

Compulsory exercise 2: Group 16

TMA4268 Statistical Learning V2022

Weicheng Hua, Emil Johannesen Haugstvedt, Torbjørn Baadsvik

31 March, 2022

Contents

Problem 1	2
a)	2
b)	3
c)	3
d)	5
Problem 2	5
a)	6
b)	7
Problem 3	8
a	8
b	8
Problem 4	8
a)	8
b)	9
c)	9
d)	11
Problem 5	13
a)	13
b)	13

Problem 6	15
a)	15
b)	18
c)	20
d)	20
d)	21

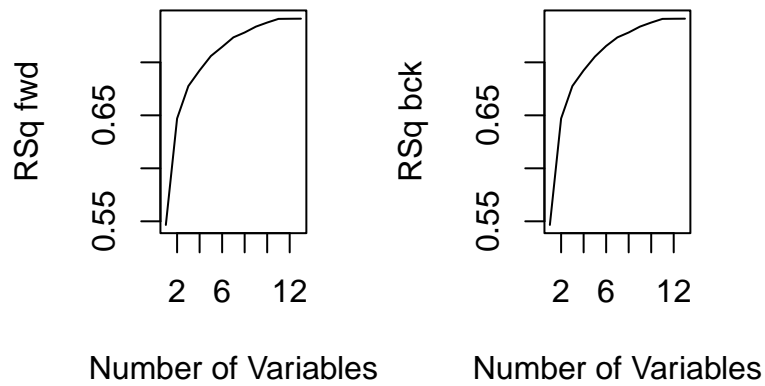
Problem 1

```
## 'data.frame': 506 obs. of 14 variables:
## $ crim : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn : num 18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
## $ chas : int 0 0 0 0 0 0 0 0 0 0 ...
## $ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
## $ rm : num 6.58 6.42 7.18 7 7.15 ...
## $ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis : num 4.09 4.97 4.97 6.06 6.06 ...
## $ rad : int 1 2 2 3 3 3 5 5 5 5 ...
## $ tax : num 296 242 242 222 222 222 311 311 311 311 ...
## $ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
## $ black : num 397 397 393 395 397 ...
## $ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
## $ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

## [1] 404 14
```

a)

```
library(leaps)
fwd_regfit <- regsubsets(medv~., data = boston.train, nvmax = 14, method = "forward" )
summary_fwd_regfit <- summary(fwd_regfit)
bck_regfit <- regsubsets(medv~., data = boston.train, nvmax = 14, method = "backward" )
summary_bck_regfit <- summary(bck_regfit)
par(mfrow=c(1,2))
plot(summary_fwd_regfit$rsq, xlab = "Number of Variables", ylab = "RSq fwd" ,type = "l")
plot(summary_bck_regfit$rsq, xlab = "Number of Variables", ylab = "RSq bck" ,type = "l")
```



b)

```
number_predictors_selected <- 4
variables_fwd <- names(coef(fwd_regfit, id = number_predictors_selected))
variables_fwd
```

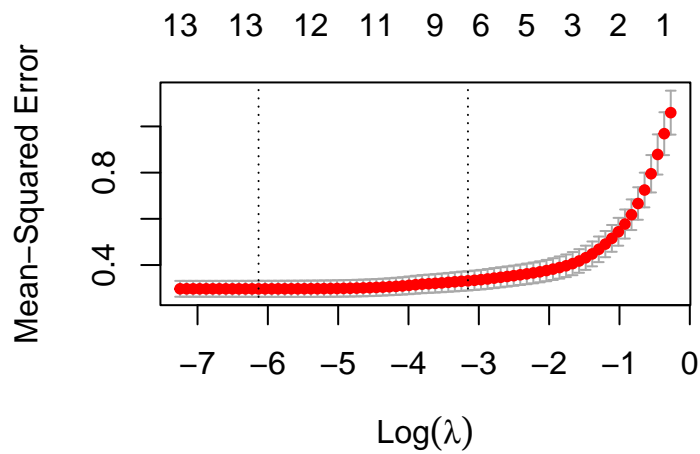
```
## [1] "(Intercept)" "rm" "dis" "ptratio" "lstat"
```

The four “predictors” from the forward stepwise selection are “rm”, “dis”, “ptratio” and “lstat”.

c)

```
library(glmnet)
set.seed(1)
x_train <- model.matrix(medv~., boston.train)[,-1]
y_train <- boston.train$medv

model_lass <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 5)
plot(model_lass)
```



```
best_lambda <- model_lass$lambda.min
best_lambda
```

```
## [1] 0.002172032
```

```
coef_lass <- coef(model_lass, s = model_lass$lambda.min )
coef_lass
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##              s1
## (Intercept)  0.023625259
## crim        -0.081492676
## zn           0.094199322
## indus        0.003673933
## chas         0.087338926
## nox          -0.174725136
## rm           0.312944488
## age          -0.010989639
## dis          -0.315732328
## rad          0.269191647
## tax          -0.207059619
## ptratio     -0.204123213
## black        0.102992828
## lstat       -0.428585719
```

(ii)

```
best_lambda <- model_lass$lambda.min
best_lambda
```

```
## [1] 0.002172032
```

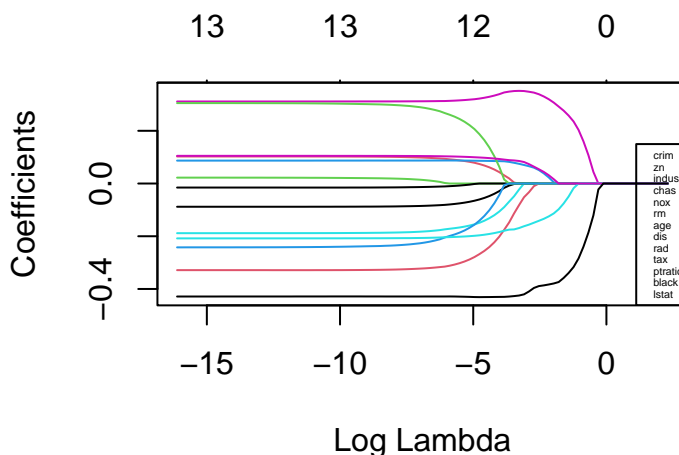
Th best lambda value is given as:0.00315

- (iii) The fitted coefficients at the best λ value is given by the function below. A plot for the coefficients value at different lambda is given as well. Indus is not a relevant coefficient at the lambda value with lowest MSE and the coefficient for age is very small meaning that it is most likely significantly less important than the rest. This is possible to do for scaled data.

```
coef_lass <- coef(model_lass, s = model_lass$lambda.min )
coef_lass

## 14 x 1 sparse Matrix of class "dgCMatrix"
##              s1
## (Intercept)  0.023625259
## crim        -0.081492676
## zn          0.094199322
## indus        0.003673933
## chas         0.087338926
## nox         -0.174725136
## rm           0.312944488
## age         -0.010989639
## dis         -0.315732328
## rad          0.269191647
## tax         -0.207059619
## ptratio     -0.204123213
## black        0.102992828
## lstat       -0.428585719

lambdas_to_try <- 10^seq(-7,1, length.out = 100)
res<- glmnet(x_train, y_train, alpha=1, lambda =lambdas_to_try ,standardize= FALSE)
plot(res, xvar= "lambda")
legend("bottomright", legend =colnames(x_train), cex =0.3)
```



d)

- (i) True. Lasso is generally faster than step-wise especially when n (number of datapoints) and p (number of predictors) are very large and the number of relatively important predictors are small. This is because lasso

can eliminate multiple predictors at once by increasing the value of lambda while stepwise can only eliminate it one by one. <https://www.stat.cmu.edu/~ryantibs/papers/bestsubset.pdf>

(ii) False. It is impossible for ridge regression to result in coefficients equal to zero. The coefficients approach zero as lambda's value get extremely large. The coefficients can however become zero in Lasso regression.

(iii) False. Lasso is expected to perform better when there is a only relatively small number of important predictors and a significant proportion of unimportant coefficients that have very small or zero value. Ridge is expected to perform better when there is a high proportion of important predictors with predictors of all having roughly the same size.

(iv) True. The formula for Elastic Net is given as

$$\min_{\beta_0, \beta} \frac{1}{N} \sum_{i=1}^N w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda [(1 - \alpha) \|\beta\|_2^2 / 2 + \alpha \|\beta\|_1]$$

Where the α value can be varied between 0 and 1 to change the weighting between Ridge and Lasso.

Problem 2

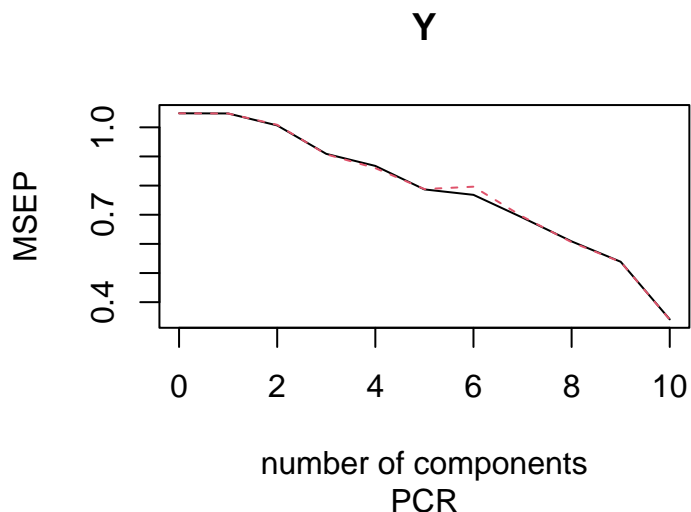
```
library(MASS)
set.seed(1)
# load a synthetic dataset
id <- "1CWZYfrL0rFdrIZ6Hv73e3xxt0SFgU4Ph" # google file ID
synthetic <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id))
# split into training and test sets
train.ind = sample(1:nrow(synthetic), 0.8 * nrow(synthetic))
synthetic.train = data.frame(synthetic[train.ind, ])
synthetic.test = data.frame(synthetic[-train.ind, ])
# show head(...)
# Y: response variable; X: predictor variable
head(synthetic)
```

##	Y	X1	X2	X3	X4	X5
## 1	-1.43753239	-0.75905055	-0.69720326	-0.3016852	-0.7434697	0.8807558
## 2	-1.70972989	-0.28635632	0.04809182	0.5791725	-0.7446170	0.9935311
## 3	1.33931240	0.09574117	-0.89605758	-0.9636347	0.5554647	-0.5341800
## 4	0.20354906	-0.28702695	1.72952687	1.4289705	-0.1596993	-0.7161976
## 5	-0.09261896	0.02345825	0.51201583	0.1544345	0.4318039	-0.8674060
## 6	1.69952325	1.19231791	-0.98179754	-0.9567773	-0.6933918	0.4656891
##	X6	X7	X8	X9	X10	
## 1	-0.8705750	-0.7448252	-0.4639697	0.62502272	-0.8149674	
## 2	0.3532248	-0.5860332	-0.7964403	0.84868110	-0.1065119	
## 3	0.4707434	-0.6588069	-0.7327518	-0.29429307	0.6588927	
## 4	-0.7774007	0.2502145	0.5987052	-0.04428773	0.6247479	
## 5	-0.9066908	0.8946086	-0.9700185	0.09082626	0.6102134	
## 6	-0.7381794	0.8650175	0.4108119	0.75677429	-0.2281439	

a)

```
library(pls)
```

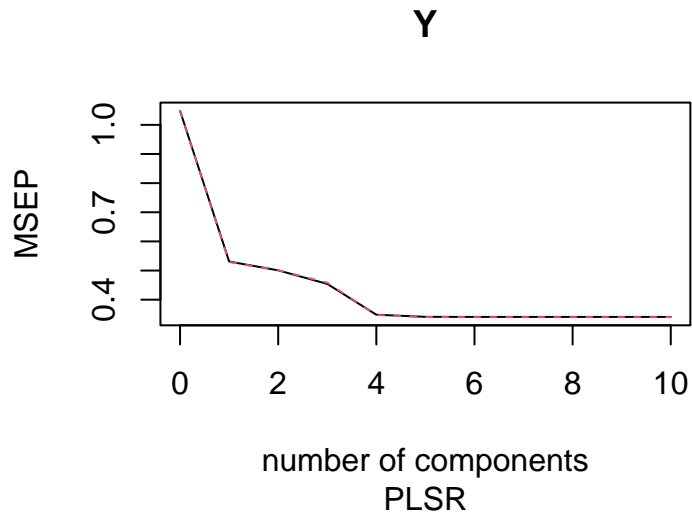
```
pcr_model <- pcr(Y~., data =synthetic.train, scale =TRUE, validation ="CV")
validationplot(pcr_model, val.type ="MSEP")
title(sub ="PCR")
```



```
summary(pcr_model)
```

```
## Data:      X dimension: 800 10
## Y dimension: 800 1
## Fit method: svdpc
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##      (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.024    1.024    1.003    0.9534   0.9314   0.8871   0.8765
## adjCV        1.024    1.023    1.004    0.9525   0.9274   0.8877   0.8923
##      7 comps  8 comps  9 comps 10 comps
## CV           0.8307   0.7801   0.7340   0.5839
## adjCV        0.8329   0.7784   0.7335   0.5834
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    19.7903   31.827   43.00   53.52   63.50   72.97   82.30   91.30
## Y     0.1648    5.761   15.31   19.65   25.73   25.93   35.56   42.87
##      9 comps 10 comps
## X     99.53   100.00
## Y     49.78    68.28
```

```
plsr_model<- plsr(Y~., data =synthetic.train, scale =TRUE, validation ="CV")
validationplot(plsr_model, val.type = "MSEP")
title(sub = "PLSR")
```



b)

The PCR method show an almost uniform reduction in MSEP with increasing number of principal components. On the other hand, PLSR showed a sharp drop in MSEP when moving from 0 to 1 component and from 3 to 4 components, with the MSEP approaching zero from 4 components and onwards. The main difference between the PCR and the PLSR method is that in PCR, the principal components are created without considering their significance to Y, while in PLSR the principal components are created such that each additional principal components is weighted to have less significance to the response variable.

Problem 3

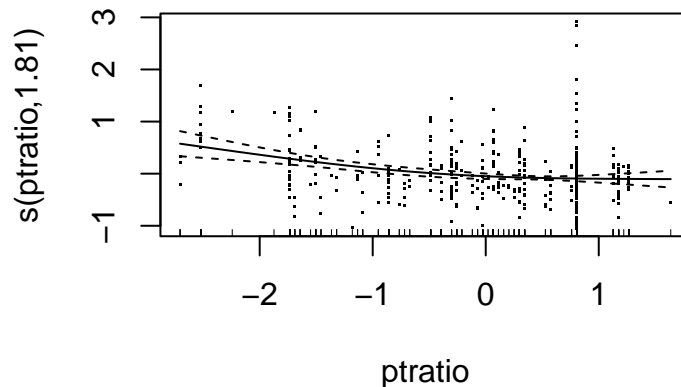
a

- (i) True. The proof for this is pretty long and complicated. You can see it in the link
- (ii) False. By increasing the number of cutpoints each step function will get more and more affected by the points within their range, and thus overfit.
- (iii) False. The penalty term is $\int g''(t)^2 dt$.
- (iv) True. With high k the number more neighbors are needed in order to classify a point, thus the variance will be low and the bias will increase.

b


```
# Fit model
model <- gam(medv ~ rm + s(ptratio, k = 3) + poly(lstat, df = 2), data = boston.train)

# Plot model with training data
plot(model, boston.train)
```



Problem 4

a)

- (i) False. The trees can handle interaction terms, but you can not specify them yourself. The tree will “find” them by its nature.
- (ii) True.
- (iii) True.
- (iv) True. It decides the number of “iterations” in the boosting algorithm.

b)

c)

First some given R code

```
library(tidyverse)
library(palmerpenguins) # Contains the data set "penguins".
data(penguins)

names(penguins) <- c("species", "island", "billL", "billD", "flipperL", "mass", "sex", "year")

Penguins_reduced <- penguins %>%
```

```

dplyr::mutate(mass = as.numeric(mass),
             flipperL = as.numeric(flipperL),
             year = as.numeric(year)) %>%
drop_na()

# We do not want "year" in the data (this will not help for future predictions)
Penguins_reduced <- Penguins_reduced[,-c(8)]

set.seed(4268)
# 70% of the sample size for training set
training_set_size <- floor(0.7 * nrow(Penguins_reduced))
train_ind <- sample(seq_len(nrow(Penguins_reduced)), size = training_set_size)
train <- Penguins_reduced[train_ind, ]
test <- Penguins_reduced[-train_ind, ]

```

(i)

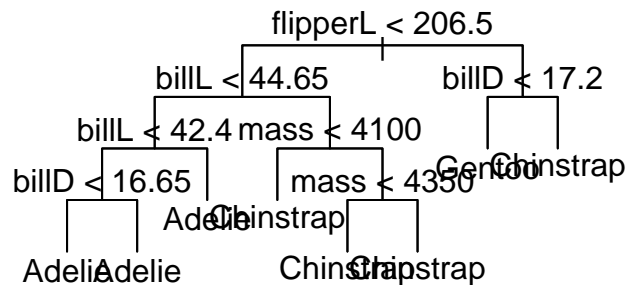
```

library(tree)

# Create the tree
tree.penguin <- tree(species ~ .,
                    data = train,
                    split = 'gini')

# Plot the tree
plot(tree.penguin, type = 'uniform');text(tree.penguin)

```



(ii) Now, apply 10-fold cross-validation

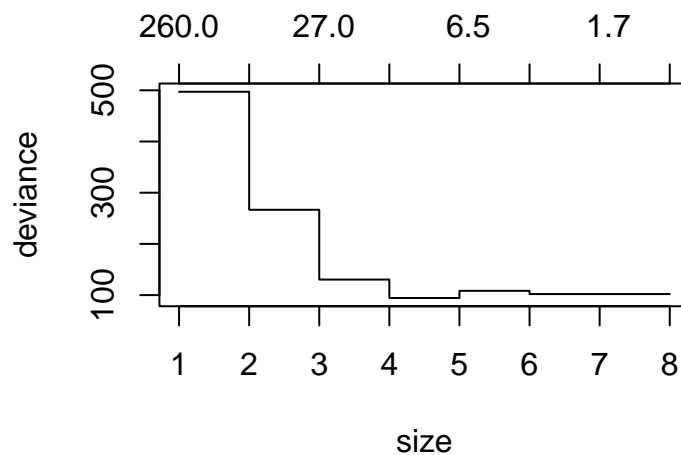
```

set.seed(123)

```

```
# Do 10-fold cross-validation
cv.penguin <- cv.tree(tree.penguin)

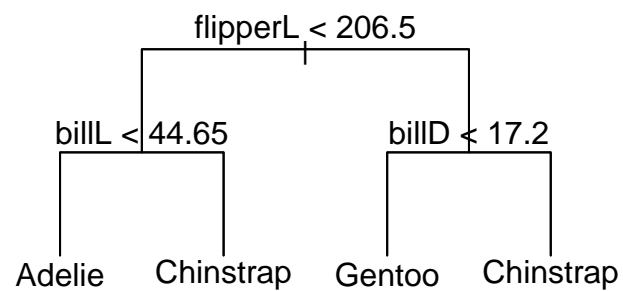
# Plot result from the 10-fold cross-validation
plot(cv.penguin)
```



(iii) From the above plot the optimal tree looks to be of size 4.

```
# Prune the tree according to the observation in the above plot
prune_tree.penguin <- prune.misclass(tree.penguin, best = 4)

plot(prune_tree.penguin, type = 'uniform'); text(prune_tree.penguin)
```



```
pred <- predict(prune_tree.penguin, test, type = 'class')
tab <- table(pred, test$species)
tab
```

```
##
## pred      Adelie Chinstrap Gentoo
##   Adelie      42        5       1
##   Chinstrap    0       15       0
##   Gentoo       0        0      37
```

Above you can see a table showing how good, and bad, the tree performs on the test data.

Now we will calculate the misclassification rate using this table.

```
missclassification_rate <- round((1 - sum(diag(tab))/sum(tab)), 2)
print(paste('The missclassification rate is:', missclassification_rate))
```

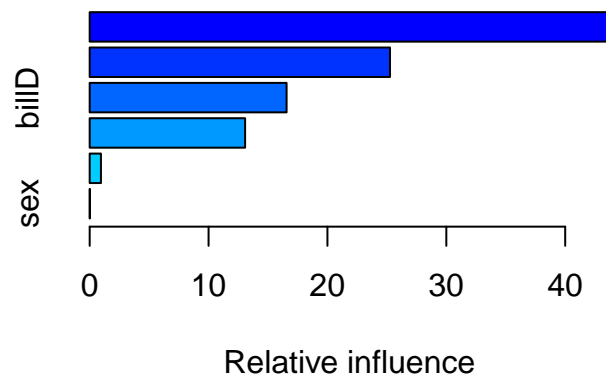
```
## [1] "The missclassification rate is: 0.06"
```

d)

```
library(gbm)
library(dismo)

# Fit boosted forest with initial parameters
boosted.penguin <- gbm(species~ .,
  data = train,
  n.trees = 100,
  distribution = 'multinomial')

summary(boosted.penguin)
```



```
##           var      rel.inf
## billL      billL 44.13675398
## flipperL flipperL 25.26210607
## billD      billD 16.56220398
## island     island 13.07455667
## mass       mass  0.94557585
## sex        sex   0.01880345
```

From the above plot you can see that “billL” and “flipperL” are the most influential ones when it comes to predicting the species of the penguins.

Now we predict on the test data and calculate the misclassification rate:

```
pred <- predict.gbm(boosted.penguin, test, type = 'response')

pred.penguin <- colnames(pred)[apply(pred,1,which.max)]

tab.boost <- table(pred.penguin, test$species)
tab.boost
```

```
##
## pred.penguin Adelie Chinstrap Gentoo
##   Adelie      42         2        0
##   Chinstrap   0         18        0
##   Gentoo      0         0        38
```

Above you can see a table showing the right and wrong classifications of the boosted forest on the test data.

Now we will calculate the misclassification rate using this table.

```
misclassification.rate.boost <- 1-sum(diag(tab.boost))/sum(tab.boost)
print(paste('The misclassification rate is:', misclassification.rate.boost))
```

```
## [1] "The misclassification rate is: 0.02"
```

Problem 5

a)

i)	ii)	iii)	iv)
FALSE	TRUE	FALSE	TRUE

b)

```
svc.cvtune <- function(kernel, paramgrid, k){
  ctrl <- tune.control(sampling="cross", cross=k, nrepeat = 1)
  cvtune.result <- tune(method=svm, species=., kernel=eval(kernel),
    data=train, ranges=paramgrid, tunecontrol=ctrl)
  cvtune.result
}
```

```

k <- 5
svc.fit_and_eval <- function(kernel, paramgrid, k){
  res <- svc.cvttune(kernel, paramgrid, k)
  print(strrep("_", 60))
  print(paste(eval(kernel), "support vector classifier"))
  print(strrep("-", 60))
  print("Parameters:")
  print(c(res$best.parameters))

  model <- res$best.model
  pred <- as.factor(predict(model, test[-c(1)]))
  cm <- confusionMatrix(data=pred, reference=test$species)
  print(cm)
}

svc.fit_and_eval("linear", data.frame(cost=.1*1:20), k)

```

```

## [1] "-----"
## [1] "linear support vector classifier"
## [1] "-----"
## [1] "Parameters:"
## $cost
## [1] 1.8
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie      42         0         0
##   Chinstrap    0         20         0
##   Gentoo       0          0        38
##
## Overall Statistics
##
##           Accuracy : 1
##           95% CI : (0.9638, 1)
##   No Information Rate : 0.42
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##
##   McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: Adelie Class: Chinstrap Class: Gentoo
## Sensitivity           1.00           1.0           1.00
## Specificity           1.00           1.0           1.00
## Pos Pred Value        1.00           1.0           1.00
## Neg Pred Value        1.00           1.0           1.00
## Prevalence            0.42           0.2           0.38
## Detection Rate        0.42           0.2           0.38
## Detection Prevalence  0.42           0.2           0.38

```

```
## Balanced Accuracy          1.00          1.0          1.00

svc.fit_and_eval("radial", data.frame(cost=.1*1:20, gamma=.1*1:20), k)

## [1] "-----"
## [1] "radial support vector classifier"
## [1] "-----"
## [1] "Parameters:"
## $cost
## [1] 1.1
##
## $gamma
## [1] 0.4
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie      42         0         0
##   Chinstrap    0        20         0
##   Gentoo       0         0        38
##
## Overall Statistics
##
##           Accuracy : 1
##           95% CI : (0.9638, 1)
##   No Information Rate : 0.42
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: Adelie Class: Chinstrap Class: Gentoo
## Sensitivity           1.00           1.0           1.00
## Specificity           1.00           1.0           1.00
## Pos Pred Value        1.00           1.0           1.00
## Neg Pred Value        1.00           1.0           1.00
## Prevalence            0.42           0.2           0.38
## Detection Rate        0.42           0.2           0.38
## Detection Prevalence  0.42           0.2           0.38
## Balanced Accuracy     1.00           1.0           1.00
```

Using a linear rather than radial kernel in the support vector classifier yields slightly superior results on the test set. Thus, the linear kernel is preferred.

Problem 6

a)

```
id <- "1NJ1SuUBebl5P8rMSIwm_n3S8a7K43yP4" # google file ID
happiness <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id), fileEncoding="UTF-8")
colnames(happiness)
```

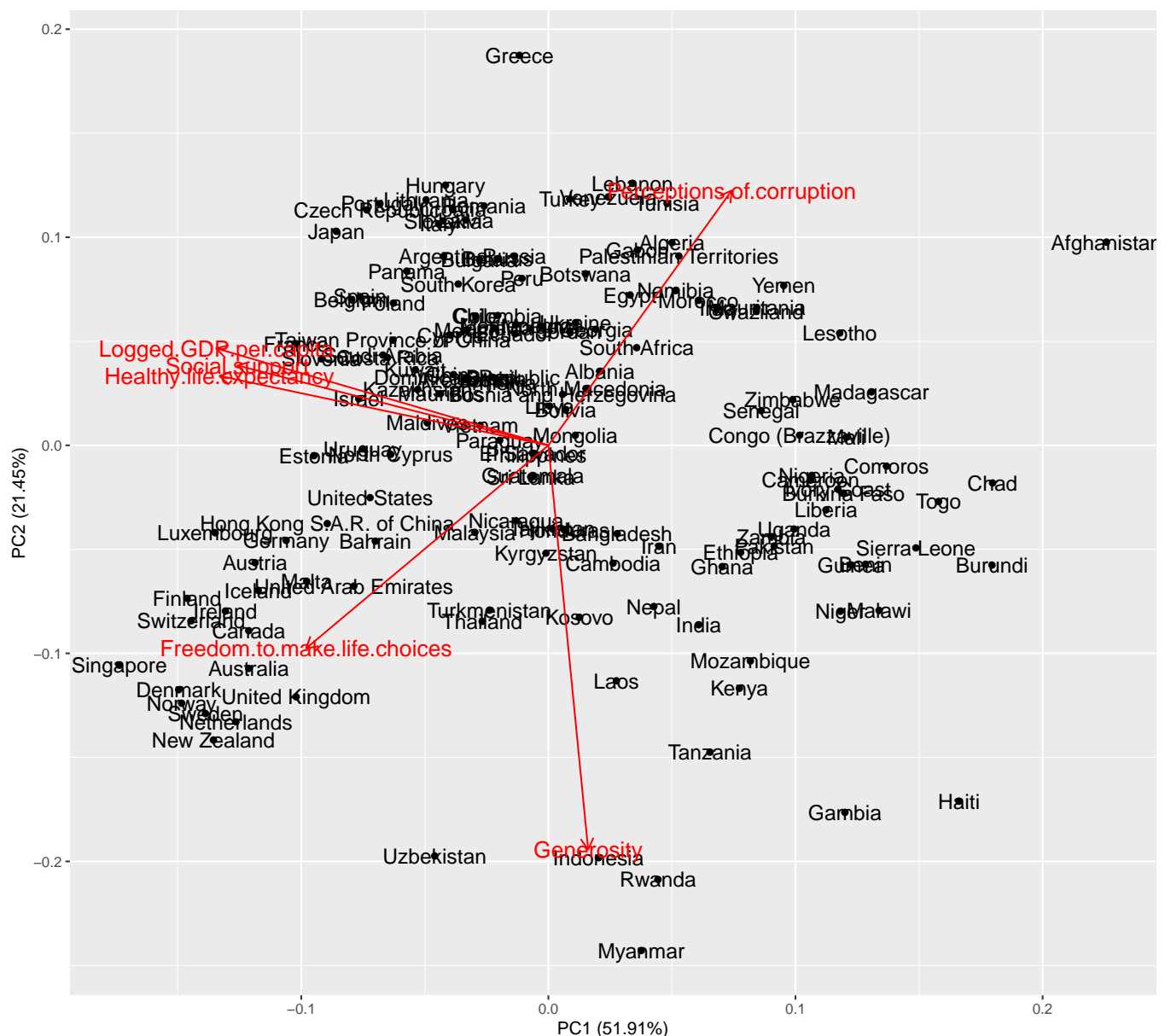
```
## [1] "Country.name"
## [2] "Regional.indicator"
## [3] "Ladder.score"
## [4] "Standard.error.of.ladder.score"
## [5] "upperwhisker"
## [6] "lowerwhisker"
## [7] "Logged.GDP.per.capita"
## [8] "Social.support"
## [9] "Healthy.life.expectancy"
## [10] "Freedom.to.make.life.choices"
## [11] "Generosity"
## [12] "Perceptions.of.corruption"
## [13] "Ladder.score.in.Dystopia"
## [14] "Explained.by..Log.GDP.per.capita"
## [15] "Explained.by..Social.support"
## [16] "Explained.by..Healthy.life.expectancy"
## [17] "Explained.by..Freedom.to.make.life.choices"
## [18] "Explained.by..Generosity"
## [19] "Explained.by..Perceptions.of.corruption"
## [20] "Dystopia...residual"
```

```
cols = c('Country.name',
          'Ladder.score', # happiness score
          'Logged.GDP.per.capita',
          'Social.support',
          'Healthy.life.expectancy',
          'Freedom.to.make.life.choices',
          'Generosity', # how generous people are
          'Perceptions.of.corruption')
# We continue with a subset of 8 columns:
happiness = subset(happiness, select = cols)
rownames(happiness) <- happiness[, c(1)]
# And we create an X and a Y matrix
happiness.X = happiness[, -c(1, 2)]
happiness.Y = happiness[, c(1, 2)]
happiness.XY = happiness[, -c(1)]
# scale
happiness.X = data.frame(scale(happiness.X))
str(happiness)
```

```
## 'data.frame': 149 obs. of 8 variables:
## $ Country.name : chr "Finland" "Denmark" "Switzerland" "Iceland" ...
## $ Ladder.score : num 7.84 7.62 7.57 7.55 7.46 ...
## $ Logged.GDP.per.capita : num 10.8 10.9 11.1 10.9 10.9 ...
```



```
## $ Social.support      : num  0.954 0.954 0.942 0.983 0.942 0.954 0.934 0.908 0.948 0.934 ..
## $ Healthy.life.expectancy : num  72 72.7 74.4 73 72.4 73.3 72.7 72.6 73.4 73.3 ...
## $ Freedom.to.make.life.choices: num  0.949 0.946 0.919 0.955 0.913 0.96 0.945 0.907 0.929 0.908 ...
## $ Generosity           : num -0.098 0.03 0.025 0.16 0.175 0.093 0.086 -0.034 0.134 0.042 ..
## $ Perceptions.of.corruption : num  0.186 0.179 0.292 0.673 0.338 0.27 0.237 0.386 0.242 0.481 ..
```



i)

We observe that the variables “Logged.GDP.per.capita”, “Healthy.life.expenctancy” and “Social.support” are highly correlated as they have nearly equal loadings on PC1 and PC2. The loading vectors for the variables “Freedom.to.make.like.choices” and “Perceptions.of.corruption” are nearly antiparallel, indicating that these variables have a strong negative correlation.

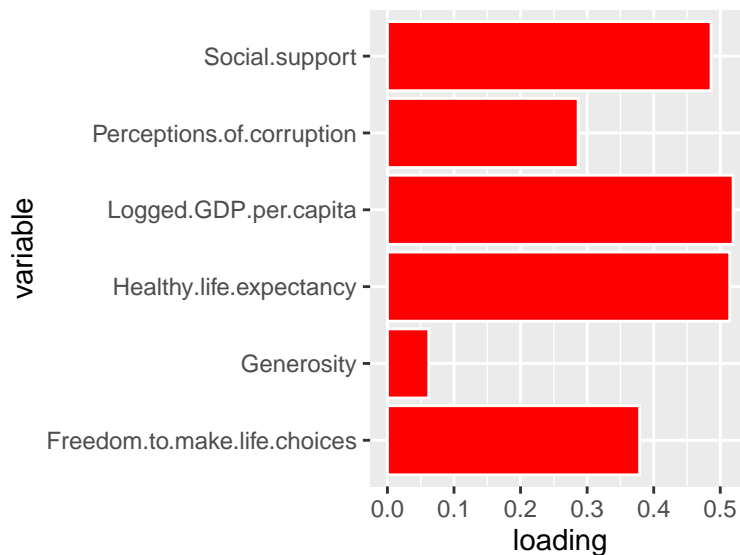
ii)

Afghanistan appears to be clearly separated from the other countries in this plot, and may be considered to be an outlier.

b)

i)

```
rot <- pca_mat$rotation
loading.PC1 <- data.frame(variable=rownames(rot), loading=abs(rot[,1]))
ggplot(data=loading.PC1, aes(x=variable, y=loading)) +
  geom_col(color="white", fill="red") +
  coord_flip()
```



ii)

```
summary(happiness.XY)
```

##	Ladder.score	Logged.GDP.per.capita	Social.support	Healthy.life.expectancy
##	Min. :2.523	Min. : 6.635	Min. :0.4630	Min. :48.48
##	1st Qu.:4.852	1st Qu.: 8.541	1st Qu.:0.7500	1st Qu.:59.80
##	Median :5.534	Median : 9.569	Median :0.8320	Median :66.60

```
## Mean :5.533 Mean : 9.432 Mean :0.8147 Mean :64.99
## 3rd Qu.:6.255 3rd Qu.:10.421 3rd Qu.:0.9050 3rd Qu.:69.60
## Max. :7.842 Max. :11.647 Max. :0.9830 Max. :76.95
## Freedom.to.make.life.choices Generosity Perceptions.of.corruption
## Min. :0.3820 Min. : -0.28800 Min. :0.0820
## 1st Qu.:0.7180 1st Qu.: -0.12600 1st Qu.:0.6670
## Median :0.8040 Median : -0.03600 Median :0.7810
## Mean :0.7916 Mean : -0.01513 Mean :0.7274
## 3rd Qu.:0.8770 3rd Qu.: 0.07900 3rd Qu.:0.8450
## Max. :0.9700 Max. : 0.54200 Max. :0.9390
```

```
plsr_model.1 <- plsR(formula=Ladder.score~., data=happiness.XY,
                     scaleX=T, scaleY=T, nt=1)
```

```
## _____
## ____Component____ 1 ____
## ____Predicting X without NA neither in X nor in Y____
## ****_****
```

```
plsr_model.1
```

```
## Number of required components:
## [1] 1
## Number of successfully computed components:
## [1] 1
## Coefficients:
##                                     [,1]
## Intercept                        -1.97230057
## Logged.GDP.per.capita             0.23557013
## Social.support                     2.27673343
## Healthy.life.expectancy           0.03925535
## Freedom.to.make.life.choices      1.85325316
## Generosity                       -0.04082942
## Perceptions.of.corruption         -0.81205234
## Information criteria and Fit statistics:
##      AIC      RSS_Y      R2_Y R2_residY RSS_residY AIC.std DoF.dof
## Nb_Comp_0 447.0933 170.69015      NA      NA 148.00000 425.8403 1.000000
## Nb_Comp_1 241.9638  42.50913 0.7509573 0.7509573  36.85832 220.7108 2.927703
##      sigmahat.dof AIC.dof BIC.dof GMDL.dof DoF.naive sigmahat.naive
## Nb_Comp_0  1.0739236 1.1610522 1.1843038 17.26983      1  1.0739236
## Nb_Comp_1  0.5376203 0.2966547 0.3137149 -77.34935      2  0.5377525
##      AIC.naive BIC.naive GMDL.naive
## Nb_Comp_0 1.1610522 1.1843038 17.26983
## Nb_Comp_1 0.2930593 0.3047194 -80.93057
```

iii)

```
rot.PC1 <- plsr_model.1$ww
rot.PC1
```

```
##                                     Comp_1
```

```
## Logged.GDP.per.capita      0.78975970
## Social.support             0.75688765
## Healthy.life.expectancy    0.76809946
## Freedom.to.make.life.choices 0.60775307
## Generosity                 -0.01779928
## Perceptions.of.corruption  -0.42114000
```

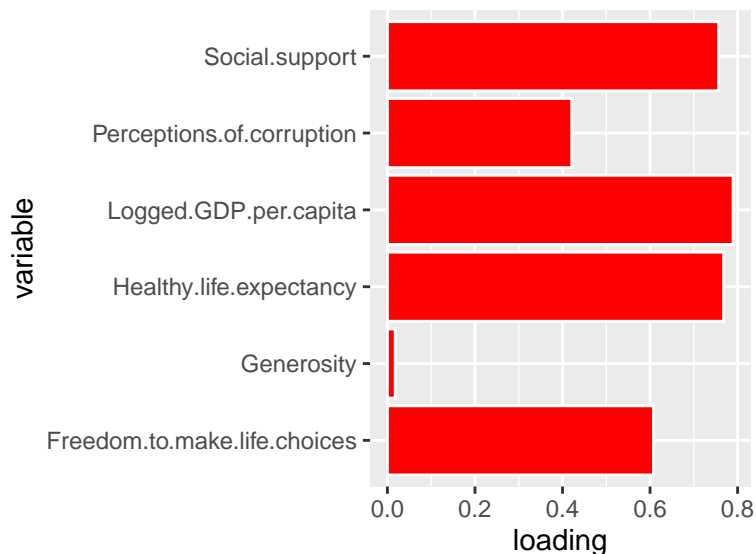
```
rownames(rot.PC1)
```

```
## [1] "Logged.GDP.per.capita"      "Social.support"
## [3] "Healthy.life.expectancy"    "Freedom.to.make.life.choices"
## [5] "Generosity"                 "Perceptions.of.corruption"
```

```
loading.PC1 <- data.frame(variable=rownames(rot.PC1), loading=abs(rot.PC1[,1]))
loading.PC1
```

```
##                               variable    loading
## Logged.GDP.per.capita      Logged.GDP.per.capita 0.78975970
## Social.support              Social.support      0.75688765
## Healthy.life.expectancy    Healthy.life.expectancy 0.76809946
## Freedom.to.make.life.choices Freedom.to.make.life.choices 0.60775307
## Generosity                  Generosity           0.01779928
## Perceptions.of.corruption  Perceptions.of.corruption 0.42114000
```

```
ggplot(data=loading.PC1, aes(x=variable, y=loading)) +
  geom_col(color="white", fill="red") +
  coord_flip()
```



iv)

Based on the PLSR we see that the variables “Logged.GDP.per.capita”, “Healthy.life.expectancy”, and “Social.support” are the most important predictors.

c)

d)

i)	ii)	iii)	iv)
FALSE	FALSE	TRUE	TRUE

d)

i)

```
K = 3
km.out <- kmeans(happiness.X, K, iter.max = 10)
invalidclustering <- function(km.out){
  clust <- km.out$cluster
  scand <- clust[c("Norway", "Denmark", "Sweden", "Finland")]
  !((sd(scand) == 0) & (clust["United States"] != scand["Norway"]))
}

i <- 1
print(paste("iteration", i))
```

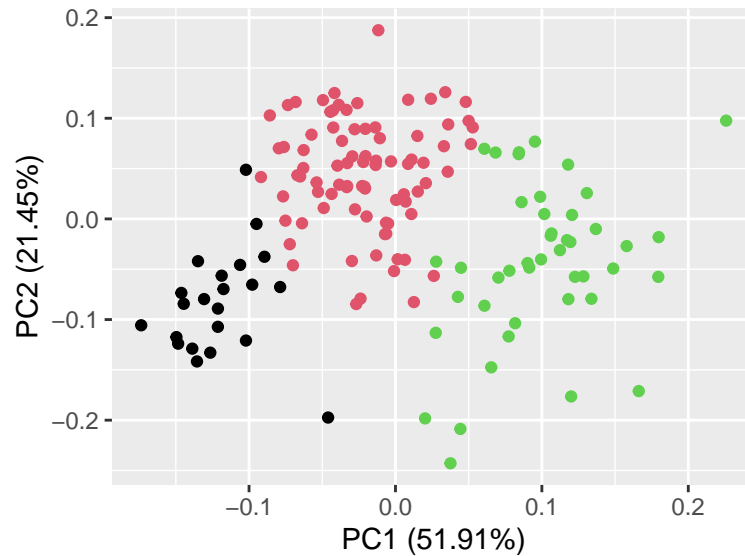
```
## [1] "iteration 1"
```

```
while(invalidclustering(km.out)){
  i <- i + 1
  print(paste("iteration", i))
  km.out <- kmeans(happiness.X, K, iter.max = 10)
}
```

```
## [1] "iteration 2"
```

```
## [1] "iteration 3"
```

```
autoplot(pca_mat, data = happiness.X, colour=km.out$cluster,
          label=F, label.size=5,
          loadings = F, loadings.colour = 'blue',
          loadings.label = F, loadings.label.size = 3)
```



$K = 3$ was the minimum parameter required to obtain the desired clustering conditions.

ii)

```
clust <- km.out$cluster
clust[c("Norway", "Denmark", "Sweden", "Finland")]
```

```
## Norway Denmark Sweden Finland
##      1      1      1      1
```

```
clust["United States"]
```

```
## United States
##           2
```

```
mean_ladder <- sapply(1:3, function(i) mean(happiness.Y[names(clust[clust == i]),2]))
mean_ladder
```

```
## [1] 7.015455 5.711000 4.455455
```

We observe that the clustering algorithm places the scandinavian countries in cluster 1, and the United States in cluster 2. When computing the average value of the “Ladder.score” variable within each cluster, we find that “Ladder.score” decreases with increasing cluster index. Thus, we conclude that the scandinavian countries belong the “happiest” cluster, and that the United States belong to the “medium happiness” cluster.