Midterm for Statistical Learning

Laurits Wieslander Lyngbaek - e1324720

2024-03-18

Set seed:

```
set.seed(4)
```

Question 1:

Consider a dataset $\{(xi, yi) : 1 \leq i \leq n\}$ with predictor variables xi = (xi1, xi2) and responses yi. Assume the dataset is generated from the model

$$(1);\ y_i = x_{i1}x_{i2} + \epsilon_i$$

where ϵ_i are independent with $\mu = 0$ and variance $= \sigma^2$. Boosting is applied to the dataset with d splits.

Subquestion 1.a)

Describe the boosting algorithm for regression trees.

Regression tree boosting is an additive model of the base learner *regression trees*. This explanation will assume that the algorithm for constructing a regression tree is known, and focus on explaining the boosting element. The boosting algorithm is also applicable to other slow base learners, that is learners with a tendency to underfit rather than overfit. A regression tree is ensured to be a weak learner by limiting the amount of terminal notes. This tree is fitted on the residuals of the dataset. The boosting method then fits a tree to the residuals, and updates the residuals before fitting a new tree. The residuals are updated using the function $r_i \leftarrow r_i + \lambda\hat{f}^b(x)$, where λ is the learning rate reducing overfitting. The tree is added to the prediction model $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda\hat{f}^b(x)$ The algorithm ends when a the requested amount of trees has been grown. The predicted value of new data is then $\hat{f}(x) = \sum_{b=1}^{B} \lambda\hat{f}^b(x)$.

For more advanced boosting algorithms the λ can be varied depending on the goodness of fit of the tree.

Subquestion 1.b)

Is the selection of d = 1 appropriate? How about the selection of d = 2 and d = 3? Explain.
Historically, the boosting algorithm was seen as tool for combining weak trees an additive manner, the trees would therefore just be trimmed as if they were the last tree. This makes the tree the primary workhorse of the statistical learning, and the highly effective boosting algorithm would have a secondary role with degraded performance and an increase in computation cost. It has been found that by keeping the trees small the boosting algorithm converges at a way better prediction when the amount of trees are sufficient. It is recommended to use stumps as your tree as a thumb-rule, if you want to avoid your prior knowledge about causality. However, as we know the causality of the dataset we can make a better educated guess. According to Elements of Statistical Learning (2nd edition) Hastie, Tibshirani and Friedman (2009), the the value chosen for d should reflect the level of dominant interactions. For our prediction variable we know that Y is defined by a two-variable interaction effect, and our trees would therefore benefit from two splits, defined by a d = 2. That being said, to explicitly answer the question, d would be appropriate in the range 1, 2 . . . , 8 according to the ESLII for real-world problems. For this problem a d = 2 would be preferable over 1 and 3,

but the optimization would be minimal before convergence. It would still be a good idea to do a three-way split of your data and optimize the hyperparameters to estimate the best simulated value.

Subquestion 1.c)

Design and implement a Monte Carlo study to investigate your response to (b), with the following components: (i) Generate n observations from model (1).

```
pacman::p_load(tidyverse)
# Simulate the Dataset
n <- 500
x_1 <- rnorm(n, mean = 3, sd = 1)
x_2 <- rnorm(n, mean = 2, sd = 1)
epsilon <- rnorm(n, mean = 0, sd = 1)
y <- x_1*x_2+epsilon
sim_data <- tibble(x_1, x_2, y)
```

(ii) Create a training set consisting of the rest n/2 observations, and a test set consisting of the remaining observations.

```
training <- sample_frac(sim_data, size = 0.5)
test_data <- anti_join(sim_data, training, by = join_by(x_1, x_2, y))
```

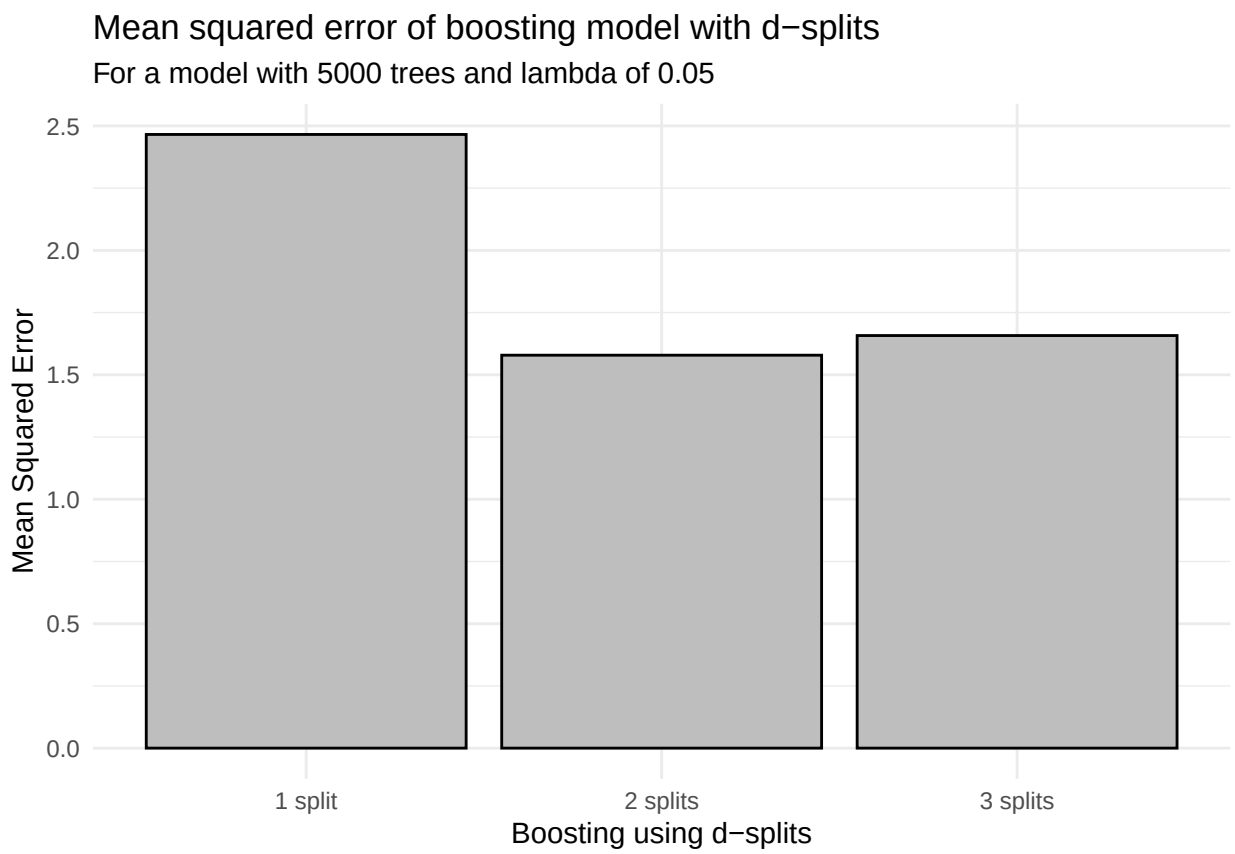
(iii) Perform boosting on the training set with d = 1, 2 or 3 splits (while keeping all the other parameters fixed).

```
pacman::p_load(gbm)
#"gaussian" = minimize(squared error)
# Interaction depth: Integer specifying the maximum depth of each tree (i.e., the highest level of vari

boost_depth_1 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 1, shrinkage = 0.05)
boost_depth_2 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 2, shrinkage = 0.05)
boost_depth_3 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 3,shrinkage = 0.05)
boost_depth_4 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 4,shrinkage = 0.05)
boost_depth_5 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 5,shrinkage = 0.05)
boost_depth_6 <- gbm::gbm(y~., data = training, distribution = "gaussian", n.trees = 5000,
 interaction.depth = 6,shrinkage = 0.05)
```

(iv) Produce a plot with d on the x-axis and corresponding test set MSE on the y-axis. Please append a printout of your R code to the solution.

```
yhat.boost_1 <- predict.gbm(object = boost_depth_1,
                            newdata = test_data,
                            n.trees = 5000)
yhat.boost_2 <- predict.gbm(object = boost_depth_2,
                            newdata = test_data,
                            n.trees = 5000)
yhat.boost_3 <- predict.gbm(object = boost_depth_3,
                            newdata = test_data,
                            n.trees = 5000)

MSE_boost_depth_1 <- mean((yhat.boost_1-test_data$y)^2)
```

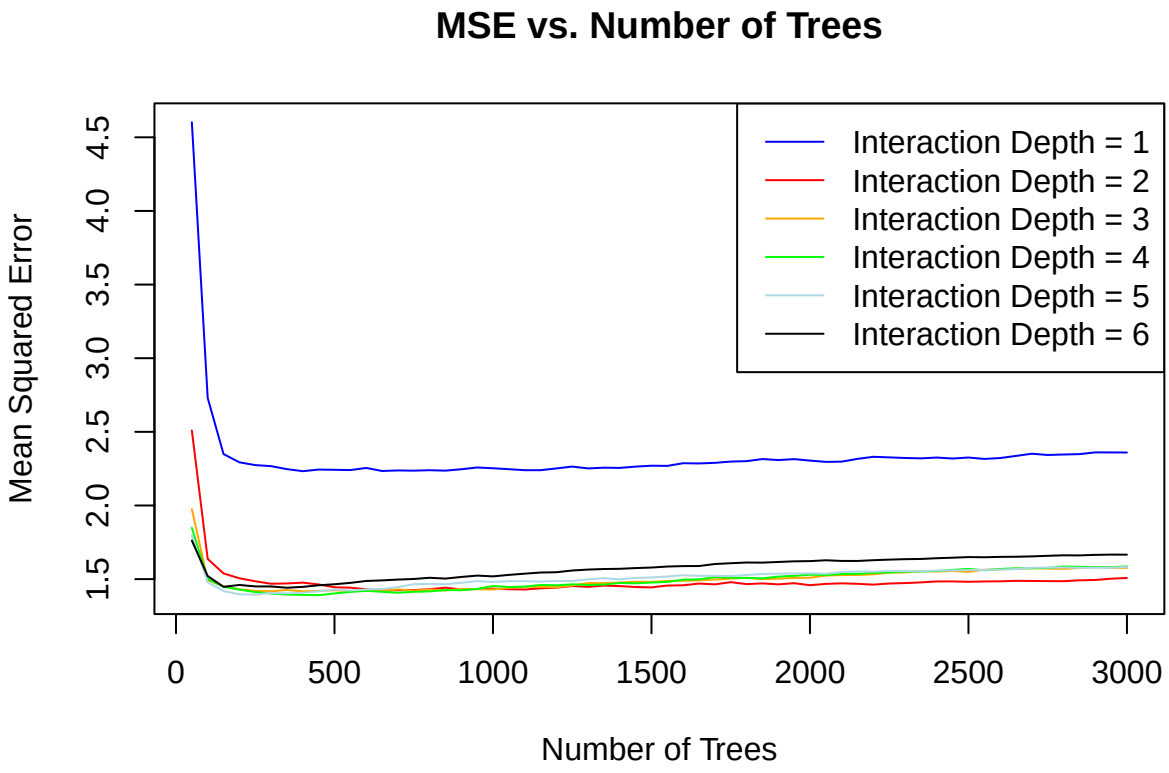
```r
MSE_boost_depth_2 <- mean((yhat.boost_2-test_data$y)^2)
MSE_boost_depth_3 <- mean((yhat.boost_3-test_data$y)^2)



data.frame(
  x = c(1, 2, 3),  # X-axis values
  y = c(MSE_boost_depth_1, MSE_boost_depth_2,MSE_boost_depth_3)  # Y-axis values
  ) %>%
  ggplot(aes(x = factor(x), y = y)) +
  geom_bar(stat = "identity", fill = "grey", color = "black") +
  labs(x = "Boosting using d-splits",
       y = "Mean Squared Error",
       title = "Mean squared error of boosting model with d-splits",
       subtitle = "For a model with 5000 trees and lambda of 0.05") +
  scale_x_discrete(labels = c("1 split", "2 splits", "3 splits")) +  # Set x-axis labels
  theme_minimal()
```




```r
# Initialize empty vectors to store MSE values
trees <- seq(from = 50, to = 3000, by = 50)
mse_boost_1 <- numeric(length(trees))
mse_boost_2 <- numeric(length(trees))
mse_boost_3 <- numeric(length(trees))
mse_boost_4 <- numeric(length(trees))
mse_boost_5 <- numeric(length(trees))
mse_boost_6 <- numeric(length(trees))
```

```r
# Iterate over different numbers of trees
for (i in 1:length(trees)) {
  # Current number of trees
  n_trees <- trees[i]

  # Predictions for boost_depth_1 model
  yhat.boost_1 <- predict.gbm(object = boost_depth_1,
                              newdata = test_data,
                              n.trees = n_trees)
  # Predictions for boost_depth_2 model
  yhat.boost_2 <- predict.gbm(object = boost_depth_2,
                              newdata = test_data,
                              n.trees = n_trees)
    # Predictions for boost_depth_3 model
  yhat.boost_3 <- predict.gbm(object = boost_depth_3,
                              newdata = test_data,
                              n.trees = n_trees)
      # Predictions for boost_depth_4 model
  yhat.boost_4 <- predict.gbm(object = boost_depth_4,
                              newdata = test_data,
                              n.trees = n_trees)
       # Predictions for boost_depth_5 model
  yhat.boost_5 <- predict.gbm(object = boost_depth_5,
                              newdata = test_data,
                              n.trees = n_trees)
       # Predictions for boost_depth_6 model
  yhat.boost_6 <- predict.gbm(object = boost_depth_6,
                              newdata = test_data,
                              n.trees = n_trees)


  # Calculate MSE for boost_depth_1 model
  mse_boost_1[i] <- mean((yhat.boost_1 - test_data$y)^2)

  # Calculate MSE for boost_depth_2 model
  mse_boost_2[i] <- mean((yhat.boost_2 - test_data$y)^2)

    # Calculate MSE for boost_depth_3 model
  mse_boost_3[i] <- mean((yhat.boost_3 - test_data$y)^2)

      # Calculate MSE for boost_depth_4 model
  mse_boost_4[i] <- mean((yhat.boost_4 - test_data$y)^2)
       # Calculate MSE for boost_depth_5 model
  mse_boost_5[i] <- mean((yhat.boost_5 - test_data$y)^2)
       # Calculate MSE for boost_depth_6 model
  mse_boost_6[i] <- mean((yhat.boost_6 - test_data$y)^2)
}

# Plot MSE vs. number of trees
plot(trees, mse_boost_1, type = "l", col = "blue", xlab = "Number of Trees", ylab = "Mean Squared Error
     ylim = c(
       min(min(mse_boost_1), min(mse_boost_2), min(mse_boost_3), min(mse_boost_4), min(mse_boost_5), mi
       max(max(mse_boost_1), max(mse_boost_2),max(mse_boost_3),max(mse_boost_4),max(mse_boost_5),max(ms
```

```r
lines(trees, mse_boost_2, col = "red")
lines(trees, mse_boost_3, col = "orange")
lines(trees, mse_boost_4, col = "green")
lines(trees, mse_boost_5, col = "lightblue")
lines(trees, mse_boost_6, col = "black")
legend("topright", legend = c("Interaction Depth = 1", "Interaction Depth = 2", "Interaction Depth = 3"
```



Subquestion 1.d)

Do you expect the training MSE to decrease as the number of trees B increases? You may assume that all the other parameters remain the same as B increases.

I expect the MSE to roughly stay the same, but increase if anything. As trees increase a boosting model have a tendency to slowly overfit, however this can be regulated by the learning rate and the amount of splits. As we are able to directly sample the 'real distribution' with no bias, this also means that increasing the N would make an increase in trees decrease the error, as the there would be no effective difference between the training data, the real distribution and test-data. An increase of trees in this circumstance would allow the model to more closely fit the real interaction effect.

To recap; an increase in trees generally lead to overfitting and thereby an increased MSE, however it depends on the parameters of the model.

Question 2:

TheCarseats dataset is available in the library ISLR. Treat the feature 'Sales' as a quantitative response and the other features as predictor variables. Answer the following questions and append a printout of your R

code to the solution.

Subquestion 2.a)

Split the dataset equally into training and test subsets.

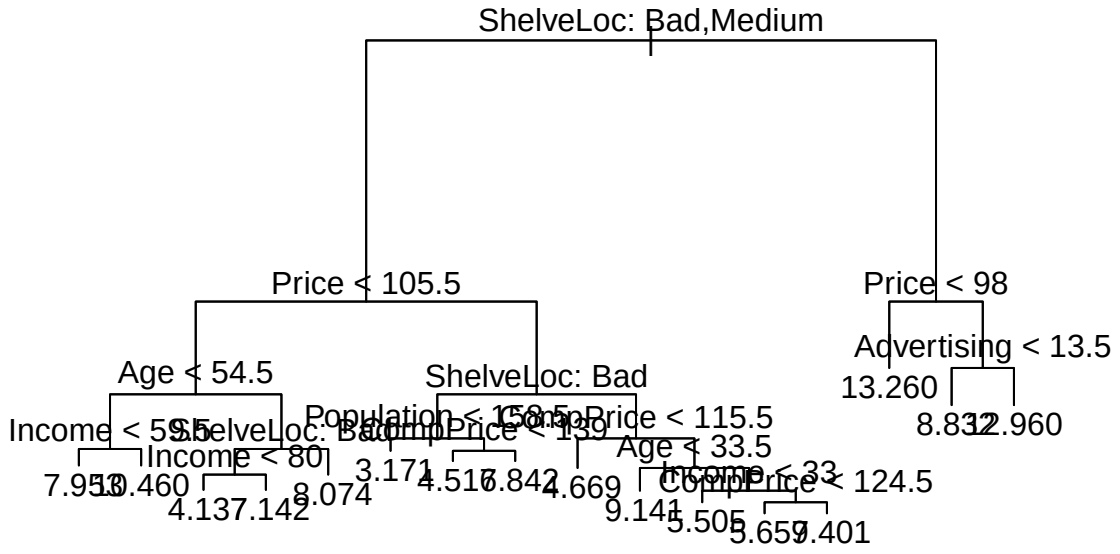
```
set.seed(7933)
# Load data
pacman::p_load(ISLR, tree)
Carseats <- tibble(ISLR::Carseats)
samples=sample(nrow(Carseats), 0.5*nrow(Carseats))
training = Carseats[samples, ]
testing = Carseats[-samples,]
```

Fit a regression tree with default parameter values to the training set.

```
tree.carseats <-tree(Sales ~ ., training)
```

Plot the tree and interpret the results.

```
plot(tree.carseats)
text(tree.carseats, pretty = FALSE)
```



Interpretation of tree: To spare everyone the trouble, Im only gonna interpret the first 2 layers of splits. First the tree is split based on Shelve location. This means that shelve location is the factor that can be split such that the average sales has the least residuals. The second predicting variable is Price, no matter what the shelve location is. If the shelve location is Bad or Medium, then the predicted sales are determined by whether the price is above or below 98, if the location is good it is determined by the price being above or below 105.5 These splits are continued until a end leaf is reached. Each end leaf has a number, this number

is the mean of all the training data described by this node.

What test MSE do you obtain?

```
tree.pred <-predict(tree.carseats, testing)

MSE <- mean((tree.pred - testing$Sales)^2)
print(paste0("The trees MSE is: ", round(MSE, 3)))
```

```
## [1] "The trees MSE is: 4.528"
```

Subquestion 2.b)

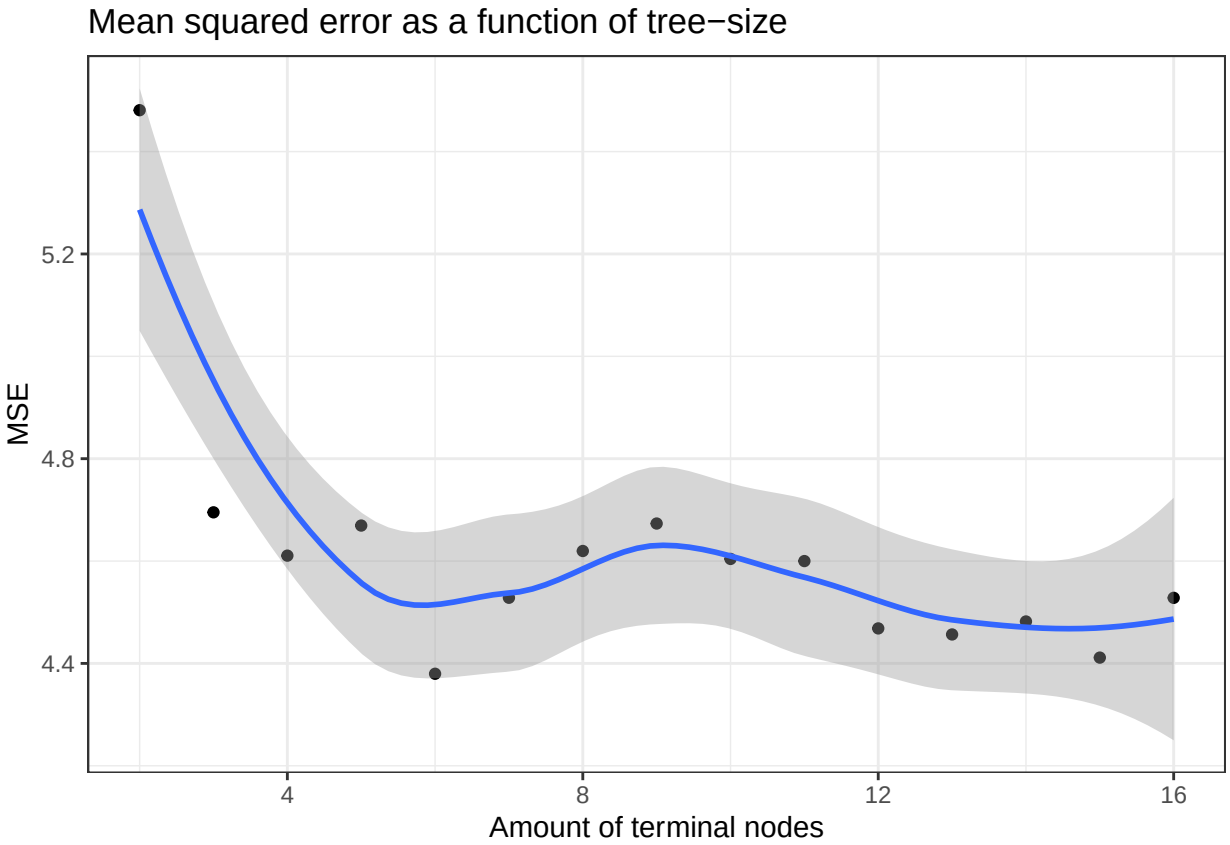
Prune the tree from (a) down to k terminal nodes (leaves) for all possible values of k ≥ 2.

```
max_final_nodes <- summary(tree.carseats)$size
leaves_vec <- c()
MSE_vec <- c()


for (i in seq(2:max_final_nodes)+1){
  pruned_tree <- prune.tree(tree.carseats, best = i)
  tree.pred <- predict(pruned_tree, testing)
  MSE <- mean((tree.pred - testing$Sales)^2)


  leaves_vec <- c(leaves_vec, i)
  MSE_vec <- c(MSE_vec, MSE)
}
```

Plot the test MSE against the number of terminal nodes.

```
pacman::p_load(ggplot2)
data <- data.frame(Amount_of_leaf_nodes = leaves_vec, MSE = MSE_vec)

ggplot(data, aes(x = Amount_of_leaf_nodes, y = MSE)) +
  geom_point() +                              # Add points
  geom_smooth() +    # Add linear regression line
  labs(x = "Amount of terminal nodes",            # x-axis label
       y = "MSE",                             # y-axis label
       title = "Mean squared error as a function of tree-size")+ # plot title
  theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Provide some comments on the plot: As the amount of terminal nodes increases the average MSE for predicting the test-set decreases. This makes intuitive sense as the increase in nodes decreases the variance of each terminal node. However, problems arise as the amount of nodes become to big, as the bias of the model begin to counteract the decrease in variance. It may be beneficial to use an ensemble learning technique to avoid bias errors.

Subquestion 2.c)

Use bagging to analyze this dataset.

```
pacman::p_load(randomForest)
set.seed(1)
bag.carseats <- randomForest(Sales ~ .,
                data = Carseats,
                subset = samples,
                mtry = length(Carseats)-1,
                importance = TRUE)
```

What is the test MSE? *numeric investigation*:

```
paste0("The best MSE of the testset was: ", round(min(bag.carseats$mse),3))
```

```
## [1] "The best MSE of the testset was: 2.786"
```

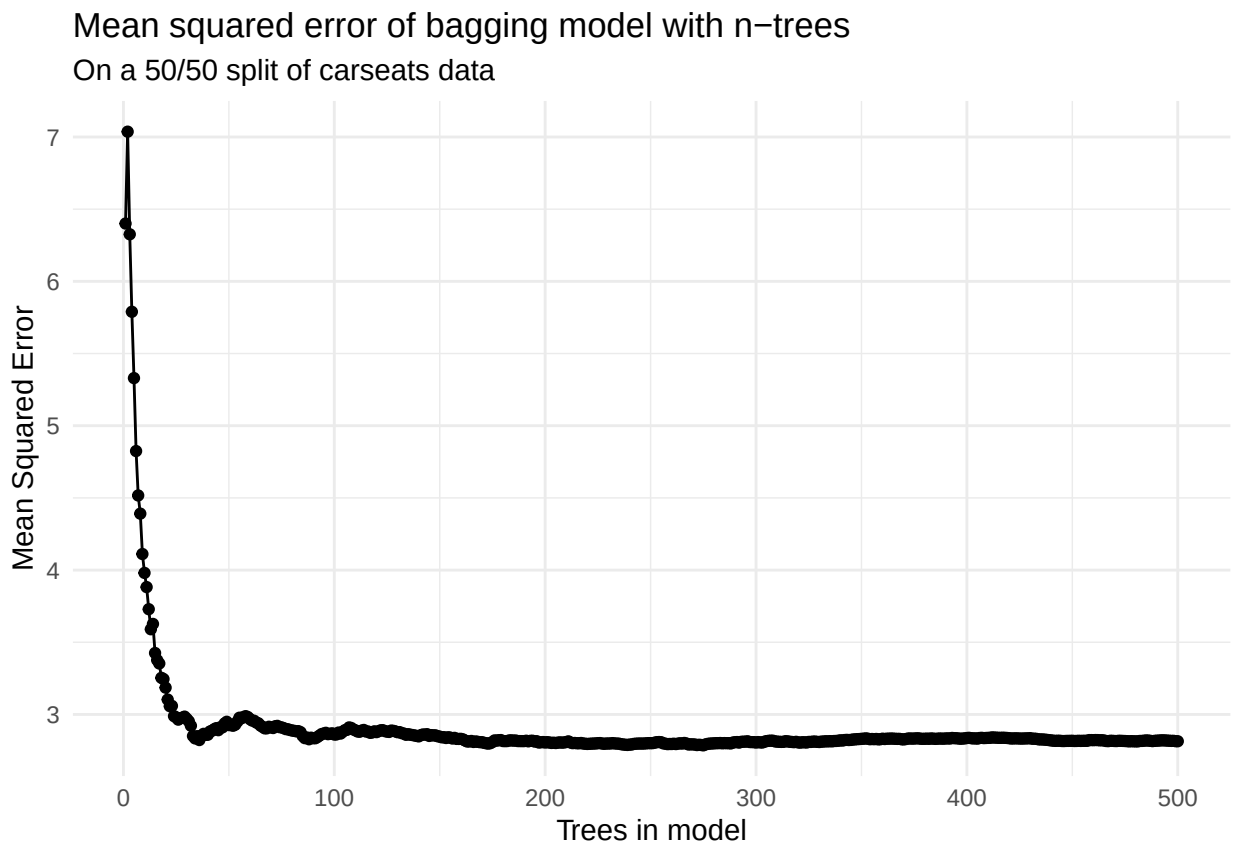
```
paste0("The mean MSE of the testset was: ", round(mean(bag.carseats$mse),3))
```

```
## [1] "The mean MSE of the testset was: 2.899"
```

plot of mse as a function of trees

```r
data_frame(x = seq(1:500), y = bag.carseats[["mse"]]) %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line()+
    labs(x = "Trees in model",
       y = "Mean Squared Error",
       title = "Mean squared error of bagging model with n-trees",
       subtitle = "On a 50/50 split of carseats data") +
  theme_minimal()
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Use the importance function to determine which variables are most important.

```r
importance(bag.carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice   28.332858    183.533311
## Income      10.848375    102.508184
## Advertising 20.420098    125.242991
## Population   1.619847     58.523124
## Price       58.886432    513.514500
## ShelveLoc   58.465132    598.376045
```

```
## Age         14.331039    137.294410
## Education   -1.366582     35.436806
## Urban       -2.469788      8.024748
## US           1.317710      4.465707
```

Across the bagging samples the most important variables are Price and ShelveLoc, these are equally responsible for decreasing prediction error The third most important variable is CompPrice, followed by Advertising.

Subquestion 2.d)

Use random forests to analyze this dataset. Do not specify mtry. What mtry is used by randomForest?

```
set.seed(1)
rF.carseats <- randomForest(Sales ~ .,
                data = Carseats,
                subset = samples,
                importance = TRUE)
print(paste0("Ff mrty is not specified, then randomForsest uses 'variables/3' --> floor(10/3) --> ", fl
```

```
## [1] "Ff mrty is not specified, then randomForsest uses 'variables/3' --> floor(10/3) --> 3"
```

```
rF.carseats
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, importance = TRUE,      subset = samples)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 3.349643
##                     % Var explained: 62.9
```

```
print("3 is correctly used as the mrty for this random forest")
```

```
## [1] "3 is correctly used as the mrty for this random forest"
```

What test MSE do you obtain?

```
print(paste("Mean of squared residuals:",
            round(tail(rF.carseats$mse, n = 1),5)))
```

```
## [1] "Mean of squared residuals: 3.34964"
```

Use the importance function to determine which variables are most important.

```
importance(rF.carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice   16.129774     182.25889
## Income       4.768071     143.93545
## Advertising 12.368435     143.64364
## Population   0.745901     112.65986
## Price       36.300230     434.33316
## ShelveLoc   38.063038     409.56303
## Age         11.931789     187.52470
## Education   -1.531076      66.24015
## Urban       -1.876504      15.82821
## US           1.948178      12.82304
```

The same variables are important for the random forest as in the bagging: The most important variables are Price and ShelveLoc, these are equally responsible for decrease in prediction error. The third most important variable is CompPrice, followed by Advertising.

What can you say about the importance of the variables in bagging and random forest? It makes sense that the important variables are important regardless of the algorithm, and only the importance weights differ relatively between the algorithms. For this seed the relative importance of the runner-up variables increase in the random forest. This could be an effect of the limited number of variables randomly sampled as candidates at each split, as the correlated information between comp-price and and price might not 'fight' about right to be expressed. That is the causal effect of a shared confounding variable will only be accessible to the algorithm in one variable for most trees.

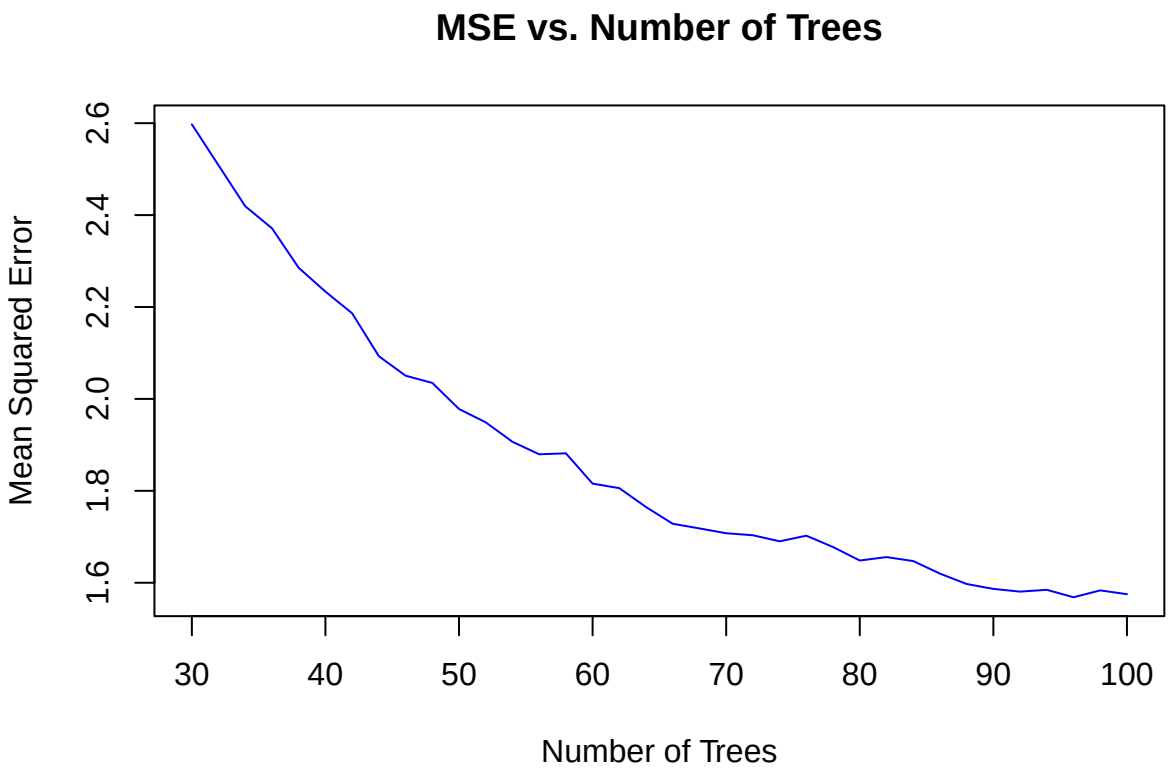
Subquestion 2.e)

Find the test MSE for boosting with d = 2 splits. You can use the default shrinking parameter and number of trees.

```
set.seed(1)
boost.carseats <- gbm::gbm(Sales ~ ., data = training, distribution = "gaussian", interaction.depth = 2
yhat.boost.carseats <- predict.gbm(object = boost.carseats,
                           newdata = testing,
                           n.trees = boost.carseats$n.trees)
MSE_Boost <- mean((yhat.boost.carseats-testing$Sales)^2)
print(paste("Mean Squared Error for boosting with 2 splits: ", round(MSE_Boost, 3)))
```

```
## [1] "Mean Squared Error for boosting with 2 splits:  1.575"
```

```
trees <- seq(from = 30, to = 100, by = 2)
mse_boost_vec <- numeric(length(trees))
for (i in 1:length(trees)) {
  # Current number of trees
  n_trees <- trees[i]

  # Predictions for model
  yhat.boost <- predict.gbm(object = boost.carseats,
                            newdata = testing,
                            n.trees = n_trees)


 # Calculate MSE for boost_depth_1 model
  mse_boost_vec[i] <- mean((yhat.boost - testing$Sales)^2)

}

# Plot MSE vs. number of trees
plot(trees, mse_boost_vec, type = "l", col = "blue", xlab = "Number of Trees", ylab = "Mean Squared Err
```



Provide some comments. The boosting regression trees algorithm outperforms the other predicitve methods by fair margin of minimum 1 less MSE. This could be prescribed the boostings algorithms ability to avoid overfitting by limiting the size of the tree. This is not unique to boosting, but boosting still outperforms random forest, that performed quite poorly on this collection of seeds. This difference should be described to the boostings algorithms ability to fit trees to the updated residuals. It seems that boosting is able to keep its low variance without getting a high bias, but the algorithm looses some interpretability features of methods like the single regression tree.

Question 3:

Consider the dataset $(x_i, y_i) : 1 \leq i \leq 100$ in Q3.csv, which has been generated from the model

$$y_i = m(x_i) + \epsilon_i$$

where ϵ_i are independent with mean 0 and variance σ^2. To estimate m(x), we approximate it by the cubic spline with 3 knots,

$$m(x) \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - 0.3)_+^3 + \beta_5 (x - 0.5)_+^3 + \beta_6 (x - 0.7)_+^3.$$

Answer the following questions and append a printout of your R code to the solution.

Subquestion 3.a)

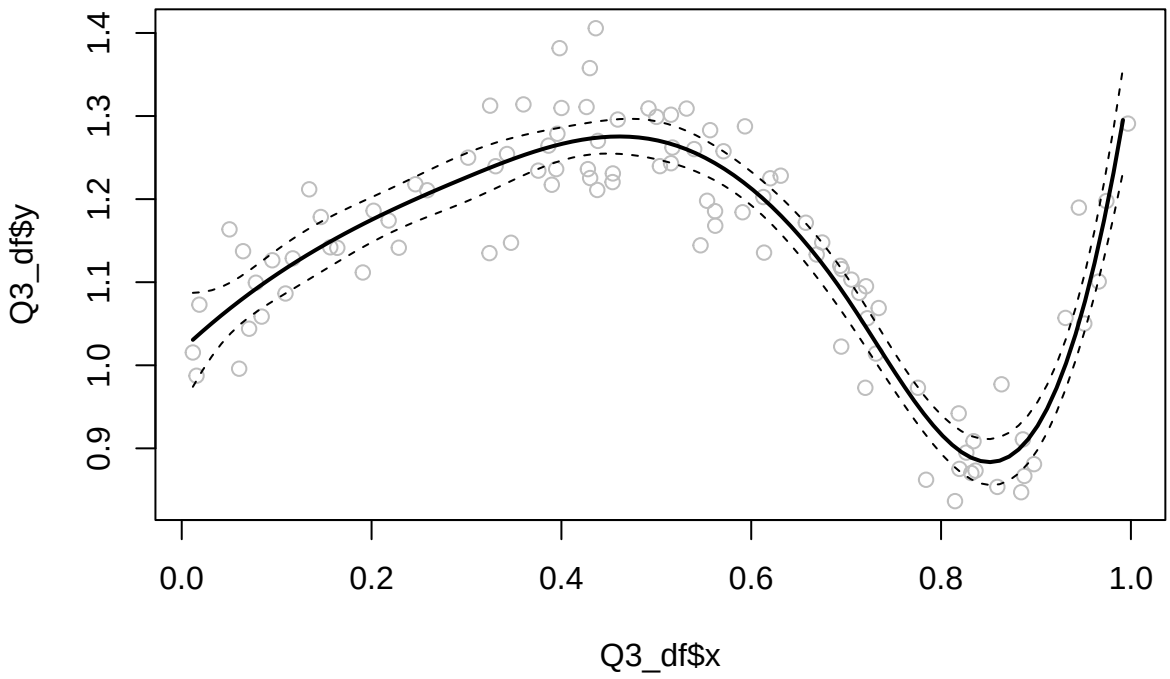
Fit the model and write down the estimated function $\hat{m}(x)$.

```r
# Load data:
Q3_df <- read_csv(file = "Q3.csv")
```

```
## Rows: 100 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): y, x
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
pacman::p_load(splines)

#Fit model
fit <-lm(y ~ bs(x, knots = c(0.3, 0.5, 0.7)), data = Q3_df)
```

```r
#x-grid
xlims <-range(Q3_df$x)
x.grid <- seq(from = xlims[1], to = xlims[2], by = 0.01)
pred <-predict(fit, newdata = list(x = x.grid), se = T)


#
plot(Q3_df$x, Q3_df$y, col = "gray")
lines(x.grid, pred$fit, lwd = 2)
lines(x.grid, pred$fit + 2 * pred$se, lty = "dashed")
lines(x.grid, pred$fit- 2 * pred$se, lty = "dashed")
```



```r
# Output parameters
print("Parameters from fitted model: (beta0 -> beta6)")
```

```
## [1] "Parameters from fitted model: (beta0 -> beta6)"
```

```r
as.numeric(fit$coefficients)
```

```
## [1]  1.03064427  0.09894121  0.18555769  0.28500204  0.07403421 -0.39486610
## [7]  0.30020442
```

$$\hat{m}(x) = 1.031 + 0.099x + 0.186x^2 + 0.285x^3 + 0.074(x-0.3)_+^3 - 0.395(x-0.5)_+^3 + 0.300(x-0.)7_+^3.$$

Subquestion 3.b)

To investigate whether m(x) indeed varies with x, test the following hypothesis at signicance level 0.05,

$$H_0 : m(x) \equiv \text{a constant}$$

$$H_0 : \beta_1 + \beta_2 + ... + \beta_6 = 0$$

```r
sum_fit <- summary(fit)
sum_fit
```

```
##
## Call:
## lm(formula = y ~ bs(x, knots = c(0.3, 0.5, 0.7)), data = Q3_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.108094 -0.038900  0.002729  0.030799  0.139090
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      1.03064    0.02828  36.450  < 2e-16 ***
## bs(x, knots = c(0.3, 0.5, 0.7))1  0.09894    0.05877   1.683   0.0956 .
## bs(x, knots = c(0.3, 0.5, 0.7))2  0.18556    0.04093   4.533 1.73e-05 ***
## bs(x, knots = c(0.3, 0.5, 0.7))3  0.28500    0.04131   6.899 6.27e-10 ***
## bs(x, knots = c(0.3, 0.5, 0.7))4  0.07403    0.04271   1.733   0.0863 .
## bs(x, knots = c(0.3, 0.5, 0.7))5 -0.39487    0.04942  -7.990 3.56e-12 ***
## bs(x, knots = c(0.3, 0.5, 0.7))6  0.30020    0.04380   6.854 7.72e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05194 on 93 degrees of freedom
## Multiple R-squared:  0.8673, Adjusted R-squared:  0.8587
## F-statistic: 101.3 on 6 and 93 DF,  p-value: < 2.2e-16
```

```r
print("P-value of H0:")
```

```
## [1] "P-value of H0:"
```

```r
print(paste0(round(prod(sum_fit[["coefficients"]][2:7,4]),40), "*** << 0.001"))
```

```
## [1] "2.454e-37*** << 0.001"
```

This can also be seen at a glance at the summary table, as all parameters except β_1 and β_4 are below p-vale 0.05. The H_0 is therefore rejectable.

Subquestion 3.c)

For the sequence of x values $0, 0.1, 0.2, \cdots, 0.8, 0.9, 1$, predict the expected y values and obtain the corresponding 95% condence intervals.

```
#x-grid

x.grid <- seq(from = 0.1, to = 1, by = 0.1)
pred <-predict(fit, newdata = list(x = x.grid), se = T)
```

```
## Warning in bs(x, degree = 3L, knots = c(0.3, 0.5, 0.7), Boundary.knots =
## c(0.0117, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases
```

```
data_frame(x = x.grid,
           lower_95 = pred$fit- 2 * pred$se,
           y.hat = pred$fit,
           higher_95 = pred$fit+ 2 * pred$se)
```

```
## # A tibble: 10 x 4
##        x lower_95 y.hat higher_95
##    <dbl>    <dbl> <dbl>     <dbl>
##  1   0.1    1.08  1.11      1.14
##  2   0.2    1.15  1.18      1.20
##  3   0.3    1.20  1.23      1.26
##  4   0.4    1.25  1.27      1.29
##  5   0.5    1.25  1.27      1.29
##  6   0.6    1.19  1.21      1.23
##  7   0.7    1.06  1.08      1.11
##  8   0.8    0.894 0.917     0.940
##  9   0.9    0.895 0.924     0.953
## 10   1      1.28  1.35      1.42
```

Subquestion 3.d)

Suppose m(x) is linear in the boundary regions $x < 0.3$ and $x > 0.7$. Show that this implies the constraints $\beta_2 = 0, \beta_3 = 0, \beta_4 + \beta_5 + \beta_6 = 0$ and $0.3\beta_4 + 0.5\beta_5 + 0.7\beta_6 = 0$.

ANSWER: This implies that our cubic spline becomes a 'Natrual cubic spline'.

The spline function is constrained to be linear when $X <$ smallest knot c_1 by $f_0^{''}(c_1) = 0$

The spline function is constrained to be linear when $X >$ largest knot c_n by $f_n^{''}(c_n) = 0$

For $f_0^{''}(c_1)$ we have $f_0(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 \cdot 0 + \beta_5 \cdot 0 + \beta_6 \cdot 0$-> $f_0^{'}(x) = \beta_1 + 2\beta_2 x + 3\beta_3 x^2$ -> $f_0^{''}(x) = 2\beta_2 + 6\beta_3 x$

$f_0^{''}(c_1) = 0$ shoud be true when we assume $\beta_2 = 0, \beta_3 = 0, \beta_4 + \beta_5 + \beta_6 = 0$;

$$f_0^{''}(0) = 2\beta_2 + 6\beta_3 x \rightarrow f_0^{''}(0) = 2 \cdot 0 + 6 \cdot 0x = 0$$

The first implication holds true, as $\beta_4 + \beta_5 + \beta_6 = 0$ is the computational trick to set the respective $0 * (x - \xi)^3 = 0$

For the second implication we just need to see when

$$f_n^{''}(x > 0.7) = 0 \rightarrow \frac{d^2}{dx^2}(\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - 0.3)^3 + \beta_5 (x - 0.5)^3 + \beta_6 (x - 0.7)^3) = 0$$

For the second implication we just need to see when

$$2\beta_2 + 6\beta_3 x + 6\beta_4(x-0.3) + 6\beta_5(x-0.5) + 6\beta_6(x-0.7) = 0$$

We know that $2\beta_2 + 6\beta_3 x = 0$, so:

$$0 + 6\beta_4(x-0.3) + 6\beta_5(x-0.5) + 6\beta_6(x-0.7) = 0$$

$$\beta_4(x-0.3) + \beta_5(x-0.5) + \beta_6(x-0.7) = 0$$

$$\beta_4 x - \beta_4 0.3 + \beta_5 x - \beta_5 0.5 + \beta_6 x - \beta_6 0.7 = 0$$

$$\beta_4 x + \beta_5 x + \beta_6 x = \beta_4 0.3 + \beta_5 0.5 + \beta_6 0.7$$

For any x>0.7 any non-zero beta parameter will always leave the left side of the equation bigger.

$$\beta_4 0.3 + \beta_5 0.5 + \beta_6 0.7 = 0 \rightarrow 0 \cdot 0.3 + 0 \cdot 0.5 + 0 \cdot 0.7 = 0$$

Subquestion 3.e)

How many degrees of freedom does the constrained cubic spline in (d) have? For a natural cubic spline with 3 interior knots, or 'the constrained cubic spline in (d)', there exists a total of five knots, including the two boundary knots. This can be expressed as a natrual cubic spline with K interoir knots has $K + 2 + 4$ degrees of freedom.

This specific costrained cubic spline has 4 total constraints: 1. First order derivative is linear at splines 2. Second order derivative is linear at splines 3. The natural function is required to be linear when X < smallest knot 4. The natural function is required to be linear when X > largest knot This means that the spline has 9-4 = 5 degrees of freedom. Since this includes a constant, which is absorbed in the intercept, it is counted as four degrees of freedom according to 'An Introduction to statistical learning'

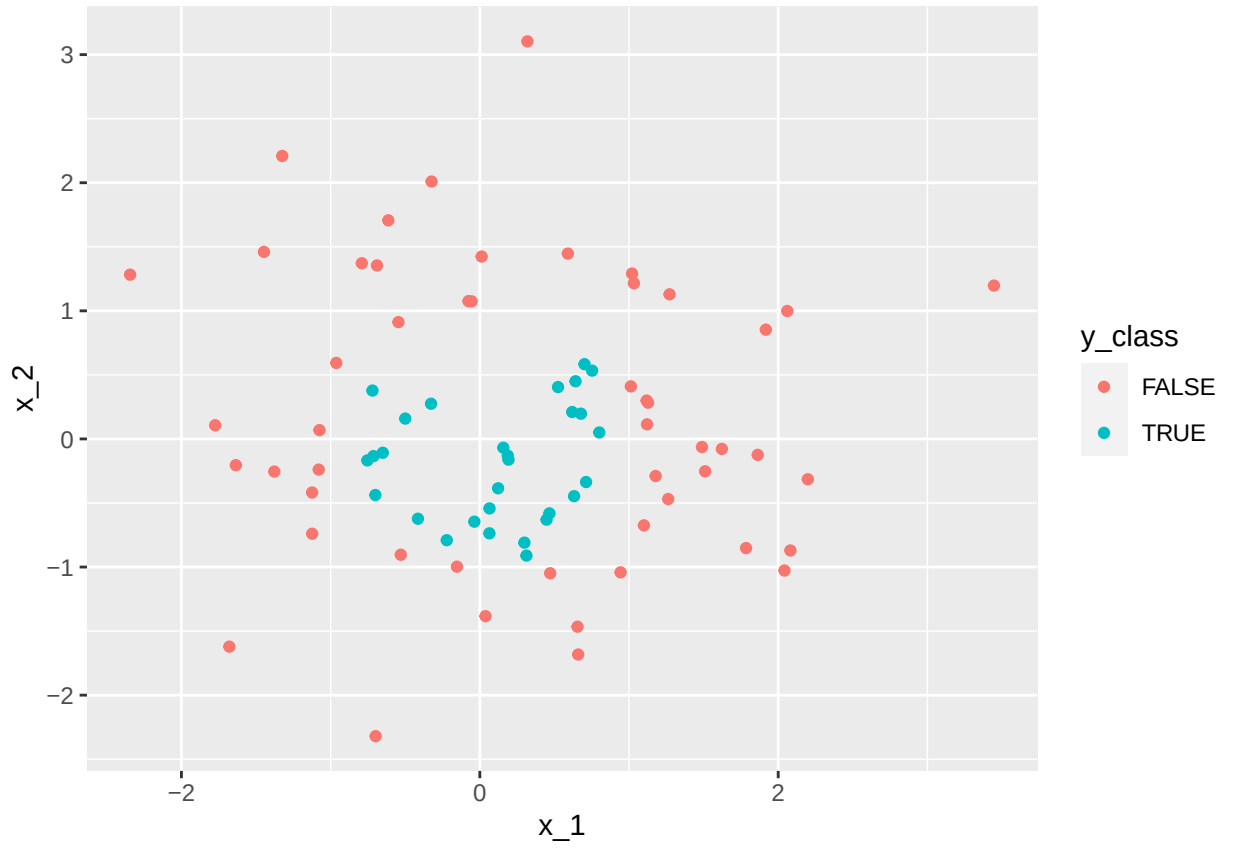
Question 4:

Answer the following questions and append a printout of your R code to the solution.

Subquestion 4.a)

Generate a simulated two-class data set $\{(x_i, y_i) : 1 \leq i \leq 100\}$ with features $x_i = (x_{i1}, x_{i2})$ and binary responses y_i, in which there is a visible but non-linear separation between the two classes. Split the dataset equally into training and test subsets.

```
set.seed(1337)
n <- 80
x_1 <- rnorm(n,mean = 0,sd = 1)
x_2 <- rnorm(n,mean = 0,sd = 1)
y_class <- x_1^2+x_2^2 < 1
SVM_DF <- tibble(x_1, x_2, y_class)

SVM_DF %>%
  ggplot(aes(x = x_1, y = x_2, color = y_class))+
  geom_point()
```




```
samples=sample(x = nrow(SVM_DF),size =  0.5*nrow(SVM_DF),replace = FALSE)
training = SVM_DF[samples, ]
testing = SVM_DF[-samples,]
```

Subquestion 4.b)

Fit a support vector classifier on the training data, using CV to choose cost. What is the test error rate?

```
pacman::p_load(e1071)
set.seed(13)
costvec <- seq(0.1, 1,length.out=100)
tuned_svm <-tune(svm, y_class ~ ., data = training, kernel = "linear", type = "C", ranges = list(cost =
print(paste0("The best 'test error rate' found by 10-k-fold is: ", round(tuned_svm$best.performance, 4))
```

```
## [1] "The best 'test error rate' found by 10-k-fold is: 0.35"
```

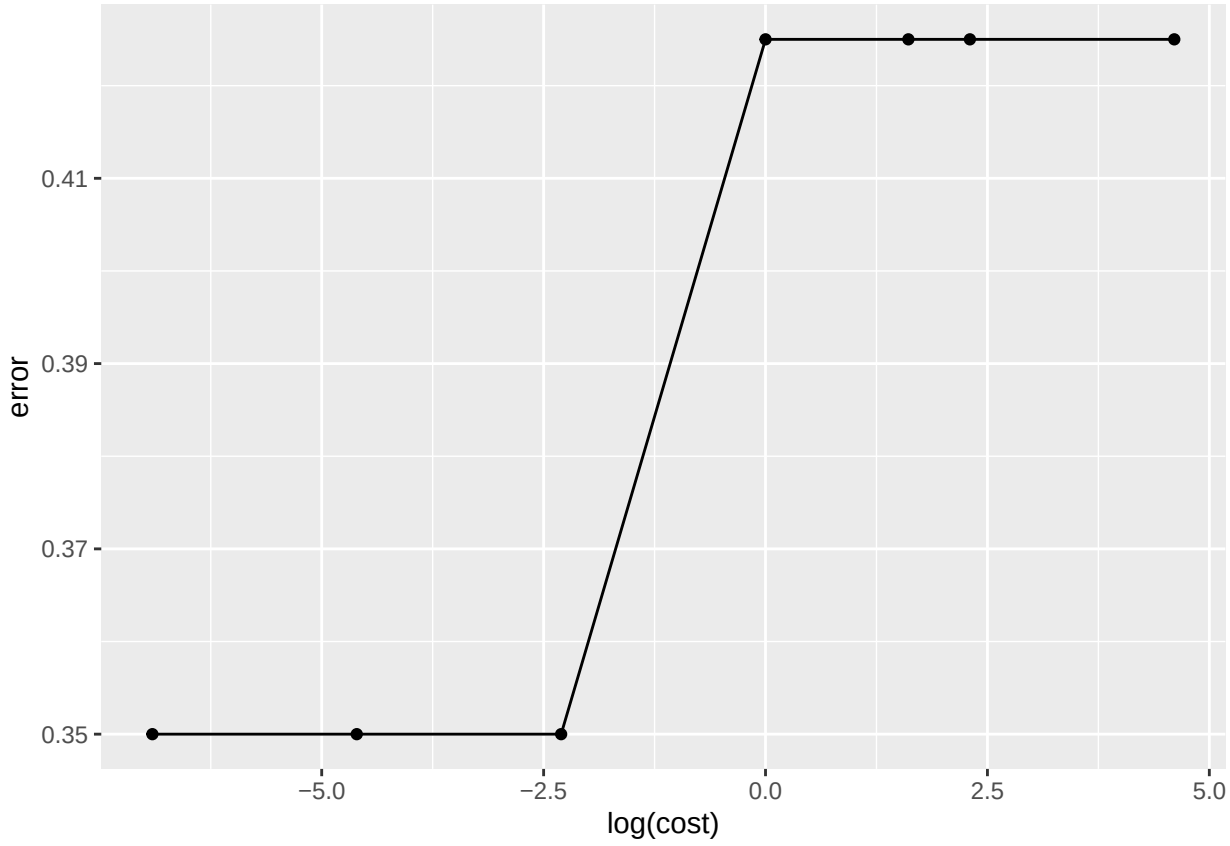
```
print(paste0("This test error is found at: 'cost' = " , round(tuned_svm$best.parameters$cost, 3)))
```

```
## [1] "This test error is found at: 'cost' = 0.001"
```

```
print(paste("And at", round(count(filter(tuned_svm$performances,error == tuned_svm$best.performance))[[1
```

```
## [1] "And at 2 other checked costs."
```

```
tuned_svm$performances %>%
  ggplot(aes(x = log(cost), y = error))+
  geom_point()+
  geom_line()
```



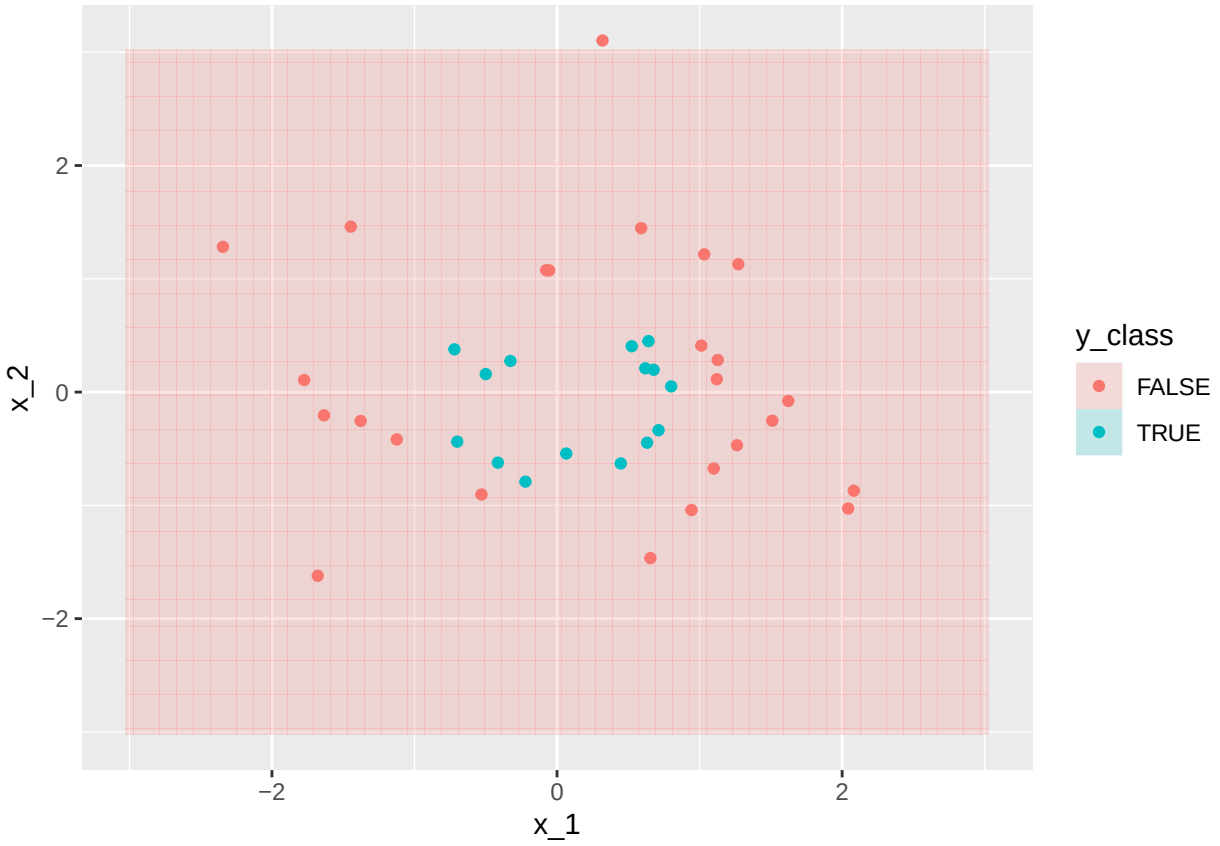

```
bestmod = tuned_svm$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = y_class ~ ., data = training, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear", type = "C", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.001
##
## Number of Support Vectors:  30
##
##  ( 14 16 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

```
class_pred = predict(bestmod, testing)
table(predicted = class_pred, true = testing$y_class)
```

```
##           true
## predicted FALSE TRUE
##     FALSE    25   15
##     TRUE      0    0
```

```
# Create a plot of predictions in ggplot
model_for_gg <- bestmod # change here

#####################################################
m <- 101
grid <- expand.grid(x_1=seq(-3,3,length=m),
                    x_2=seq(-3,3,length=m))
grid <- grid %>%
  mutate(y_class = predict(model_for_gg, newdata=grid) )
Pred <- ggplot(grid, aes(x=x_1, y=x_2, fill=y_class)) +
  geom_tile(aes(), alpha=.2) +
  geom_point(data=testing, aes(color=y_class))
Pred
```



Subquestion 4.c)

The decision boundary in (b) is given by $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$. In class, an alternative representation is given as

$$f(x) = \beta_0 + \sum_{i \epsilon s} \alpha_i \langle x, x_i \rangle$$

where S is the collection of indices of the support vectors. Express β_1 and β_2 in terms of α_i and x_i, $1 \leq i \leq 100$.

Answer: *Express* β_1*:* $\beta_1 = \alpha_i^{transpose} x_{1_i}$ $\beta_2 = \alpha_i^{transpose} x_{2_i}$

β_1 & β_2 can be expressed as the product between a transposed α-parameter vector and a vector consisting of the respective dimension of the support-vectors.

```
support_vector <- cbind("alpha" = bestmod$coefs, bestmod$SV)
colnames(support_vector)[1] <- "alpha"
support_vector
```

```
##            alpha         x_1         x_2
## 1   0.0010000000  0.18746019 -0.13090499
## 3   0.0010000000  0.29887827 -0.80947218
## 7   0.0010000000  0.19249191 -0.16169753
## 11  0.0010000000  0.46653706 -0.58097800
## 21  0.0010000000  0.70160645  0.58376718
## 23  0.0010000000  0.06427374 -0.73678148
## 29  0.0010000000  0.75250489  0.53339896
## 30  0.0010000000  0.15754969 -0.06857533
## 31  0.0010000000 -0.03645455 -0.64590441
## 32  0.0010000000  0.31264588 -0.91055344
## 34  0.0010000000 -0.75372581 -0.16762034
## 35  0.0010000000 -0.65031591 -0.10787613
## 38  0.0010000000  0.12264043 -0.38502931
## 39  0.0010000000 -0.71286911 -0.13401439
## 4  -0.0010000000  0.47183089 -1.04827752
## 5  -0.0010000000  1.11796617  0.29940591
## 6  -0.0010000000  0.65947131 -1.68283890
## 9  -0.0010000000 -1.12355513 -0.73977783
## 10 -0.0005393978  1.78456732 -0.85209216
## 12 -0.0001963132 -0.69826061 -2.32015947
## 15 -0.0010000000 -1.07460214  0.06873847
## 16 -0.0008036868  1.02056754  1.29119985
## 18 -0.0010000000 -0.96215911  0.59339153
## 19 -0.0010000000  1.48913612 -0.06298292
## 22 -0.0010000000  0.03818993 -1.38341700
## 25 -0.0010000000 -0.54535603  0.91201146
## 26 -0.0004606022 -0.79098417  1.37178507
## 28 -0.0010000000 -0.15320385 -0.99700049
## 33 -0.0010000000 -1.07960965 -0.23914384
## 37 -0.0010000000  1.17842106 -0.28949467
```

It is also easy to then express the function as: $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

```
# Find parameters of hyperplane expressed as (wx+c=0)
my_betas <- t(bestmod$coefs) %*% bestmod$SV # In essence this finds the beta_parametrs of the hyper pla
beta0 <- -bestmod$rho # the intercept
print(paste0(
  "f(x) = ",
  round(beta0,7),
  " + ",
  round(my_betas[,1],7),
  "*x_1 + ",
  round(my_betas[,2],7),
  "*x_2"
))
```

```
## [1] "f(x) = -0.9999241 + -0.0001947*x_1 + 9.27e-05*x_2"
```

Subquestion 4.d)

Fit a support vector machine on the training data with a polynomial kernel (with degree d > 1), using CV to choose cost. Compare the test error rate with (b).

```
set.seed(13)
costvec <- seq(0.1, 1,length.out=100)
tuned_svm <- tune.svm(y_class ~ ., data = training, kernel = "polynomial" ,type = "C", cost = costvec, 
print(paste0("The best 'test error rate' found by 10-k-fold is: ", round(tuned_svm$best.performance, 4))
```

```
## [1] "The best 'test error rate' found by 10-k-fold is: 0.05"
```

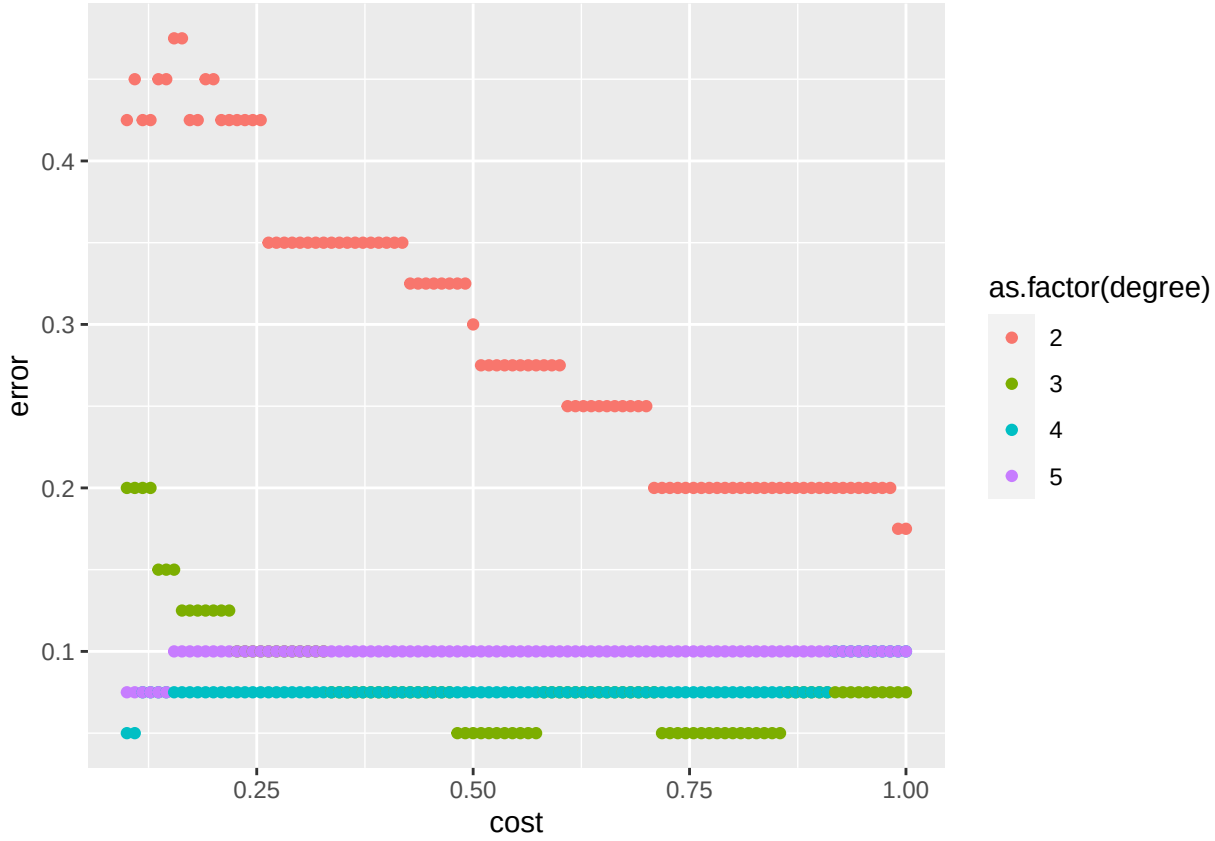
```
print(paste0("This test error is found at: 'cost' = " , round(tuned_svm$best.parameters$cost, 3)))
```

```
## [1] "This test error is found at: 'cost' = 0.1"
```

```
print(paste("And at", round(count(filter(tuned_svm$performances,error == tuned_svm$best.performance))[[1
```

```
## [1] "And at 28 other checked costs."
```

```
tuned_svm$performances %>%
  ggplot(aes(x = cost, y = error))+
  geom_point(aes(color = as.factor(degree)))
```




```
bestmod = tuned_svm$best.model
summary(bestmod)
```

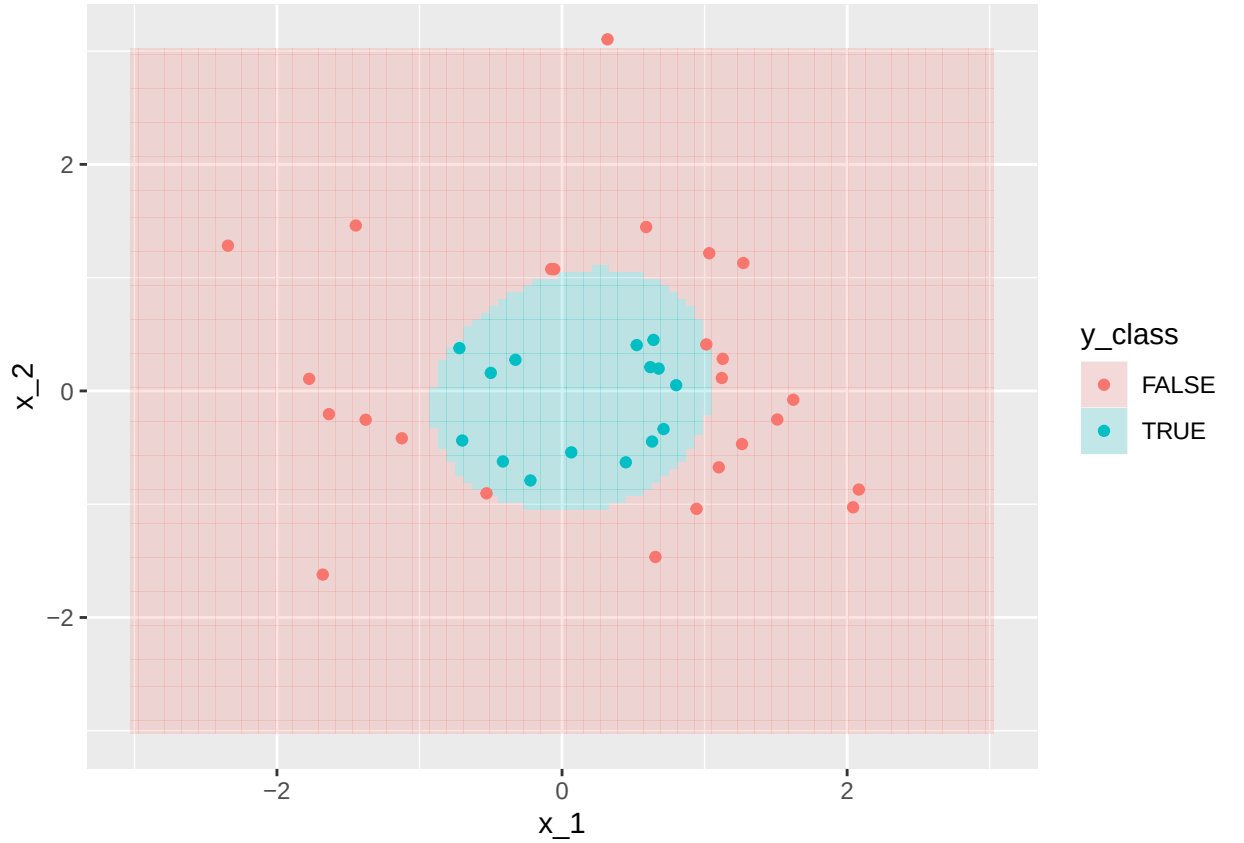
```
##
```

```
## Call:
## best.svm(x = y_class ~ ., data = training, degree = c(2, 3, 4, 5),
##     coef0 = 4, cost = costvec, kernel = "polynomial", type = "C",
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.1
##      degree:  4
##      coef.0:  4
##
## Number of Support Vectors:  10
##
##  ( 5 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

```
class_pred = predict(bestmod, testing)
table(predicted = class_pred, true = testing$y_class)
```

```
##          true
## predicted FALSE TRUE
##     FALSE    24    0
##     TRUE      1   15
```

```
# Create a plot of predictions in ggplot
model_for_gg <- bestmod # change here

#####################################################
m <- 101
grid <- expand.grid(x_1=seq(-3,3,length=m),
                    x_2=seq(-3,3,length=m))
grid <- grid %>%
  mutate(y_class = predict(model_for_gg, newdata=grid) )
Pred <- ggplot(grid, aes(x=x_1, y=x_2)) +
  geom_tile(aes(fill=y_class), alpha=.2) +
  geom_point(data=testing, aes(color=y_class))
Pred
```



Conclusion to 4.d: By fitting a model with a 4th degree polynomial and coef0 = 4, then the test error is reduced to: 0.025 For the linear model this test error was equal to $\frac{\text{True Cases}}{\text{False Cases + True Cases}}$, as it would always predict FALSE within reasonably observable intervals. This was 15/(15+25) for the generated test set, or as 0.375.

The polynomial kernel greatly outperforms the linear as 0.025 » 0.375.

Subquestion 4.e)

For your choice of degree d in (d), derive the transformed features $h(x) = (h_1(x), \cdots, h_p(x))$ such that the decision boundary has the form $f(x) = \beta_0 + \sum_{j=1}^{p} \beta_j h_j(x)$

```
support_vector <- cbind("alpha" = bestmod$coefs, bestmod$SV)
colnames(support_vector)[1] <- "alpha"
support_vector
```

```
##          alpha        x_1        x_2
## 3   0.09343479  0.2988783 -0.8094722
## 29  0.09717317  0.7525049  0.5333990
## 32  0.10000000  0.3126459 -0.9105534
## 34  0.10000000 -0.7537258 -0.1676203
## 39  0.07359074 -0.7128691 -0.1340144
## 4  -0.10000000  0.4718309 -1.0482775
## 5  -0.10000000  1.1179662  0.2994059
## 25 -0.06419870 -0.5453560  0.9120115
## 28 -0.10000000 -0.1532038 -0.9970005
## 33 -0.10000000 -1.0796096 -0.2391438
```

$$f(x) = \beta_0 + \sum_{i \epsilon s} \alpha_i K(x, x_i)$$

$$f(x) = \beta_0 + \sum_{i \epsilon s} \alpha_i (4 + \sum_{j=1}^{p} x_{ij} x_{i'j})^4$$

$$f(x) = \beta_0 + \sum_{i \epsilon s} \alpha_i (4 + x_{i1} x_{i'1} + x_{i2} x_{i'2})^4$$

I will avoid expanding this further, but it would follow the same expansion as :

$$(4 + \ a \cdot \ b + \ c \cdot \ d)^4 =$$

$a^4b^4+c^4d^4+16a^3b^3+16c^3d^3+96a^2b^2+96c^2d^2+256ab+256cd+4a^3b^3cd+4abc^3d^3+48a^2b^2cd+48abc^2d^2+192abcd+6a^2b^2c^2d^2+256$

Which consists of 15 different h(x), each separated by a +.

It can therefore be rewritten as a nonlinear function h(x_i), $0 < i < 16$

$$f(x) = \beta_0 + \sum_{i \epsilon s} \alpha_i h(x)$$