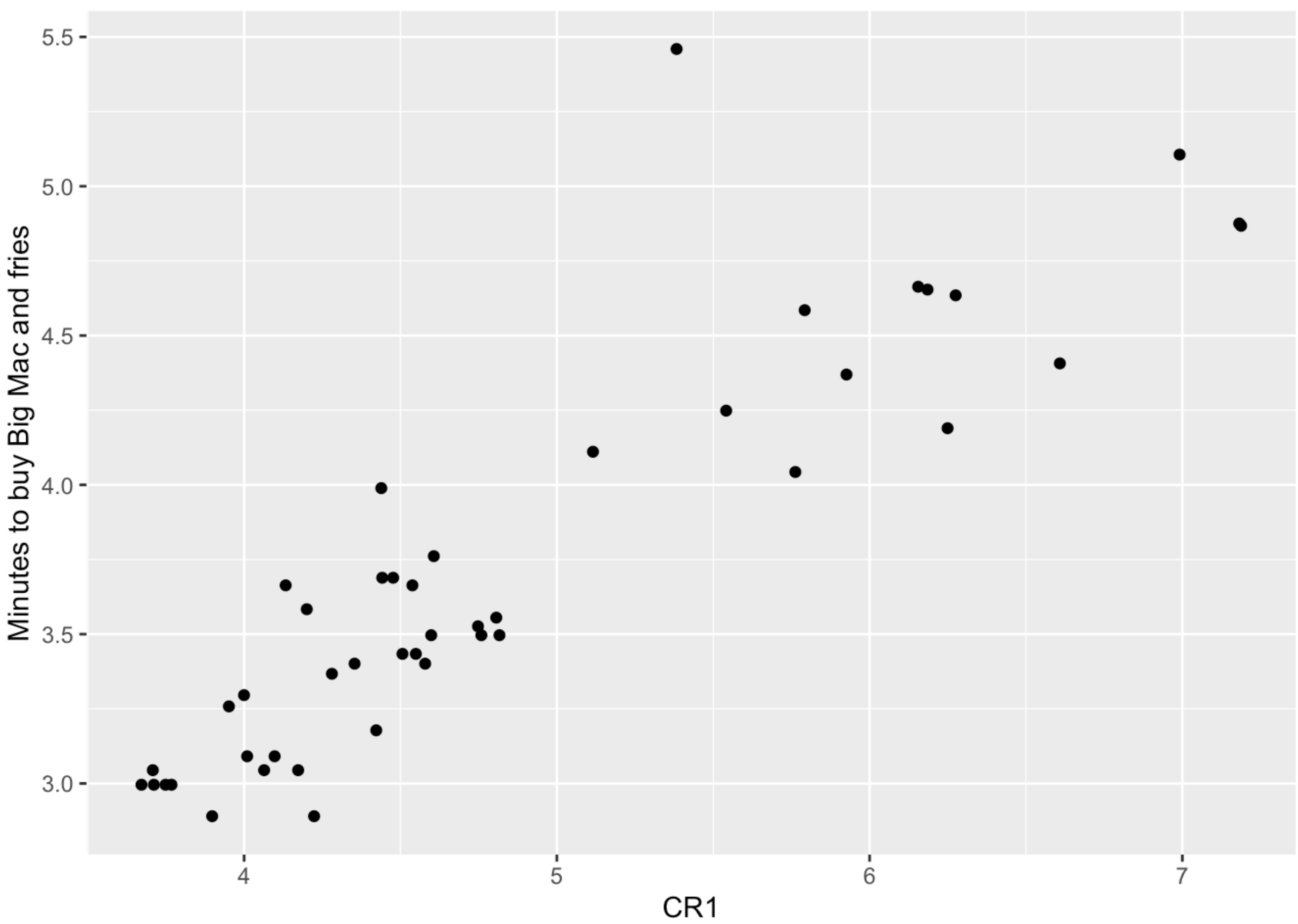
```
mac.cr <- cr(mac[,2:10], mac[,1], 0.05)
```

Then, we make two plots: the response vs. the first principle component from (b); and the response vs. the first sufficient predictor from the contour regression.

```
ggplot() +
  geom_point(aes(x = mac.pca$scores[,1],
                 y = mac$BigMac)) +
  labs(x = 'PC1',
       y = 'Minutes to buy Big Mac and fries')
```




```
ggplot() +
  geom_point(aes(x = as.matrix((mac[,2:10])) %*%
                   as.matrix(mac.cr[,1]),
                 y = mac$BigMac)) +
  labs(x = 'CR1',
       y = 'Minutes to buy Big Mac and fries')
```



There does not seem to be as strong of a correlation between the first principle component and the average minutes of labor to buy a Big Mac and fries, while there is a much clearer correlation for the first sufficient predictor from the contour regression. This is likely because the Central Subspace given by the contour regression contains the regression information of the response variable on the predictors.

Problem 2 (Pen digit data).

Using the data `pendigits.tra` , train the contour regression model to classify the digits 0, 6, and 9.

```
## Pen digit data
pen.train <-read.table("pendigits.tra", sep=",")
pen.tes <-read.table("pendigits.tes", sep=",")

names(pen.train) = c(paste0(c("X"), rep(1:16)), "digit")
names(pen.tes) = c(paste0(c("X"), rep(1:16)), "digit")

str(pen.train)
```

```
## 'data.frame':    7494 obs. of  17 variables:
##  $ X1   : int  47 0 0 0 0 100 0 0 13 57 ...
##  $ X2   : int  100 89 57 100 67 100 100 39 89 100 ...
##  $ X3   : int  27 27 31 7 49 88 3 2 12 22 ...
##  $ X4   : int  81 100 68 92 83 99 72 62 50 72 ...
##  $ X5   : int  57 42 72 5 100 49 26 11 72 0 ...
##  $ X6   : int  37 75 90 68 100 74 35 5 38 31 ...
##  $ X7   : int  26 29 100 19 81 17 85 63 56 25 ...
##  $ X8   : int  0 45 100 45 80 47 35 0 0 0 ...
##  $ X9   : int  0 15 76 86 60 0 100 100 4 75 ...
##  $ X10  : int  23 15 75 34 60 16 71 43 17 13 ...
##  $ X11  : int  56 37 50 100 40 37 73 89 0 100 ...
##  $ X12  : int  53 0 51 45 40 0 97 99 61 50 ...
##  $ X13  : int  100 69 28 74 33 73 65 36 32 75 ...
##  $ X14  : int  90 2 25 23 20 16 49 100 94 87 ...
##  $ X15  : int  40 100 16 67 47 20 66 0 100 26 ...
##  $ X16  : int  98 6 0 0 0 20 0 57 100 85 ...
##  $ digit: int  8 2 1 4 1 6 4 0 5 0 ...
```

```
head(pen.train)
```

```
##    X1  X2 X3  X4  X5  X6  X7  X8 X9 X10 X11 X12 X13 X14 X15 X16 digit
## 1  47 100 27  81  57  37  26   0  0  23  56  53 100  90  40  98     8
## 2   0  89 27 100  42  75  29  45 15  15  37   0  69   2 100   6     2
## 3   0  57 31  68  72  90 100 100 76  75  50  51  28  25  16   0     1
## 4   0 100  7  92   5  68  19  45 86  34 100  45  74  23  67   0     4
## 5   0  67 49  83 100 100  81  80 60  60  40  40  33  20  47   0     1
## 6 100 100 88  99  49  74  17  47  0  16  37   0  73  16  20  20     6
```

```
dim(pen.train)
```

```
## [1] 7494   17
```

```
train = pen.train %>% filter(digit==0|digit==6|digit==9)
test = pen.tes %>% filter(digit==0|digit==6|digit==9)
head(train)
```

```
##    X1  X2 X3  X4 X5 X6 X7 X8  X9 X10 X11 X12 X13 X14 X15 X16 digit
## 1 100 100 88  99 49 74 17 47   0  16  37   0  73  16  20  20     6
## 2   0  39  2  62 11  5 63  0 100  43  89  99  36 100   0  57     0
## 3  57 100 22  72  0 31 25  0  75  13 100  50  75  87  26  85     0
## 4  74  87 31 100  0 69 62 64 100  79 100  38  84   0  18   1     9
## 5  91  74 54 100  0 87 23 59  81  67 100  39  79   4  21   0     9
## 6  99  80 63 100 25 76 79 68 100  62  97  23  54   0   0  16     9
```

```
head(test)
```

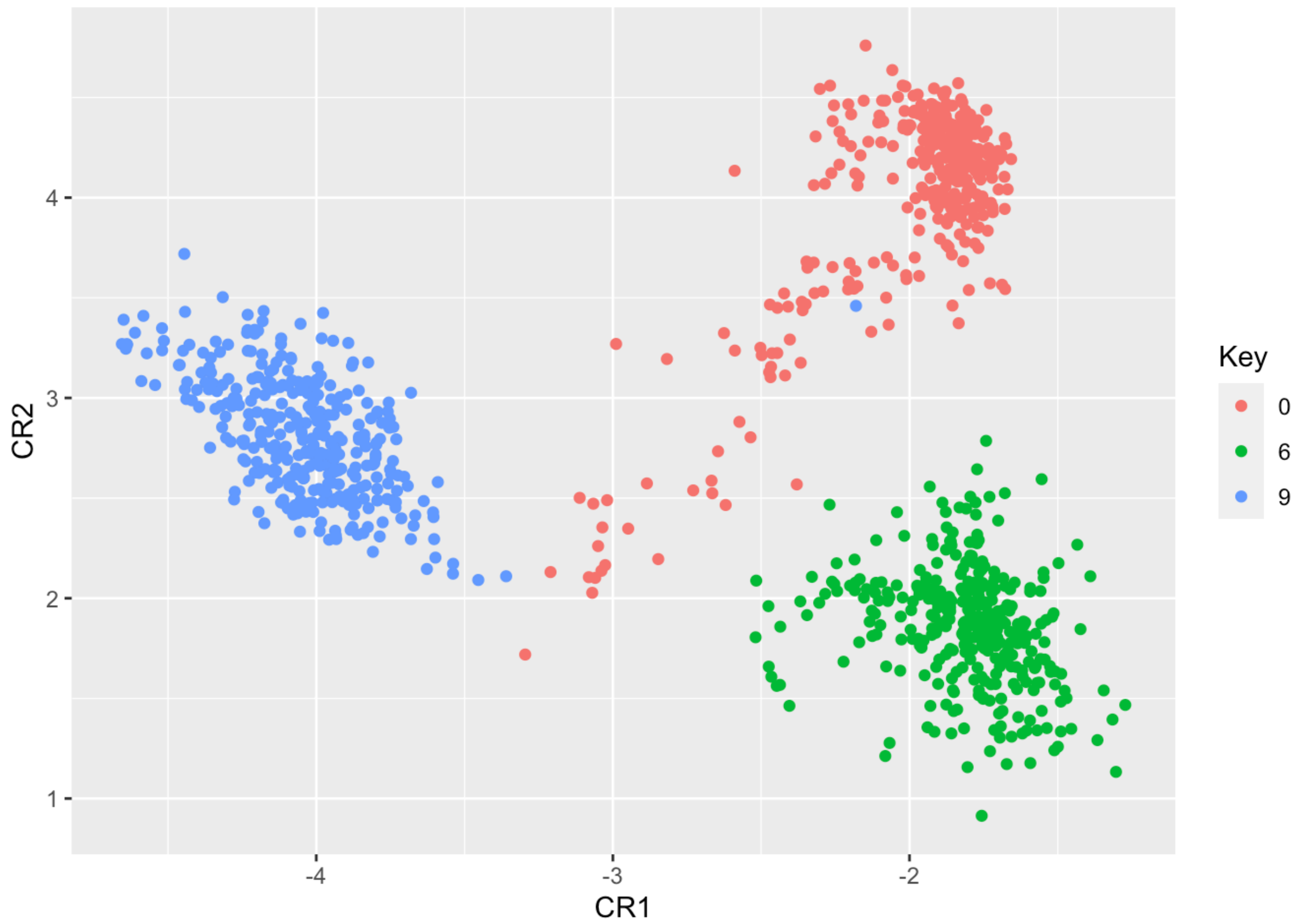
```
##    X1  X2  X3  X4 X5  X6 X7 X8  X9 X10 X11 X12 X13 X14 X15 X16 digit
## 1 95  82  71 100 27  77 77 73 100  80  93  42  56  13   0   0     9
## 2 68 100   6  88 47  75 87 82  85  56 100  29  75   6   0   0     9
## 3 79  87  98  81 71 100 72 73 100  66  91  21  48   0   0  13     9
## 4 92  95  30 100 34  68 87 89  84  78 100  35  64   0   0  19     9
## 5 58  64 100  96 27 100  0 63  79  65  91  72  48  36  10   0     9
## 6 34  89   3  70  1  25 49  0 100  23 100  67  56  99   0 100     0
```

```
train_X = train[,1:16]
train_Y = train[,17]
test_X = test[,1:16]
test_Y = test[,17]

# CR
x = train_X
y = train_Y
percent = 0.05
CR_beta = cr(x, y, percent)
CR_beta
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  0.0010124119 -2.668786e-03 -1.634004e-03  0.0077946778  1.802280e-02
##  [2,]  0.0022816884  3.982251e-03 -5.902806e-04  0.0003522869 -9.601800e-03
##  [3,] -0.0003947386 -1.411072e-03  2.956486e-02 -0.0088460567 -3.758852e-02
##  [4,] -0.0055136103  7.922576e-03 -1.825521e-02 -0.0208169361  2.370806e-02
##  [5,] -0.0002331903  9.530300e-04 -4.054093e-02 -0.0368530239  3.062208e-02
##  [6,] -0.0109632967  8.008756e-05  1.285148e-02  0.0470525330 -1.729768e-02
##  [7,] -0.0050387801  2.861033e-03  2.797178e-02  0.0632394523  9.445383e-03
##  [8,] -0.0090240140  2.001338e-04 -1.745835e-03 -0.0073775855 -1.919300e-02
##  [9,] -0.0016491297  6.241952e-03  1.059523e-02 -0.0553897597 -2.674857e-03
## [10,] -0.0095888037  8.912251e-03 -9.921809e-03 -0.0301525533  1.728072e-02
## [11,] -0.0086890296 -5.018414e-03 -6.421887e-03  0.0030188764  2.436907e-02
## [12,] -0.0074312908  1.315905e-02 -1.026347e-02  0.0480358592 -1.718844e-02
## [13,]  0.0006029385  9.622407e-03 -6.127164e-03  0.0105889507 -7.999117e-05
## [14,]  0.0102566410  2.319462e-03 -2.078532e-03 -0.0179296482 -1.915182e-02
## [15,] -0.0037340255 -2.341136e-03  4.718653e-03 -0.0038646708 -6.728844e-03
## [16,] -0.0059843970  1.580523e-02 -7.732427e-05  0.0045390649  7.478609e-03
##               [,6]         [,7]         [,8]         [,9]         [,10]
##  [1,] -0.014071691  0.001912061  0.006327591  0.010961176  0.0206224065
##  [2,] -0.016822352  0.002016280 -0.011664062  0.002606689  0.0007783003
##  [3,]  0.025523713 -0.007200518 -0.039956460  0.014262480 -0.0352253528
##  [4,]  0.034960976 -0.047897145 -0.020894083 -0.018617798 -0.0135416677
##  [5,]  0.004385989 -0.004915581 -0.001852932  0.019823275  0.0452655588
##  [6,] -0.002122093  0.075184852  0.056582879  0.005476924  0.0685748594
##  [7,]  0.034616110  0.011828500 -0.001670586  0.022116408 -0.0067439419
##  [8,] -0.071894025 -0.058251865 -0.089675184  0.002008673 -0.0694809111
##  [9,] -0.044543706  0.013852319 -0.029123970  0.024454002  0.0080523866
## [10,]  0.061640298 -0.017999293  0.092900880 -0.055586939  0.0574549803
## [11,]  0.036697878 -0.044607939  0.043630009  0.016321873 -0.0025115463
## [12,] -0.019130745  0.049548448 -0.050336641  0.015362937 -0.0622973998
## [13,] -0.001480708  0.054573156 -0.030261704  0.026509759 -0.0559574987
## [14,] -0.015323670 -0.049276951  0.007543431  0.025250726  0.0315612902
## [15,]  0.002696228 -0.039875800  0.015479948  0.022135738  0.0343039336
## [16,]  0.006904698  0.007386234 -0.011578647 -0.043045415  0.0057259275
##               [,11]        [,12]        [,13]        [,14]        [,15]
##  [1,]  0.0465281542  0.019388544 -0.024828414 -0.017518083  0.008161895
##  [2,]  0.0365124369 -0.047631804  0.068494013  0.045896672 -0.029909112
##  [3,]  0.0001713158 -0.041857306  0.027188974  0.007340827 -0.018301886
##  [4,] -0.0245741725  0.062778021 -0.046439801  0.031007359  0.010233859
##  [5,]  0.0166689584  0.040529405 -0.014956564 -0.009006184  0.024669546
##  [6,]  0.0036473493 -0.062971327  0.007054861  0.024585220  0.008332577
##  [7,] -0.0076411685 -0.048363325  0.021869885  0.009648230 -0.015912609
##  [8,] -0.0402340177  0.017104803 -0.021956257 -0.001839648  0.003260778
##  [9,] -0.0225593345  0.048890301 -0.005339874  0.012623929  0.025606264
## [10,]  0.0665569141 -0.032787829  0.037038143 -0.022177502  0.046269001
## [11,]  0.0344328489 -0.069492080 -0.004791143 -0.028348639 -0.011150059
## [12,] -0.0023227491  0.038402791 -0.047568437  0.029831755 -0.109812220
## [13,] -0.0168323855  0.048586782  0.006986790  0.011594561  0.038418133
## [14,] -0.0001397437 -0.036288799  0.001255502  0.021108057  0.133223456
## [15,]  0.0072413201  0.004524664  0.027258158 -0.004611640 -0.024638842
## [16,]  0.0187166024 -0.007989217 -0.004003010 -0.027328747 -0.057632405
##               [,16]
##  [1,] -3.816162e-03
##  [2,] -4.339804e-03
##  [3,]  4.265216e-03
##  [4,] -9.914742e-03
##  [5,] -1.948208e-05
##  [6,] -2.084553e-02
##  [7,]  1.483599e-03
##  [8,] -5.315804e-03
##  [9,] -7.018811e-04
## [10,]  1.166368e-02
## [11,] -1.237817e-02
## [12,] -1.266694e-02
## [13,]  4.723359e-03
## [14,]  2.031187e-02
## [15,] -1.320045e-02
## [16,] -3.801739e-02
```

```
CR_train = as.matrix(x) %*% CR_beta
CR_test = as.matrix(test_X) %*% CR_beta

CR = as.data.frame(CR_test)
CR$Y = test_Y
CR$Y[which(CR$Y == 0)] = '0'
CR$Y[which(CR$Y == 6)] = '6'
CR$Y[which(CR$Y == 9)] = '9'
CR$Y = as.factor(CR$Y)
```

```
ggplot() +
  geom_point(aes(x = as.matrix(test_X) %*%
                   as.matrix(CR_beta[,1]),
                 y = as.matrix(test_X) %*%
                   as.matrix(CR_beta[,2]),
                 color = CR$Y)) +
  labs(x = 'CR1',
       y = 'CR2',
       color = 'Key')
```



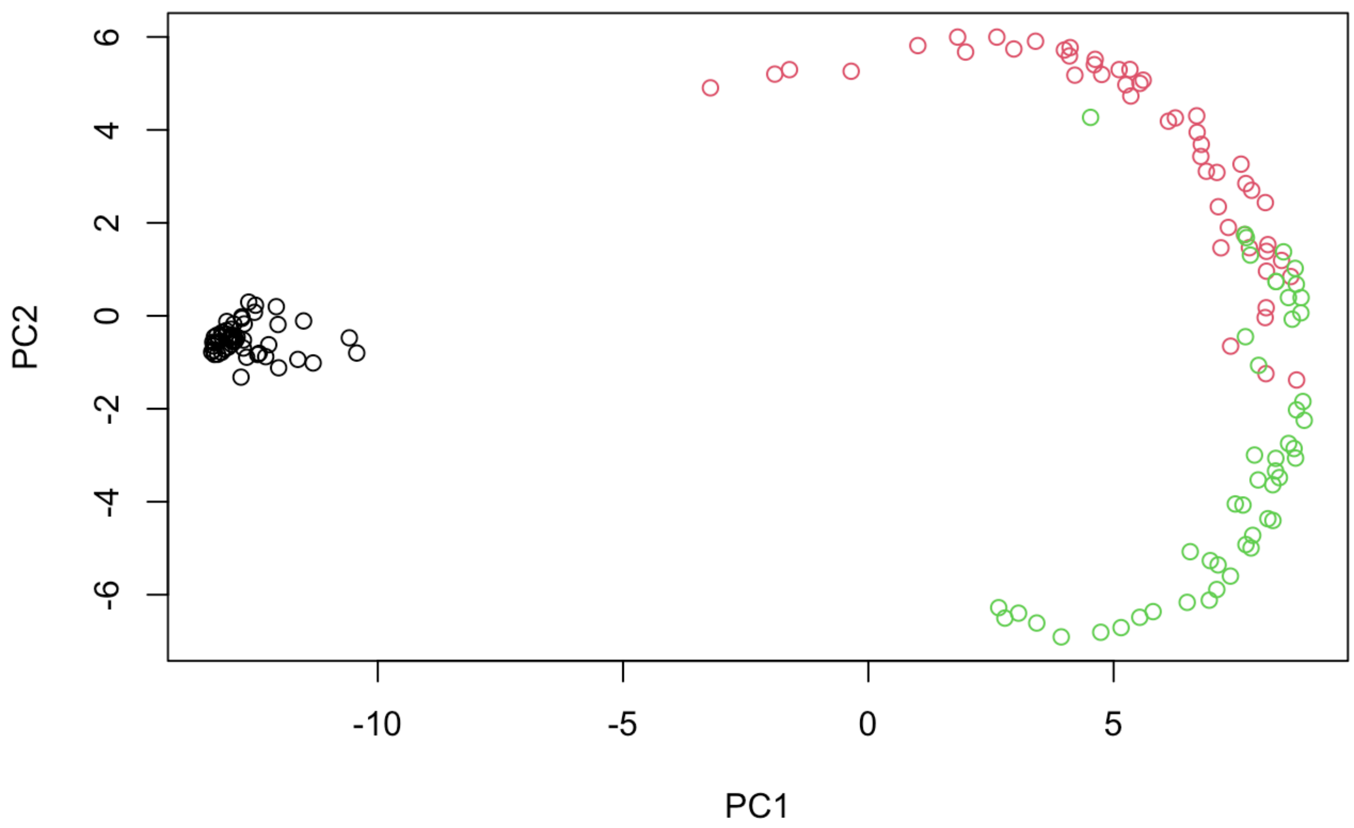
Problem 3 (Kernel PCA).

Write own code to apply Kernel PCA to the `iris` data using the kernel

$$\kappa(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{5}||\mathbf{x}-\mathbf{y}||^2\right).$$

Plot to show the result of kPCA.

```r
dat <- iris[,1:4] %>% as.data.frame()

# set up kernel
kernel <- function(dat) {
  # vec <- as.vector(X - Y)
  # exp(-(1/5) * sum(vec^2))
  mat <- NULL
  for (i in seq(1, nrow(dat))) {
    for (j in seq(1, nrow(dat))) {
      k_ij <- exp(-(1/5) * sum((dat[i,] - dat[j,])^2))
      mat <- append(mat, k_ij)
    }
  }
  mat <- matrix(mat, nrow=nrow(dat))
  mat
}

K = kernel(dat)

# recenter the kernel matrix
L = matrix(1/nrow(dat), nrow=nrow(K), ncol=ncol(K))
K2 = K - (L %*% K) - (K %*% L) + (L %*% K %*% L)

# solve the eigen decomposition problem
res = eigen(K2/nrow(dat))

pc <- res$vectors[,1:2]/sqrt(res$values[1:2]) # choose first two
## choosing all gives NaNs in row 150 since we are sqrt a neg eigenvalue

# obtain the kPCA score vector
score <- K2 %*% pc

# visualize
plot(score, col = as.integer(iris[,5]),
     xlab = 'PC1',
     ylab = 'PC2')
```



Now we compare our kPCA function from scratch with the kpca() function.

```r
# compare result with kPCA function

library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.1.2
```

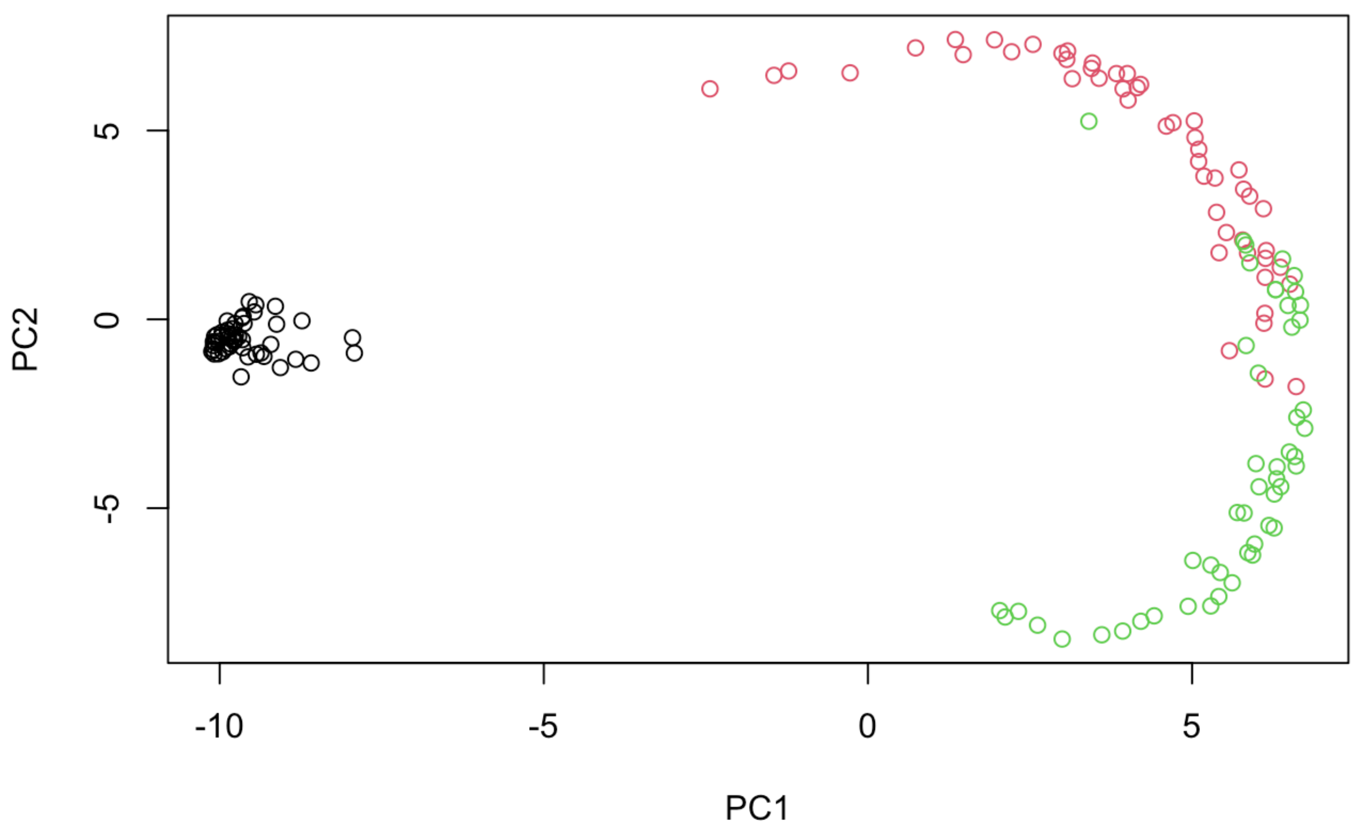
```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
kpc <- kpca(~., dat, kernel='rbfdot',
            kpar = list(sigma = .2), features = 2)
head(pcv(kpc))
```

```
##             [,1]         [,2]
## [1,] -0.2072415 -0.038803529
## [2,] -0.2012447 -0.024874040
## [3,] -0.2042499 -0.048115197
## [4,] -0.1992343 -0.025815252
## [5,] -0.2072764 -0.045066822
## [6,] -0.1872144 -0.007203895
```

```
# plot
plot(rotated(kpc), col = as.integer(iris[,5]),
     xlab = 'PC1', ylab = 'PC2')
```



They appear to be the same.