Compulsory exercise 2: Group 39

TMA4268 Statistical Learning V2021

Alexander J. Ohrt, Jim Totland

18 april, 2021

Problem 1

a)

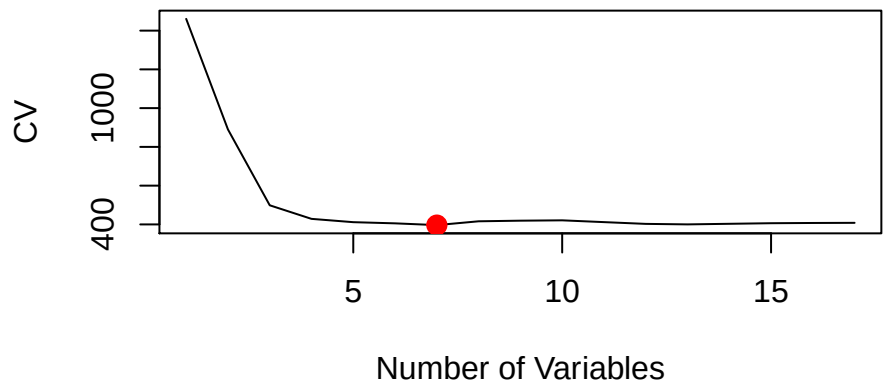
FALSE, TRUE, TRUE, FALSE

b)

We have chosen the cross-validated prediction error as the measure to use in our model selection. A plot showing the cross-validated prediction error is displayed below. The red dot shows the optimal number of variables based on this measure.

```
id <- "1iI6YaqgG0QJW5onZ_GTBsCvpKPExF30G"  # Google file ID.
catdat <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id),
    header = T)

set.seed(4268)
train.ind <- sample(1:nrow(catdat), 0.5 * nrow(catdat))
catdat.train <- catdat[train.ind, ]
catdat.test <- catdat[-train.ind, ]

# Perform best subset selection using all the predictors and the training data.
n <- ncol(catdat.train) - 1  # Number of predictors.
bss.obj <- regsubsets(birds ~ ., data = catdat.train, nvmax = n)  # Best subset selection.
sum <- summary(bss.obj)  # Save summary obj.

# Cross-validated prediction error.  Create a prediction function to make
# predictions for regsubsets with id predictors included.
predict.regsubsets <- function(object, newdata, id, ...) {
    form = as.formula(object$call[[2]])
    mat = model.matrix(form, newdata)
    coefi = coef(object, id = id)
    xvars = names(coefi)
    mat[, xvars] %*% coefi
}

# Create indices to divide the data between folds.
k <- 10
folds <- sample(1:k, nrow(catdat.train), replace = TRUE)
cv.errors <- matrix(NA, k, n, dimnames = list(NULL, paste(1:n)))
```

```r
# Perform CV.
for (j in 1:k) {
    bss.obj <- regsubsets(birds ~ ., data = catdat.train[folds != j, ], nvmax = n)
    for (i in 1:n) {
        pred <- predict(bss.obj, catdat.train[folds == j, ], id = i)
        cv.errors[j, i] <- mean((catdat.train$birds[folds == j] - pred)^2)
    }
}

# Compute mean cv errors for each model size.
mean.cv.errors <- apply(cv.errors, 2, mean)
bss.cv <- which.min(mean.cv.errors)

# Plot the mean cv errors.
plot(mean.cv.errors, xlab = "Number of Variables", ylab = "CV", type = "l")
points(bss.cv, mean.cv.errors[bss.cv], col = "red", cex = 2, pch = 20)
```



The optimal number of predictors (in addition to the intercept) according to the cross validated prediction error is 7. We would argue that cross-validation is a more reliable way of selecting among the models, since it is a resampling method, compared to only running one best subset selection on the training data and choosing based on other model selection criteria. To further strengthen this argument, we calculated the test MSE based on the best model found when using the model criteria R^2_{adj}, BIC and C_p. All these cases led to higher values for test MSE compared to the test MSE attained when applying the model selected with cross-validation. Hence, the selected variables are as shown below.

```r
# Best model, based on amount of variables chosen by CV.
coef(bss.obj, bss.cv)  # Selected variables.
```

```
#>       (Intercept)            weight           wetfood    daily.playtime
#>         110.421423        -3.259122        -7.872675       -10.059382
#>       children.13             urban              bell daily.outdoortime
#>           4.391086       -10.391531       -51.853719         1.154146
```

```r
fit <- lm(birds ~ weight + wetfood + daily.playtime + children.13 + urban + bell +
    daily.outdoortime, data = catdat.train)
pred.regbest <- predict(fit, newdata = catdat.test)
```

```r
mse.regbest <- mean((pred.regbest - catdat.test$birds)^2)  # Test Mean Square Error.
mse.regbest
```

```
#> [1] 290.2635
```

The test MSE is 290.2634716.

c)

Using Lasso regression on the same data set leads to the following. In order to choose an optimal value of λ, 10-fold cross-validation is used.

```r
x.train <- model.matrix(birds ~ ., data = catdat.train)[, -1]
y.train <- catdat.train$birds
x.test <- model.matrix(birds ~ ., data = catdat.test)[, -1]
y.test <- catdat.test$birds

lasso.mod <- glmnet(x.train, y.train, alpha = 1)  # alpha = 1 specifies Lasso regression.

# Perform cross validation.
set.seed(4268)
cv.out <- cv.glmnet(x.train, y.train, alpha = 1)
bestlam <- cv.out$lambda.min
bestlam  # Lambda is 'small', so the fit is relatively similar to least squares.
```

```
#> [1] 0.4900502
```

```r
lasso.coeff <- coef(cv.out, s = "lambda.min")
lasso.coeff
```

```
#> 18 x 1 sparse Matrix of class "dgCMatrix"
#>                               1
#> (Intercept)        1.215644e+02
#> sex                .
#> weight            -2.712575e+00
#> dryfood           -2.614636e+00
#> wetfood           -7.696906e+00
#> age                .
#> owner.income      -1.114629e-05
#> daily.playtime    -1.016868e+01
#> fellow.cats        .
#> owner.age         -1.578560e-01
#> house.area         7.068027e-02
#> children.13        3.727465e+00
#> urban             -8.351200e+00
#> bell              -4.985635e+01
#> dogs               3.418310e+00
#> daily.outdoortime  1.139876e+00
#> daily.catnip       .
#> neutered          -4.376255e+00
```

```r
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x.test)
mse.lasso <- mean((lasso.pred - y.test)^2)
mse.lasso
```

```
#> [1] 298.3532
```

As seen in the output above, the test MSE is 298.3532325. Moreover, all coefficients, except `sex`, `age`, `fellow.cats` and `daily.catnip` are non-zero.

d)

When $\lambda \to \infty$, the Lasso regression gives the null model, i.e. the model where all the coefficients are zero. When $\lambda = 0$, the Lasso regression gives the least squares fit.

e)

i) A model with only intercept always predicts the mean of the response in the training data. The test MSE for this model is given below.

```
mse.intercept <- mean((mean(catdat.train$birds) - catdat.test$birds)^2)
mse.intercept
```

```
#> [1] 4914.069
```

ii)

```
# A multiple linear regression.
least.sq <- lm(birds ~ ., data = catdat.train)
yhat.least.sq <- predict(least.sq, newdata = catdat.test)
mse.least.sq <- mean((yhat.least.sq - catdat.test$birds)^2)
mse.least.sq
```

```
#> [1] 301.9186
```

We see that using only an intercept yields a test MSE of 4914.0693576, while a standard linear regression yields a test MSE of 301.9185792. On the other hand, best subset selection resulted in a test MSE of 290.2634716 and lasso regression resulted in a test MSE of 298.3532325. Thus, both models from b) and c) are superior to the ones introduced here, in terms of test MSE.

f)

A table with the test MSE values from best subset selection, lasso regression, intercept-only and ordinary linear regression is shown below.

```
msrate <- cbind(mse.regbest, mse.lasso, mse.intercept, mse.least.sq)
colnames(msrate) <- c("Best Subset", "Lasso", "Intercept-only", "Least Squares")
rownames(msrate) <- c("Test MSE")
msrate
```

```
#>          Best Subset    Lasso Intercept-only Least Squares
#> Test MSE    290.2635 298.3532       4914.069      301.9186
```

The results are as expected. First of all, best subset selection is an exhaustive search in the space of all possible models, which means that it should find the best possible linear model (according to some measure). This could lead to overfitting however, because of the large search space, but has given the best result in this case. Secondly, shrinkage methods, like Lasso regression, are used to reduce overfitting to the training data. Then, the test MSE might be reduced, since the variance might have been reduced more than the bias has increased (bias-variance trade-off) after regularization. In this case, this gives a modest decrease in test MSE compared to the least squares regression. Finally, it is expected that the linear regression has a smaller test MSE compared to the intercept-only model, since predicting every value as the mean of the `birds`-values does not capture any trends in the data.

Problem 2

a)

TRUE, TRUE, FALSE, FALSE.

b)

The basis functions for a cubic spline with knots at the quartiles q_1 and q_2 of variable X are

$$X, X^2, X^3, (X - q_1)_+^3 \text{ and } (X - q_2)_+^3,$$

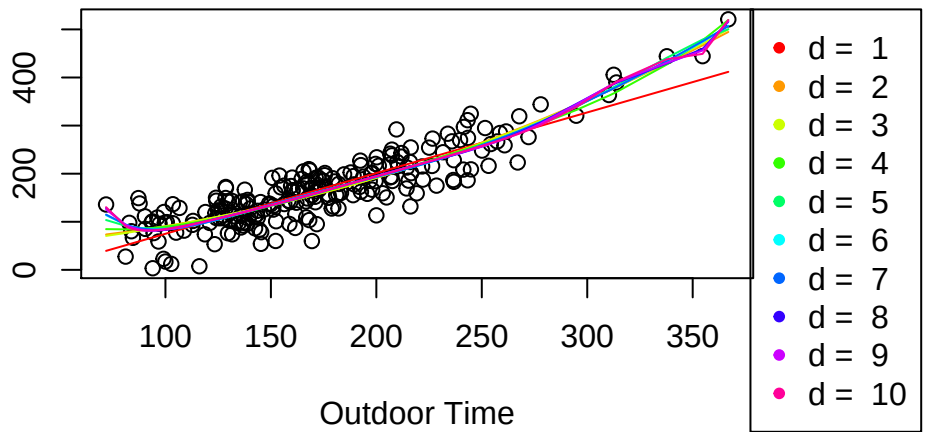
where

$$(X - q_j)_+^d = \begin{cases} (X - q_j)^d & , X > q_j \\ 0 & , \text{otherwise.} \end{cases}$$

c)

(i)

```
# Fit polynomial regression with training data.
outdoor.data <- catdat.train[c("daily.outdoortime", "birds")]  # Pick out necessary training data.
par(mar = c(4, 3, 3, 4.5), xpd = TRUE)  # Change margins of plots.
plot(outdoor.data, main = "", xlab = "Outdoor Time", ylab = "Birds")
deg <- 1:10
col <- rainbow(n = length(deg))
mse.train <- seq(from = 1, to = length(deg))
for (d in 1:length(deg)) {
    fit <- lm(birds ~ poly(daily.outdoortime, d), data = outdoor.data)
    lines(sort(outdoor.data[, 1]), fit$fit[order(outdoor.data[, 1])], col = col[d])
    mse.train[d] = mean((predict(fit, outdoor.data) - outdoor.data[, 2])^2)
}
legend("topright", inset = c(-0.25, 0), legend = paste("d = ", deg), col = col, pch = 20)
```




```
mse.train  # MSE from training data.
```

```
#>  [1] 1369.997 1199.202 1198.499 1188.227 1175.981 1173.485 1173.437 1166.437
#>  [9] 1166.019 1164.368
```

```
which.min(mse.train)
```

```
#> [1] 10
```

(ii) It is seen that the training MSE decreases steadily as the degree of the polynomial regression increases. This is because we increase the flexibility of the model when we increment the polynomial degree, which reduces the training MSE.

Problem 3

a)

TRUE, TRUE, FALSE, FALSE

b)

(**Which leaves?**) The tree from a) pruned down to three leaves should consist of `age < 81.5`, `country: indonesia, japan, Korea` and `country: France`. In essence, the entire left subtree is pruned, while the right subtree is kept as is.

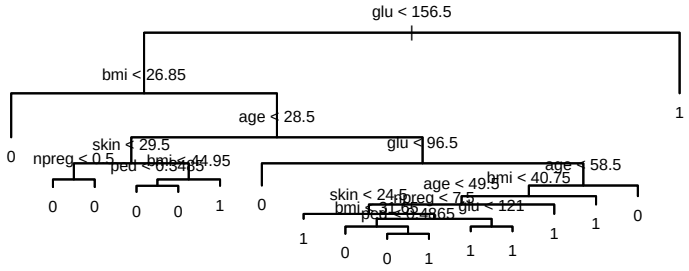
(**Why?**) This is the best way to prune the tree down to three leaves because we assume that the length of the branches is proportional to the decrease in node impurity. Hence, we choose the leaves with the longest branches, i.e. the right subtree.

c)

```
id <- "1Fv6xwKLSZHldRAC1MrcK2mzdOYnbgv0E"  # Google file ID.
d.diabetes <- dget(sprintf("https://docs.google.com/uc?id=%s&export=download", id))
d.train <- d.diabetes$ctrain
d.test <- d.diabetes$ctest
d.train$diabetes <- as.factor(d.train$diabetes)  # Added to make classification tree.
d.test$diabetes <- as.factor(d.test$diabetes)  # Added to make classification tree.
```

(i)

```
# Simple classification tree.
set.seed(1)
simple.tree <- tree(diabetes ~ ., data = d.train, split = "deviance")
plot(simple.tree, type = "proportional", cex = 0.5)
text(simple.tree, cex = 0.5, pretty = 0)
```




```
# Calculate classification error for the unpruned tree.
yhat.simple <- predict(simple.tree, newdata = d.test, type = "class")
conf.table.simple <- table(yhat.simple, d.test$diabetes)
conf.table.simple
```

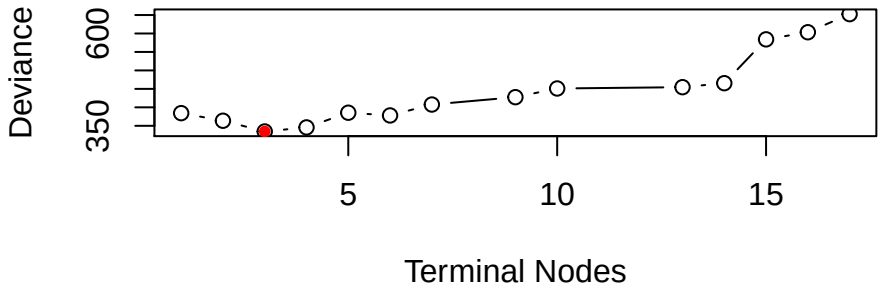
```
#>
#> yhat.simple   0   1
#>           0 126  28
#>           1  29  49
```

```
class.rate.simple <- 1 - sum(diag(conf.table.simple))/(sum(conf.table.simple))
class.rate.simple
```

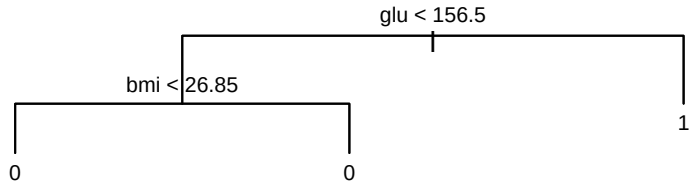
```
#> [1] 0.2456897
```

```
# Apply cost complexity pruning.
cv.diabetes <- cv.tree(simple.tree)  # 10-fold CV is default.
# Deviance as guide in the pruning process is default.
best <- which.min(cv.diabetes$dev)  # Finds best model.
```

```
plot(cv.diabetes$size, cv.diabetes$dev, type = "b", xlab = "Terminal Nodes", ylab = "Deviance")
points(cv.diabetes$size[best], cv.diabetes$dev[best], col = "red", pch = 20)
```




```
best.tree <- prune.tree(simple.tree, best = cv.diabetes$size[best])
plot(best.tree)
text(best.tree, cex = 0.7, pretty = 0)
```




```
# Calculate classification error for the pruned tree.
yhat.pruned <- predict(best.tree, newdata = d.test, type = "class")
conf.table.pruned <- table(yhat.pruned, d.test$diabetes)
conf.table.pruned
```

```
#>
#> yhat.pruned   0   1
#>           0 148  49
#>           1   7  28
```

```
class.rate.pruned <- 1 - sum(diag(conf.table.pruned))/(sum(conf.table.pruned))
```

The misclassification rate for the pruned tree is 0.2413793. It is apparent that the misclassification rate on the test set is slightly lower for the pruned tree with 3 terminal nodes compared to the unpruned tree. We notice that the left branch only predicts 0, which means that the tree is equivalent to a tree with only two terminal nodes where the left one predicts 0 and the right one predicts 1.

(ii) For the more advanced method, a random forest approach is considered. The number of trees is chosen to be $B = 1000$, since this amount seems to be sufficient for the test error to settle. Furthermore, since $p = 7$, we choose the tuning parameter $m = 3 \approx \sqrt{p}$, because this is recommended in ISLR.

```r
set.seed(1)
B <- 1000
m <- 3
rf.d <- randomForest(diabetes ~ ., d.train, mtry = m, ntree = B, importance = T)

yhat <- predict(rf.d, newdata = d.test, type = "class")
response.test <- d.test$diabetes
d.misclass <- table(yhat, response.test)
d.misclass
```

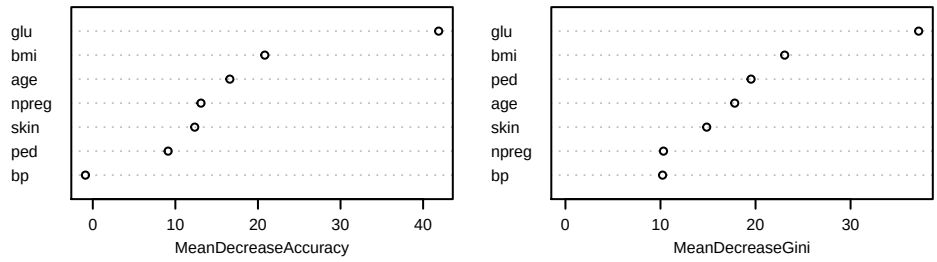
```
#>     response.test
#> yhat   0   1
#>    0 135  34
#>    1  20  43
```

```r
rf.misclass <- 1 - sum(diag(d.misclass))/sum(d.misclass)
rf.misclass
```

```
#> [1] 0.2327586
```

The resulting test misclassification rate is 0.2327586. This is slightly lower than for the simple trees. Next, we find the most influential variables in this classification problem.

```r
varImpPlot(rf.d, main = "", cex = 0.5)
```



Based on both metrics in the plots above, it is apparent that `glu` and `bmi` are the most influential, as expected based on the simple trees from earlier.

Problem 4

a)

TRUE, TRUE, FALSE, TRUE

b)

```r
id <- "1x_E8xnmz9CMHh_tMwIsWP94czPa1Fpsj"  # Google file ID.
d.leukemia <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
    id), header = T)
set.seed(2399)
t.samples <- sample(1:60, 15, replace = F)
d.leukemia$Category <- as.factor(d.leukemia$Category)
```

```
d.leukemia.test <- d.leukemia[t.samples, ]
d.leukemia.train <- d.leukemia[-t.samples, ]
```

(i) A support vector machine (SVM) is more suitable than a logistic regression here, because $p > n$. This means that the parameters in the logistic regression model cannot be estimated. Linear or quadratic discriminant analysis could also be used in this case, in addition to regularized logistic regression variants, i.e. with added penalty term when optimizing (similar to e.g. ridge regression). Classification trees and tree-based methods such as bagging, boosting and random forests can also be used. K-nearest neighbors could also be used, but the curse of dimensionality might deteriorate its performance.

(ii) The paper intends to demonstrate a new method for identifying the subset of genes (predictors) which capture the necessary information in order to classify patients into different groups (response). They introduce the Ensemble SVM-Recursive Feature Elimination (ESVM-RFE), which combines the ensemble and bagging methods used in random forests with SVMs and a backward elimination strategy.

(iii)

```
svcfit <- svm(Category ~ ., data = d.leukemia.train, kernel = "linear", cost = 1,
    scale = T)

# Confusion table from training.
train.yhat <- predict(svcfit, newdata = d.leukemia.train)
train.misclass <- table(predict = train.yhat, truth = d.leukemia.train$Category)
train.misclass
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse          30       0
#>   Relapse               0      15
```

```
# Confusion table from testing.
test.yhat <- predict(svcfit, newdata = d.leukemia.test)
test.misclass <- table(predict = test.yhat, truth = d.leukemia.test$Category)
test.misclass
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse           8       4
#>   Relapse               1       2
```

The misclassification rates are 0 for the training data and 0.333 for the test data.

The training error rate is not surprising, as finding a separating hyperplane when $p >> n$ is easy. Given that `Relapse = 1` and `Non-Relapse = 0`, the support vector classifier has 1 false positive and 4 false negatives on the test set. This means that a false negative is the most common type of error that is seen in the test set. In a clinical setting, false negatives are usually more severe than false positives; i.e. the classifier is very unsuccessful from that perspective.

The low training error and relatively large test error could indicate overfitting. Despite this, the classification method is successful in the sense that it performs slightly better than the null rate, which is attained when always predicting the most frequently occurring class in the training data, which is `Non-Relapse`. The output below substantiates this argument.

```
# Most occurring class is 'Non-Relapse'.
summary(d.leukemia.train$Category)
```

```
#> Non-Relapse     Relapse
#>          30          15
```

```
# Confusion table from predicting most occurring class.
test.misclass.most <- table(predict = rep("Non-Relapse", length(d.leukemia.test$Category)),
    truth = d.leukemia.test$Category)
test.misclass.most
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse           9       6
```

```
# Misclassification error rate from testing.
1 - sum(diag(test.misclass.most))/sum(test.misclass.most)
```

```
#> [1] 0.4
```

(iv) First, the analysis is repeated with $\gamma = 10^{-2}$.

```
svcfit.radial.1 <- svm(Category ~ ., data = d.leukemia.train, kernel = "radial",
    cost = 1, gamma = 0.01, scale = T)

# Confusion table from training.
train.yhat.radial.1 <- predict(svcfit.radial.1, newdata = d.leukemia.train)
train.misclass.radial.1 <- table(predict = train.yhat.radial.1, truth = d.leukemia.train$Category)
train.misclass.radial.1
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse          30       0
#>   Relapse               0      15
```

```
# Confusion table from testing.
test.yhat.radial.1 <- predict(svcfit.radial.1, newdata = d.leukemia.test)
test.misclass.radial.1 <- table(predict = test.yhat.radial.1, truth = d.leukemia.test$Category)
test.misclass.radial.1
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse           9       6
#>   Relapse               0       0
```

The training error rate is 0 in this case. The test misclassification rate is 0.4, i.e. no better than always predicting the most frequently occurring class (which is what the method does for the test set). The low training error and the poor performance on the test set indicate that the model is overfitted to the training data.

Next, the analysis is repeated with $\gamma = 10^{-5}$.

```
svcfit.radial.2 <- svm(Category ~ ., data = d.leukemia.train, kernel = "radial",
    cost = 1, gamma = 1e-05, scale = T)

# Confusion table from training.
train.yhat.radial.2 <- predict(svcfit.radial.2, newdata = d.leukemia.train)
train.misclass.radial.2 <- table(predict = train.yhat.radial.2, truth = d.leukemia.train$Category)
train.misclass.radial.2
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse          30      15
#>   Relapse               0       0
```

```r
# Confusion table from testing.
test.yhat.radial.2 <- predict(svcfit.radial.2, newdata = d.leukemia.test)
test.misclass.radial.2 <- table(predict = test.yhat.radial.2, truth = d.leukemia.test$Category)
test.misclass.radial.2
```

```
#>              truth
#> predict       Non-Relapse Relapse
#>   Non-Relapse           9       6
#>   Relapse               0       0
```

The training error rate is now 0.333, while the test misclassification rate is 0.4, i.e. no better than always predicting the most frequently occurring class (which is what the method does for the test set). The increase in training error can be explained by the change in γ, as a smaller value of this tuning parameter makes the decision boundary smoother, i.e. less variable. In other words, the model becomes more biased.

To compare, we see that the support vector classifier from (iii) outperforms the more flexible support vector machines from (iv) on the test set and also on the training set for $\gamma = 10^{-5}$. This indicates that a linear decision boundary is more appropriate in this situation, compared to the more flexible radial kernel.

c)

$$\begin{aligned} K(X, X') &= (1 + \sum_{j=1}^{2} X_j X'_j)^2 \\ &= (X_1 X'_1 + X_2 X'_2 + 1)^2 \\ &= (X_1 X'_1)^2 + (X_2 X'_2)^2 + 2X_1 X_2 \cdot X'_1 X'_2 + 2X_1 X'_1 + 2X_2 X'_2 + 1 \\ &= \langle h(X), h(X') \rangle, \end{aligned}$$

where

$$h(X) = (X_1^2, X_2^2, \sqrt{2}X_1 X_2, \sqrt{2}X_1, \sqrt{2}X_2, 1)^T.$$

Problem 5

a)

TRUE, FALSE, FALSE, FALSE

b)

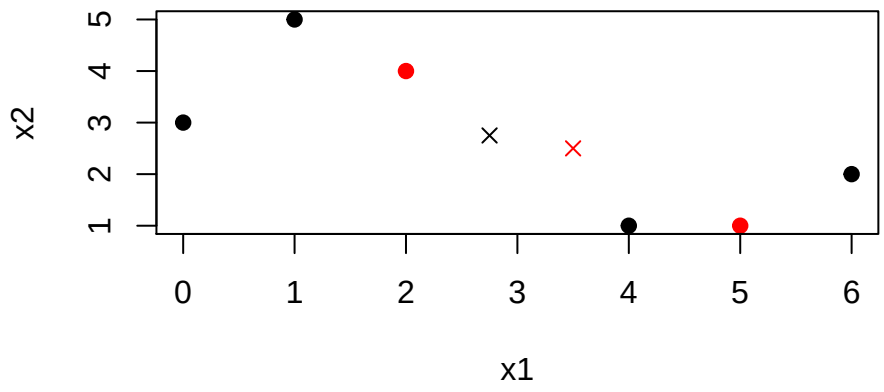
(i)

```r
set.seed(1)
x1 <- c(1, 2, 0, 4, 5, 6)
x2 <- c(5, 4, 3, 1, 1, 2)
K <- 2
cluster <- sample(1:K, length(x1), replace = T)
df <- as.data.frame(cbind(x1, x2, cluster))
df$cluster <- as.factor(df$cluster)
```

(ii)

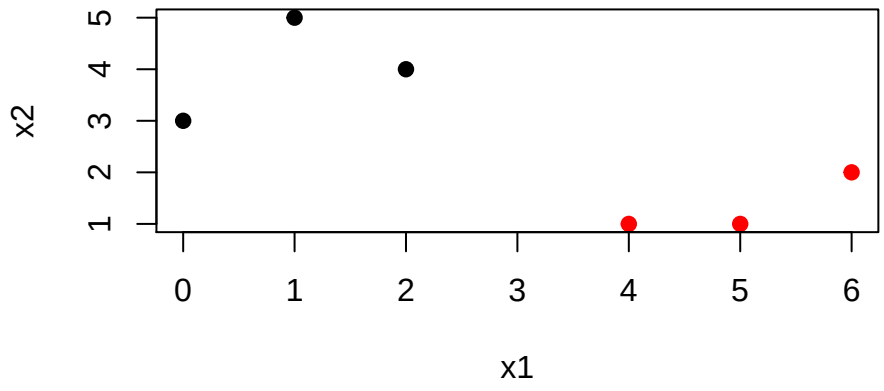
```r
calc_centr <- function(k, df) {
    feats <- df[df$cluster == k, 1:2]
    mean.1 <- mean(feats[, 1])
    mean.2 <- mean(feats[, 2])
    return(c(mean.1, mean.2))
```

```
}

c1 <- calc_centr(1, df)
c2 <- calc_centr(2, df)
plot(df$x1, df$x2, col = df$cluster, pch = 19, xlab = "x1", ylab = "x2")
points(c1[1], c1[2], col = 1, pch = 4)
points(c2[1], c2[2], col = 2, pch = 4)
```



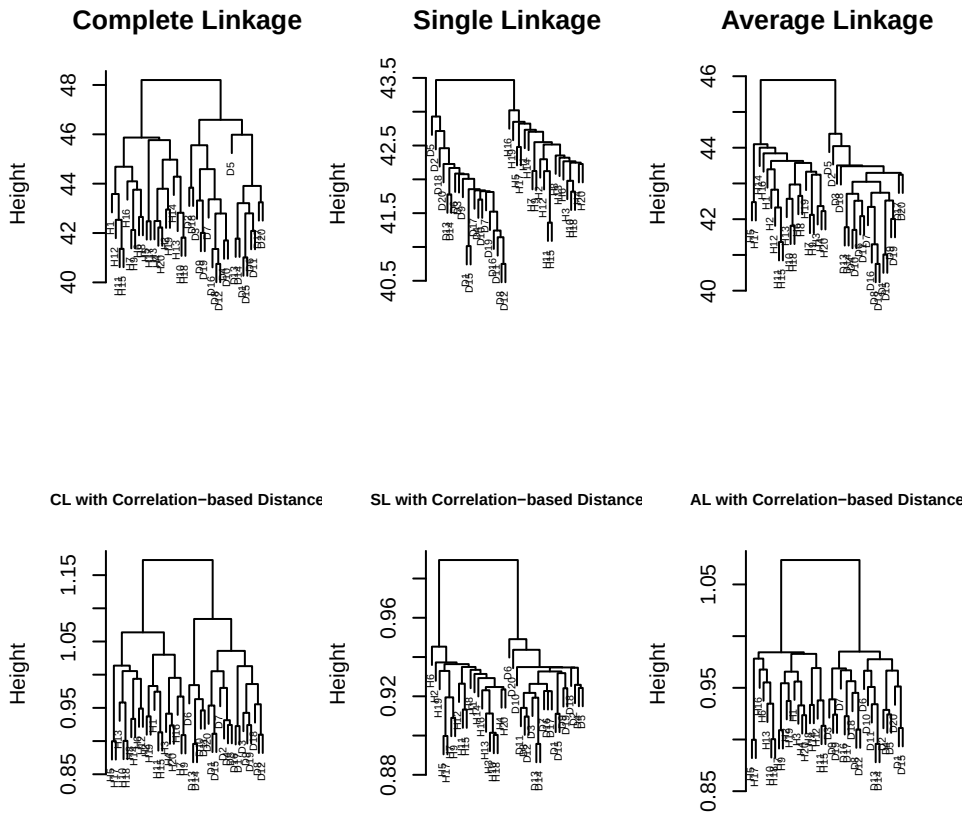
(iii)

```
eucl.dist <- function(x, y) {
    d <- length(x)  # Dimension.
    sum = 0
    for (i in 1:d) {
        sum = sum + (x[i] - y[i])^2
    }
    return(sqrt(sum))
}

reassign <- function(df, centroids) {
    n = nrow(df)
    for (i in 1:n) {
        dist1 <- eucl.dist(df[i, 1:2], centroids[1, ])
        dist2 <- eucl.dist(df[i, 1:2], centroids[2, ])
        if (dist1 > dist2) {
            df[i, 3] <- 2
        } else {
            df[i, 3] <- 1
        }
    }
    return(df)
}

df2 <- reassign(df, matrix(c(c1, c2), nrow = 2, byrow = T))
plot(df2$x1, df2$x2, col = df2$cluster, pch = 19, xlab = "x1", ylab = "x2")
```



c)

```
id <- "1VfVCQvWt121UN39NXZ4aR9Dmsbj-p9OU"  # Google file ID.
GeneData <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
    id), header = F)
colnames(GeneData)[1:20] <- paste(rep("H", 20), c(1:20), sep = "")
colnames(GeneData)[21:40] <- paste(rep("D", 20), c(1:20), sep = "")
row.names(GeneData) <- paste(rep("G", 1000), c(1:1000), sep = "")
GeneData <- t(GeneData)
GeneData <- scale(GeneData)

hc.euc.complete <- hclust(dist(GeneData), method = "complete")
hc.euc.single <- hclust(dist(GeneData), method = "single")
hc.euc.average <- hclust(dist(GeneData), method = "average")

par(mfrow = c(2, 3))
plot(hc.euc.complete, main = "Complete Linkage", xlab = " ", sub = " ", cex = 0.5)
plot(hc.euc.single, main = "Single Linkage", xlab = " ", sub = " ", cex = 0.5)
plot(hc.euc.average, main = "Average Linkage", xlab = " ", sub = " ", cex = 0.5)

dd <- as.dist(1 - cor(t(GeneData)))
hc.cor.complete <- hclust(dd, method = "complete")
hc.cor.single <- hclust(dd, method = "single")
hc.cor.average <- hclust(dd, method = "average")

plot(hc.cor.complete, cex.main = 0.7, main = "CL with Correlation-based Distance",
    xlab = " ", sub = " ", cex = 0.5)
plot(hc.cor.single, cex.main = 0.7, main = "SL with Correlation-based Distance",
    xlab = " ", sub = " ", cex = 0.5)
plot(hc.cor.average, cex.main = 0.7, main = "AL with Correlation-based Distance",
    xlab = " ", sub = " ", cex = 0.5)
```



d)

```
cutree(hc.euc.complete, 2)
```

```
#>  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
#>   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
#>  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
#>   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
all.equal(cutree(hc.euc.single, 2), cutree(hc.euc.average, 2), cutree(hc.cor.complete,
    2), cutree(hc.cor.single, 2), cutree(hc.cor.average, 2))
```

```
#> [1] TRUE
```

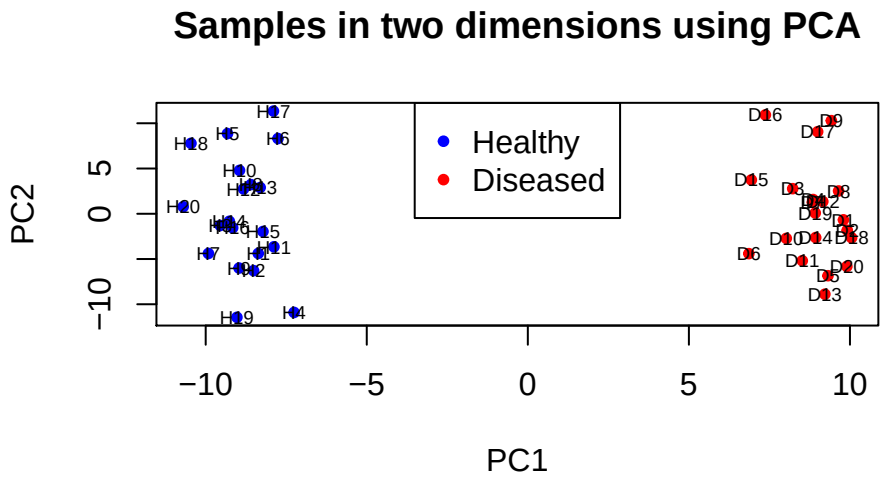
Since we know that the first 20 tissues come from healthy patients and the remaining 20 come from diseased patients, we can see that all the linkage and distance measures lead to perfect classification results on this data.

e)

(i)

```
pr.out <- prcomp(GeneData)
plot(pr.out$x[, 1:2], col = c(rep("blue", 20), rep("red", 20)), pch = 20, main = "Samples in two dimens
```

```
text(pr.out$x[, "PC1"], pr.out$x[, "PC2"], rownames(GeneData), cex = 0.6)
legend("top", legend = c("Healthy", "Diseased"), col = c("blue", "red"), pch = 20)
```



(ii)

```
pr.var <- pr.out$sdev^2
pve <- sum(pr.var[1:5])/sum(pr.var)
pve
```

```
#> [1] 0.2109659
```

The proportion of variance explained by the first 5 PCs is 21.1%.

f)

The plot in e) shows that the two groups are perfectly separated along the axis of the first principal component (PC1), which contains the highest variability in the data among all the principal components. Hence, to determine which genes vary the most across the two groups, the loadings of the first principle component are examined. The genes with the highest loadings (rotations) contribute the most to the variability. Hence, we infer that the genes displayed in the output below vary the most across the two groups.

```
sort(abs(pr.out$rotation[, "PC1"]), decreasing = T)[1:10]
```

```
#>       G502       G589       G565       G590       G600       G551       G593
#> 0.09485044 0.09449766 0.09183823 0.09173169 0.09167322 0.08768360 0.08758616
#>       G538       G584       G509
#> 0.08745400 0.08690858 0.08661015
```