Homework 4

Fiara Causo / Stefan Ciric

12/1/2019

In this homework, the problems will be more computationally intensive than previous problems in this class. You should expect the code for some problems to take a couple of minutes to complete. Re-knitting your file can take a long time so you should consider using `cache=TRUE` option in R chunks that involve code which takes a while to run. Another option would be to work some of the more computationally demanding problems in separate Rmd. Please load the following packagesfor this homework:

```r
library(tidyverse)
library(tree)
library(randomForest)
library(gbm)
library(ROCR)
library(e1071)
library(imager)
```

1. Fundamentals of the bootstrap

In the first part of this problem we will explore the fact that approximately 1/3 of the observations in a bootstrap sample are *out-of-bag*.

a)

Given a sample of size n, what is the probability that any observation j is *not* in in a bootstrap sample? Express your answer as a function of n.

Suppose original sample size is n,
We are to sample *with replacement* n obsevations from original sample,
Probability observation j is not selected on a given draw is $\frac{n-1}{n}$,
and,
probability observation j is not selected on any of the n draws is $(\frac{n-1}{n})^n$

b)

Compute the above probability for $n = 1000$.

Let X be a sample of 1000 observations generated from an original sample of 1000 observations. Then, $\mathbb{P}[j \notin X] = (\frac{999}{1000})^{1000} = 36.77\%$.

c) Verify that your calculation is reasonable by resampling the numbers 1 to 1000 with replace and printing the number of missing observations. Hint: use the `unique` and `length` functions to identify how many unique observations are in the sample.

```r
set.seed(3)
samp <- 1:1000
samp1 <- sample(samp, size = 1000, replace = TRUE)
```

```r
# The probability that an observation is not in a single
# sample is
(1000 - length(unique(samp1)))/1000
```

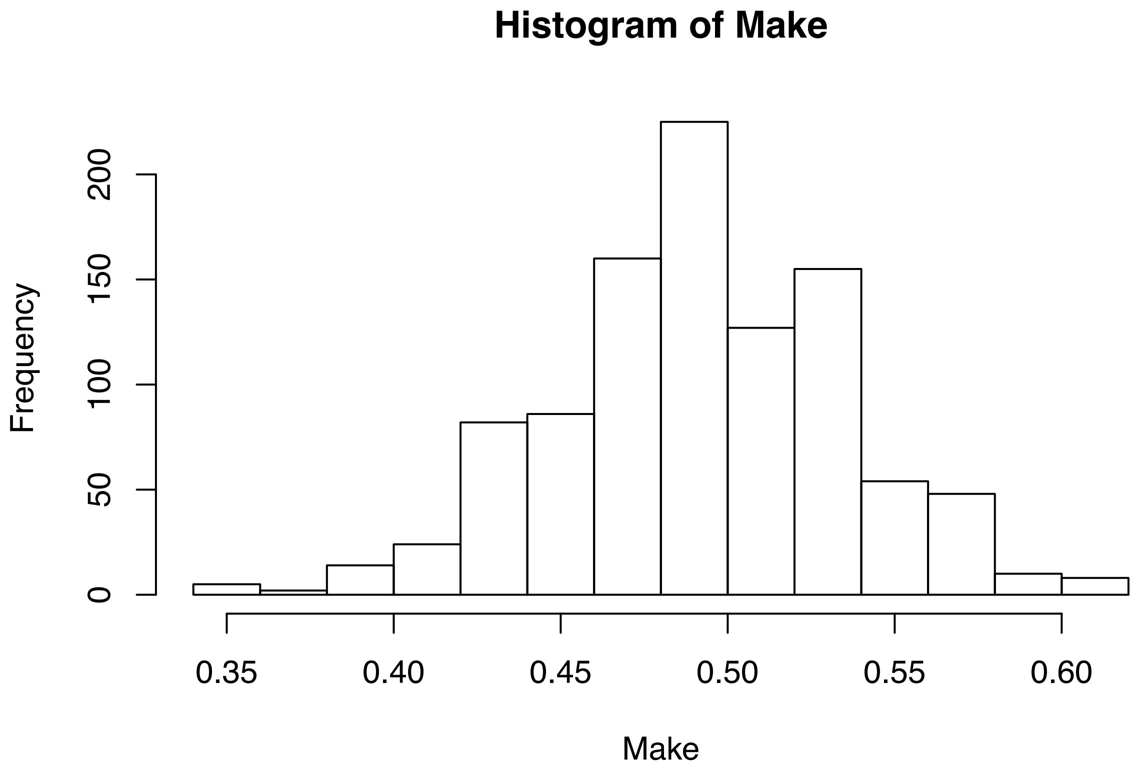
```
## [1] 0.37
```

```r
# The difference of 0.0047 between our empirical and
# theoretical result is less than 1.3% of the theoretical
# result, and so we determine that it is due to sampling
# error.
```

Here we'll use the bootstrap to compute uncertainty about a parameter of interest.

d) By November 19, 2015, Stephen Curry, an NBA basketball player regarded as one of the best players currently in the game, had made 62 out of 126 three point shot attempts (49.2%). His three point field goal percentage of 0.492, if he maintains it, will be one of the best all time for a single season. Use bootstrap resampling on a sequence of 62 1's (makes) and 64 0's (misses). For each bootstrap sample compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 95% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the `quantile` function in R. Print the endpoints of this interval. However, this estimate, and the associated uncertainty, exclude information about his career performance as well as the typical shooting skill for other players in the league. For reference, prior to this year, Stephen Curry had made about 43% of all three point shots in his career. Despite the fact that the bootstrap histogram shows that it is about equally likely that Curry's true skill is greater or elss than 0.492, why do you expect that his end-of-season field goal percentage will in fact be lower than his percentage on 11/19? *Hint:* look up the phenomenon known as "regression to the mean".

```r
mm <- c(rep(0, each = 64), rep(1, each = 62))
Make <- vector()
for (i in 1:1000) {
    samp <- sample(mm, size = 126, replace = TRUE)
    made <- sum(samp)
    Make[i] <- made/126
}
hist(Make)
```



```
# If we are to assume an unknown distribution, a 95%
# confidence interval for the median may be constructed by
# Lower Bound = median - 1.96*sqrt(median*2)/2 Upper Bound =
# 1 + median + 1.96*sqrt(median*2)/2

n <- median(Make) * 126 * 2

# Lower Bound
(median(Make) * 126 - 1.96 * sqrt(n)/2)/126
```

```
## [1] 0.4054538
```

```
# Upper Bound
(1 + median(Make) * 126 + 1.96 * sqrt(n)/2)/126
```

```
## [1] 0.5866097
```

```
# By CLT and the historgram we assume normal distribution,
# hence 95% CI for population mean is bounded by mean +/-
# 1.96*StDev However, the problem specifies to use the
# quantile function, which indicates that the median is
# wanted, rather than the mean, to represent the
# 'true'average FG%.

std <- sqrt(var(Make))
lb <- quantile(Make, 0.5) - 1.96 * std
ub <- quantile(Make, 0.5) + 1.96 * std
```

```
# We are 95% confidence the true average is between
lb
```

```
##       50%
## 0.4038643
```

```
# and
ub
```

```
##       50%
## 0.5802627
```

```
# Both solutions for the confidence intervals are nearly the
# same, so we are satisfied with either.
```

Given that the score of 49.x% would be record setting, and given that his previous year was nearly 8% lower we have reason to believe his true mean lies somewhere in the middle.

2. Eigenfaces

In this problem we will use PCA to explore variation in images of faces. Load the data saved in `faces_array.RData` with the `load` function. This will load a 100 x 100 x 1000 *array* of data. An array is a generalization of a matrix to more than 2 dimensions. In this example, the first two dimensions index the pixels in a 100 x 100 black and white image of a face. The last dimension is the index for one of 1000 face images. The faces used in this example are from 1000 images scraped from the internet. See https://cyberextruder.com/face-matching-data-set-download/ for more info.

```
load("faces_array.RData")
```

Although it is natural to think about an stack of 1000 matrices representing each of the face images, to run PCA we need to input a single matrix. To do this, we'll convert each 100 x 100 matrix to a single vector of length 100*100 = 10000. When you call `as.numeric` on a matrix, it stacks each of the columns in the matrix into one large vector. Thus, we can think of our data as 1000 observations of a 10000 variables (one variable per pixel). Run the following code to get a matrix of face observations.

```
face_mat <- sapply(1:1000, function(i) as.numeric(faces_array[,
    , i])) %>% t
```

When we want to visualization an image, we need to take the 10000 dimensional vector and reconstruct it as a matrix. The code `plot_face` takes a single 10000 dimensional vector (e.g. a column of `face_mat`), converts it back to a matrix, and plots the resulting image. You can test this functionality by printing a random face from the dataset: `plot_face(face_mat[sample(1000, 1), ])`.

```
plot_face <- function(image_vector) {
    plot(as.cimg(t(matrix(image_vector, ncol = 100))), axes = FALSE,
        asp = 1)
}
```

a) Find the "average" face in this dataset by averaging all of the columns in `face_mat`. Plot the average face by calling `plot_face` on the average.

```
sum_face = matrix(0, 1, 10000)
for (i in 1:1000) {
    sum_face = sum_face + face_mat[i, ]
}
average_face <- sum_face/1000
plot_face(average_face)
```

b) Run PCA on `face_mat` setting `center=TRUE` and `scale=FALSE`. In class we mentioned that in general it is best if `scale=TRUE` because it puts all variables on the same scale and we don't have to worry about the units of the variables (remember, the scale of the variables affects our results). In general, this is good practice, especially when the predictor variables are of mixed types. Here, each variable represents a single pixel intensity (in black & white) and so all variables already have the same units and same scale (minimum of 0 and maximum of 255). In this case, setting `scale=FALSE` actually seems to give slightly better results. Plot the PVE and cumulative PVE from the PCA. How many PCs do you need to explain at least 50% of the total variation in the face images?

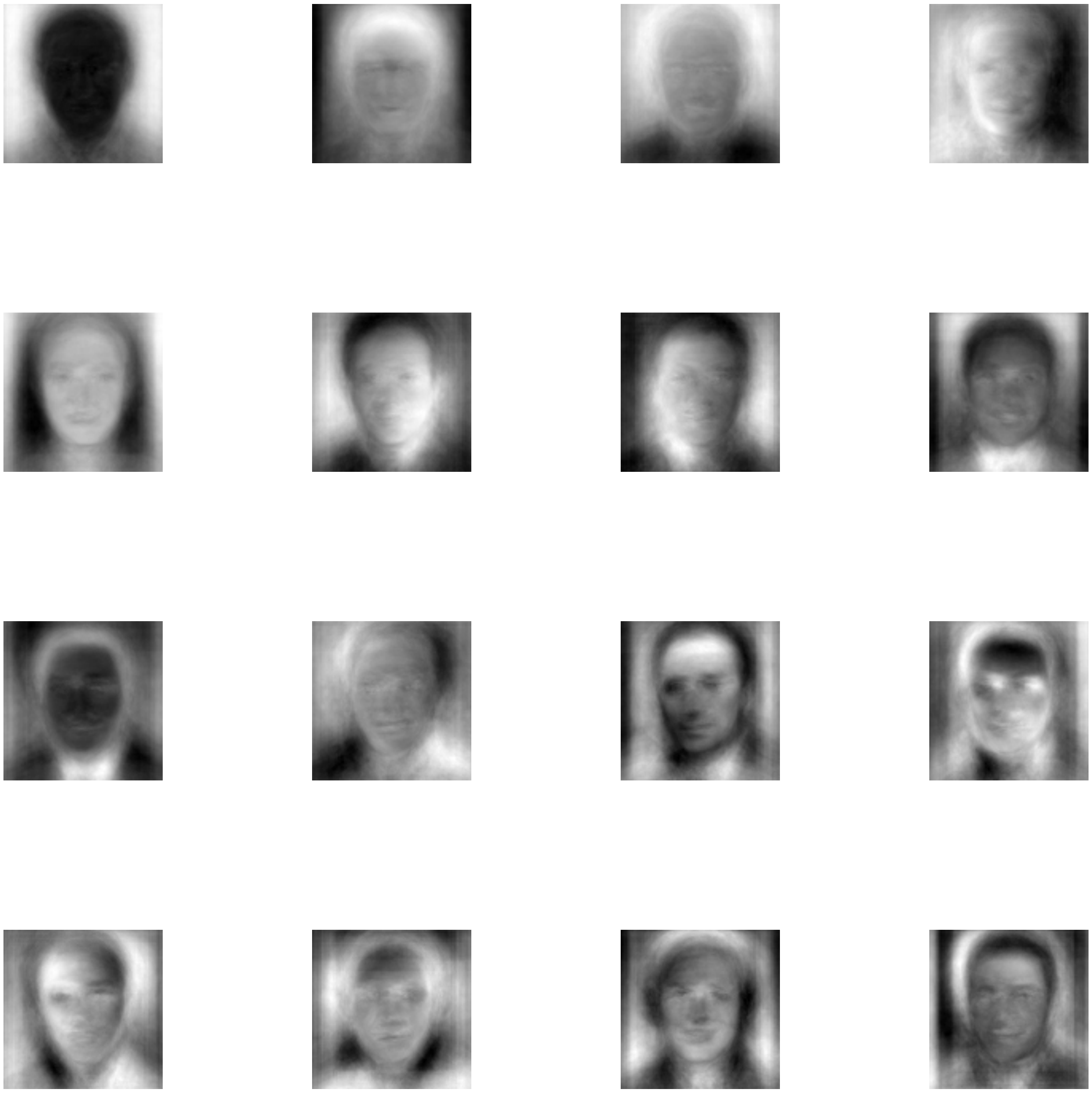
```
pr.out = prcomp(face_mat, center = TRUE, scale = FALSE)
pr.var = pr.out$sdev^2
pve = pr.var/sum(pr.var)
cumulative_pve <- cumsum(pve)
```

```
# This will put the next two plots side by side
par(mfrow = c(1, 2))
# Plot proportion of variance explained
plot(pve, type = "l", lwd = 3, xlab = "Principal Component",
    ylab = "PVE", ylim = c(0, 1))
plot(cumulative_pve, type = "l", lwd = 3, xlab = "Principal Component ",
    ylab = " Cumulative PVE ", ylim = c(0, 1))
```

You need 5 PCs to explain atleast 50% of the total variation in face images.

c) Plot the first 16 principle component directions as faces using the `plot_face` function (these are the columns of the `rotation` matrix). Early researchers termed these "eigenfaces" since they are eigenvectors of the matrix of faces. The code below will adjust the margins of you plot and specifies a layout for the 16 images. `par(mfrow=c(4,4))` specifies a grid of 4 x 4 images. Each time you call plot_face it will plot the next face in one of the new grid cells. All you need to do is call plot_face 16 times (please use a for loop). Note that these images describe “directions” of maximum variability in the face images. You should interpret light and dark regions in the eigenfaces as regions of high *contrast*, e.g. your interpretation should not change if you inverted black and white in the images.

```
par(mfrow = c(4, 4))
for (i in 1:16) {
    plot_face(pr.out$rotation[, i])
}
```

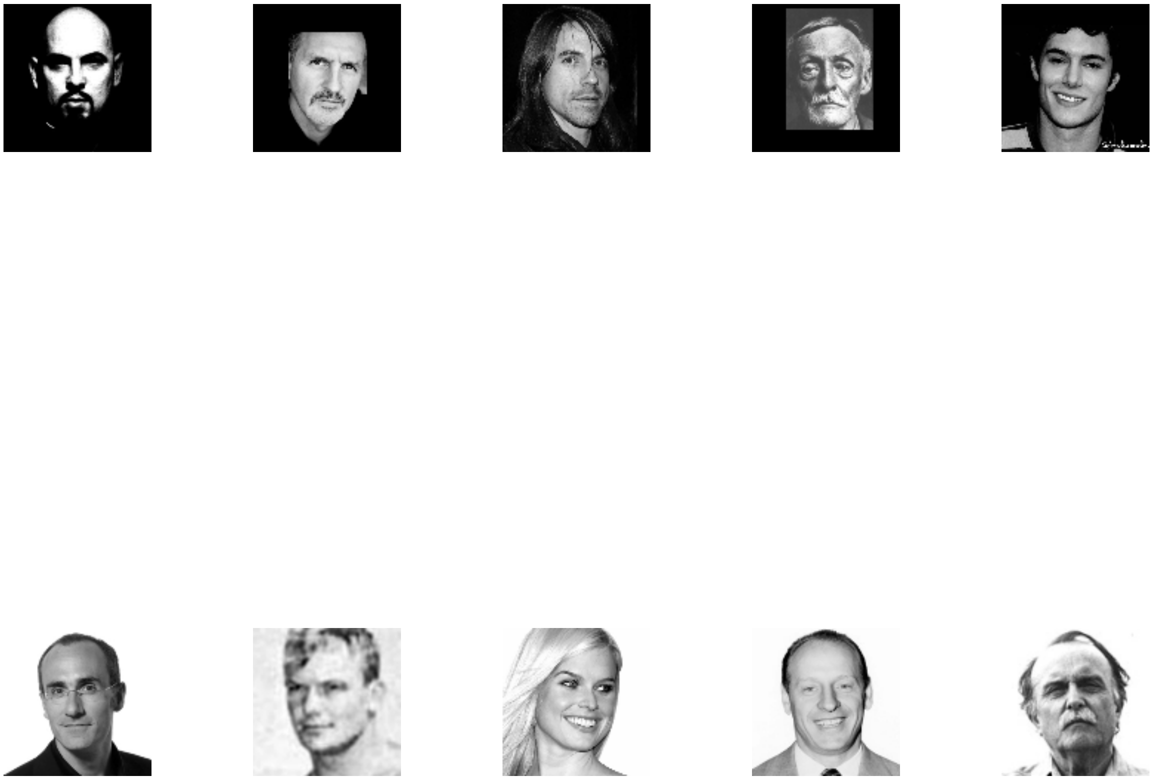


d) In this part, we will examine faces that have the highest and lowest values for specific PCs. Plot the faces with the 5 largest values on PC1 and the 5 smallest values for PC1. Based on the example faces, and the first eigenface from the previous part and the 10 example images, what aspect of variability in the face images is captured by the first component.

```r
pc1_values <- order(pr.out$x[, 1], decreasing = FALSE)
bottom_values <- pc1_values[0:5]
top_values <- tail(pc1_values, 5)
par(mfrow = c(2, 5))

# Smallest Values
for (j in bottom_values) {
    plot_face(face_mat[j, ])
}

# Largest Values
for (j in top_values) {
    plot_face(face_mat[j, ])
}
```



Based on the example faces, it seems that the first principal component captures the contrast between the face and the background. Faces on darker backgrounds are seen for lower values of PC1 and faces on lighter backgrounds are seen for higher values of PC1.

e) Repeat part d) but now display example faces with the largest and smallest values on principal component 5. Again, discuss what aspect of variability in the face images is best captured by this principal component. Based on your results, which principal component, (1 or 5) would be more useful as a feature in a face recognition model (e.g. a model which predicts the identity of the individual in an image)

```r
pc5_values <- order(pr.out$x[, 5], decreasing = FALSE)
bottom_values5 <- pc5_values[0:5]
top_values5 <- tail(pc5_values, 5)
par(mfrow = c(2, 5))
```

```r
# Smallest Values
for (j in bottom_values5) {
    plot_face(face_mat[j, ])
}

# Largest Values
for (j in top_values5) {
    plot_face(face_mat[j, ])
}
```

Based on the results, it would seem that the 5th Principal Component captures the length of hair on the face. For lower values of PC5, we see pictures with darker frames around the face and for higher values of PC5 we see faces with longer hair. It would seem that PC5 better helps us predict the identity of the individual *because length of hair can help us predict gender to a certain extent, as opposed to image background from PC1 which wouldn't help as much.*

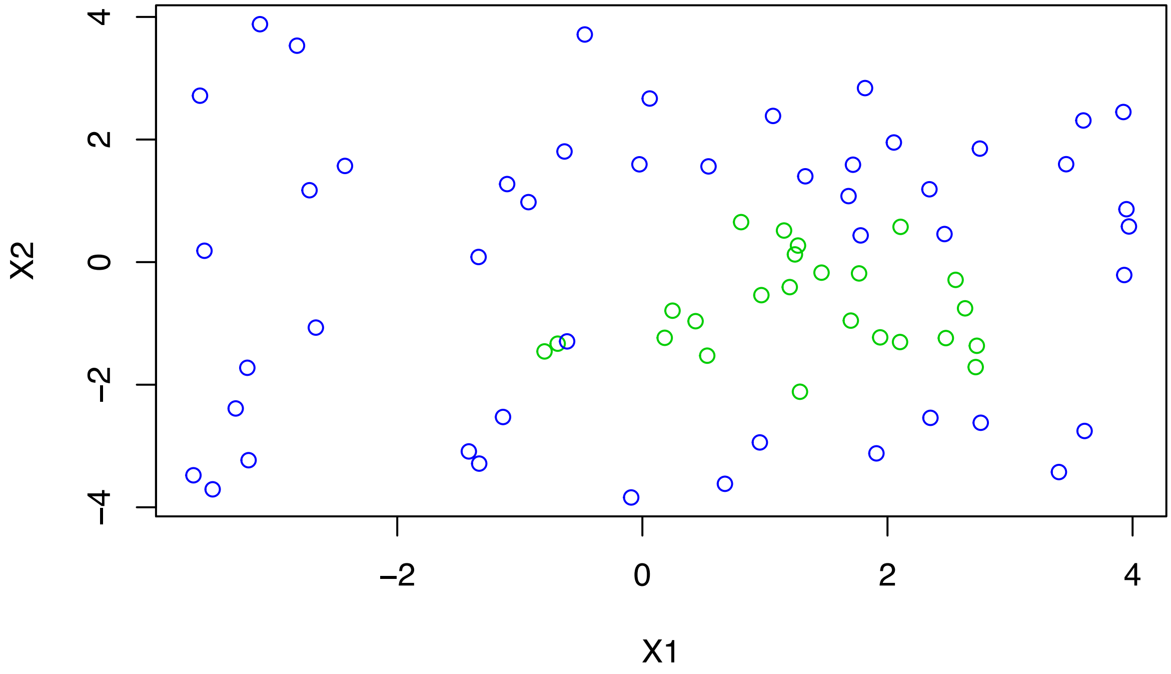
3. Logistic regression with polynomial features

a) In class, we have used polynomial linear regression several times as an example for model complexity and the bias variance tradeoff. We can also introduce polynomial logistic regression models to derive more sophisticated classification functions by introducing additional features. Use `read_csv` to load `nonlinear.csv` and plot the data. Plot each point colored according to its class, `Y`.

```
dat <- read_csv("nonlinear.csv")
```

```
## Parsed with column specification:
## cols(
##   Z = col_integer(),
##   X1 = col_double(),
##   X2 = col_double(),
##   Y = col_integer()
## )
```

```
plot(X2 ~ X1, dat, col = ifelse(dat$Y == 0, 3, 4))
```

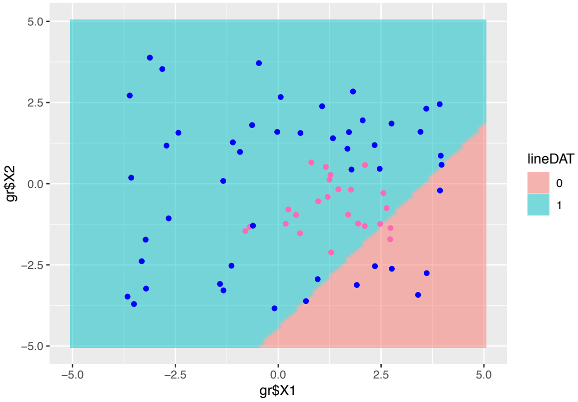


```
nrow(dat)
```

```
## [1] 72
```

b) Fit a logistic regression model of `Y` on `X1` and `X2`. The decision boundary can be visualized by making predictions of class labels over finely sampled grid points that cover your region (sample space) of interest. The following code will create grid points over the sample space as below: For each point in `gr`, predict a class label using the logistic regression model. You should classify based on the probability being greater or less than 1/2. Visualize your predictions at each point on the grid using the `geom_raster` function. This function colors in rectangles on the defined grid and is a good way to visualize your decision boundary. Set the `fill` aesthetic to your predicted label and outside of the `aes` use `alpha=0.5` to set the transparency of your predictions. Plot the observed data, colored by label, over the predictions using `geom_point`.

```r
# grid of points over sample space
gr <- expand.grid(X1=seq(-5, 5, by=0.1),  # sample points in X1
                  X2=seq(-5, 5, by=0.1))  # sample points in X2
glmDAT <- glm(Y~X1+X2, data =dat, family = binomial)
preDAT <- predict(glmDAT, gr, type = "response")
lineDAT <- as.factor(ifelse(preDAT<=0.5,0,1))
ggplot(gr, aes(gr$X1, gr$X2))+geom_raster(aes(fill=lineDAT), alpha=0.5)+geom_point(aes
      (dat$X1,dat$X2), data = dat,col = ifelse(dat$Y ==0,"hotpink",4))
```

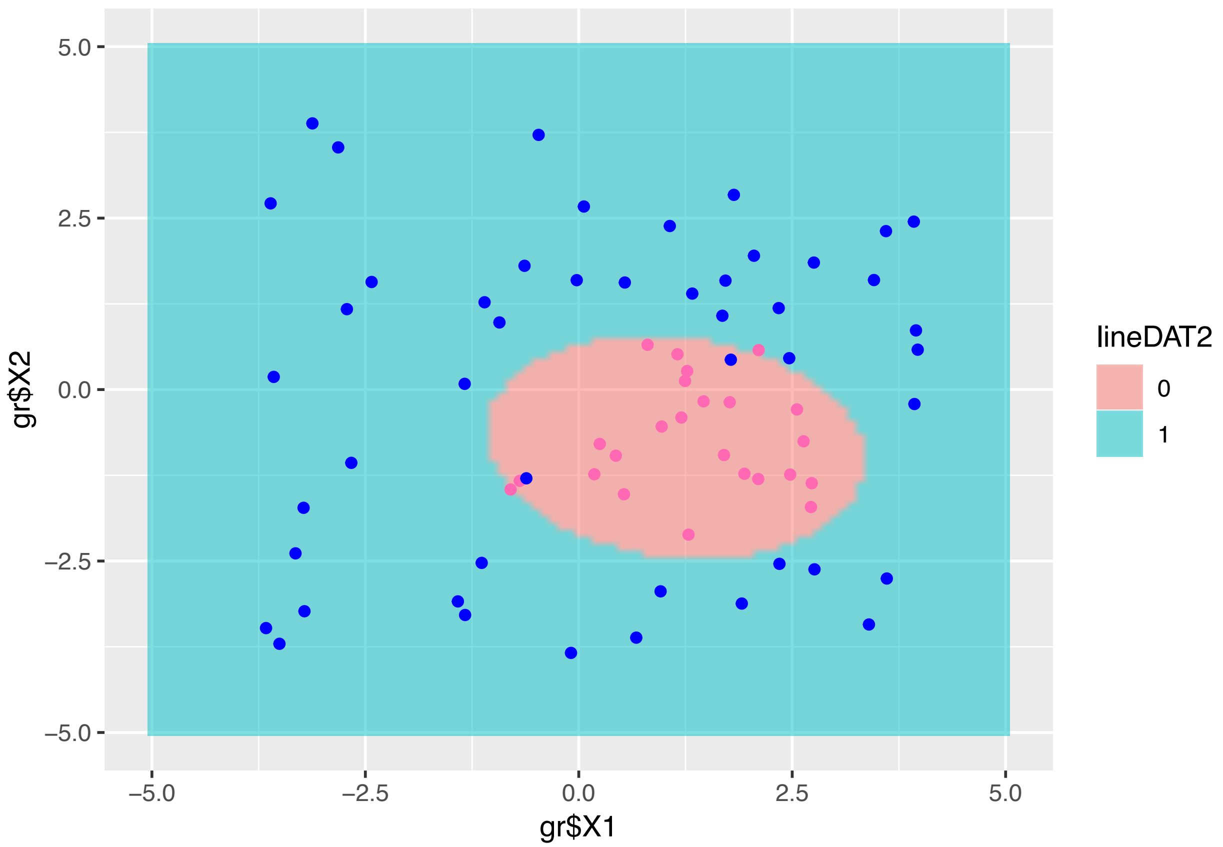


c) Fit a model involving 2nd degree polynomial of `X1` and `X2` with interaction terms. You should use the `poly()` function. Inspect result of the fit using `summary()`. Plot the resulting decision boundary.

```r
glmDAT2 <- glm(Y ~ poly(X1, X2, degree = 2), data = dat, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
preDAT2 <- predict(glmDAT2, gr, type = "response")
lineDAT2 <- as.factor(ifelse(preDAT2 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT2),
    alpha = 0.5) + geom_point(aes(dat$X1, dat$X2), data = dat,
    col = ifelse(dat$Y == 0, "hotpink", 4))
```



```r
summary(glmDAT2)
```

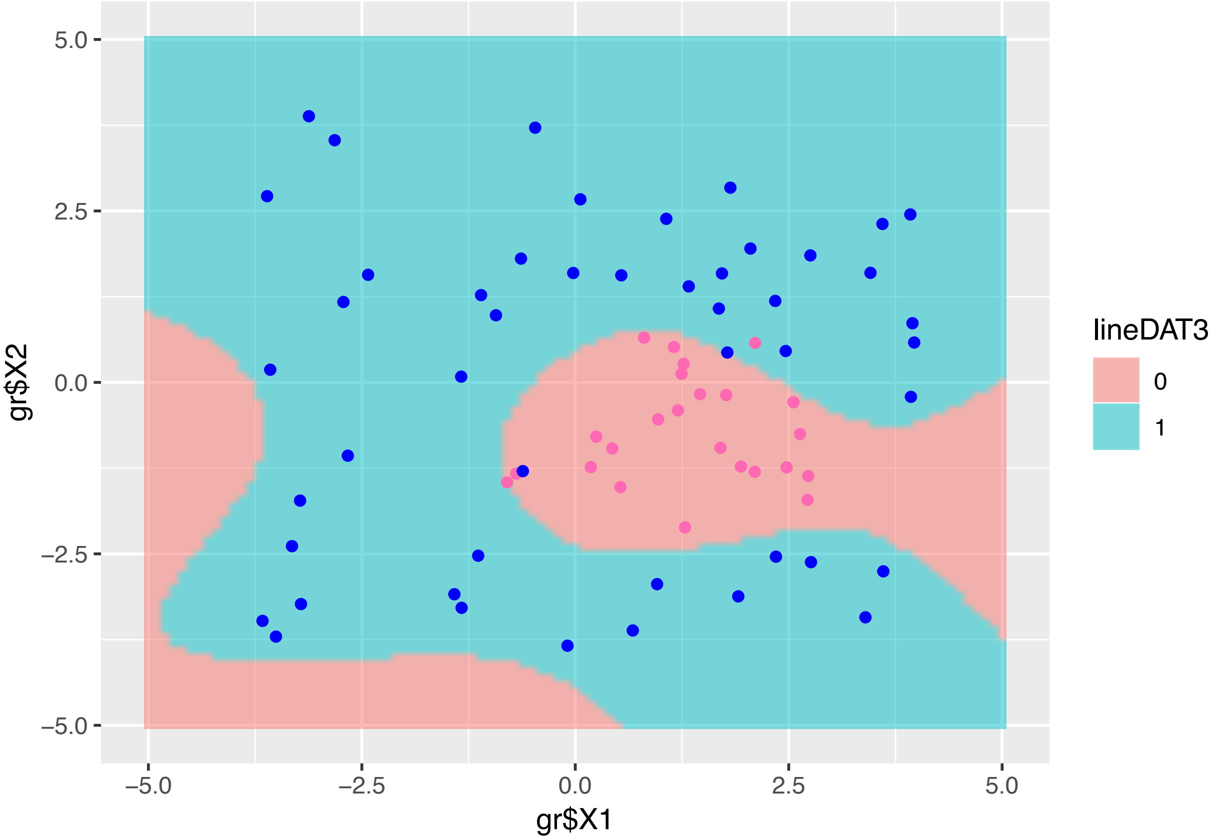
```
##
## Call:
## glm(formula = Y ~ poly(X1, X2, degree = 2), family = binomial,
##     data = dat)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.39081  -0.08271   0.00000   0.00930   1.90069
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                     11.722      4.789   2.448   0.0144 *
## poly(X1, X2, degree = 2)1.0  -49.657     27.060  -1.835   0.0665 .
## poly(X1, X2, degree = 2)2.0   57.777     29.043   1.989   0.0467 *
## poly(X1, X2, degree = 2)0.1   50.184     24.268   2.068   0.0387 *
## poly(X1, X2, degree = 2)1.1  157.199    231.044   0.680   0.4963
## poly(X1, X2, degree = 2)0.2   96.311     39.733   2.424   0.0154 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 91.658  on 71  degrees of freedom
## Residual deviance: 13.852  on 66  degrees of freedom
## AIC: 25.852
##
## Number of Fisher Scoring iterations: 10
```

d) Using the same procedure, fit a logistic regression model with 5-th degree polynomials without any interaction terms. Inspect result of the fit using `summary()`. Plot the resulting decision boundary and discuss the result. Explain the reason for any strange behvaior.

```
glmDAT3 <- glm(Y ~ poly(X1, X2, degree = 5), data = dat, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
preDAT3 <- predict(glmDAT3, gr, type = "response")
lineDAT3 <- as.factor(ifelse(preDAT3 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT3),
    alpha = 0.5) + geom_point(aes(dat$X1, dat$X2), data = dat,
    col = ifelse(dat$Y == 0, "hotpink", 4))
```



```
summary(glmDAT3)
```

```
##
## Call:
## glm(formula = Y ~ poly(X1, X2, degree = 5), family = binomial,
##     data = dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -8.49    0.00    0.00    0.00    8.49
```

```
##
## Coefficients:
##                               Estimate Std. Error   z value Pr(>|z|)
## (Intercept)                  1.364e+15  1.022e+07 133487956   <2e-16 ***
## poly(X1, X2, degree = 5)1.0  1.145e+15  9.346e+07  12251576   <2e-16 ***
## poly(X1, X2, degree = 5)2.0  5.671e+15  8.492e+07  66773905   <2e-16 ***
## poly(X1, X2, degree = 5)3.0  7.322e+14  7.884e+07   9287024   <2e-16 ***
## poly(X1, X2, degree = 5)4.0 -2.455e+15  8.225e+07 -29842754   <2e-16 ***
## poly(X1, X2, degree = 5)5.0 -5.013e+14  7.823e+07  -6408591   <2e-16 ***
## poly(X1, X2, degree = 5)0.1  4.060e+15  1.191e+08  34082507   <2e-16 ***
## poly(X1, X2, degree = 5)1.1  1.739e+16  8.966e+08  19398167   <2e-16 ***
## poly(X1, X2, degree = 5)2.1  7.025e+15  7.083e+08   9917835   <2e-16 ***
## poly(X1, X2, degree = 5)3.1  4.391e+15  7.586e+08   5788898   <2e-16 ***
## poly(X1, X2, degree = 5)4.1  1.051e+16  7.586e+08  13847620   <2e-16 ***
## poly(X1, X2, degree = 5)0.2  1.431e+16  1.533e+08  93404326   <2e-16 ***
## poly(X1, X2, degree = 5)1.2  5.958e+16  1.147e+09  51950615   <2e-16 ***
## poly(X1, X2, degree = 5)2.2  8.165e+15  7.889e+08  10350179   <2e-16 ***
## poly(X1, X2, degree = 5)3.2 -2.429e+16  8.222e+08 -29538788   <2e-16 ***
## poly(X1, X2, degree = 5)0.3 -2.901e+15  1.527e+08 -18994170   <2e-16 ***
## poly(X1, X2, degree = 5)1.3 -9.580e+15  1.193e+09  -8027227   <2e-16 ***
## poly(X1, X2, degree = 5)2.3 -1.461e+16  8.626e+08 -16931553   <2e-16 ***
## poly(X1, X2, degree = 5)0.4  1.989e+15  1.195e+08  16637650   <2e-16 ***
## poly(X1, X2, degree = 5)1.4  3.106e+16  8.761e+08  35454923   <2e-16 ***
## poly(X1, X2, degree = 5)0.5  2.498e+15  9.170e+07  27241172   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:  91.658  on 71  degrees of freedom
## Residual deviance: 288.349  on 51  degrees of freedom
## AIC: 330.35
##
## Number of Fisher Scoring iterations: 21
```

This model responds to noise as thought it's signal, and conforms it's shape to satisfy the highly idiosyncratic behavior of randomness. The plot hugs the blue points closely, and fills in the rest with pink, even though there are no pink points in the viscinity. It is therefor too sensitive to the blue points, because it is responding to every piece of information that it has, rather than leaving unknowns as unknowns.

e) Qualitatively, compare the relative magnitudes of coefficients of in the two polynomial models and the linear model. What do you notice? Your answer should mention bias, variance and/or overfitting.

```
cbind(summary(glmDAT3)$coef[, 1])
```

```
##                                      [,1]
## (Intercept)                  1.363820e+15
## poly(X1, X2, degree = 5)1.0  1.145028e+15
## poly(X1, X2, degree = 5)2.0  5.670503e+15
## poly(X1, X2, degree = 5)3.0  7.321926e+14
## poly(X1, X2, degree = 5)4.0 -2.454601e+15
## poly(X1, X2, degree = 5)5.0 -5.013456e+14
## poly(X1, X2, degree = 5)0.1  4.060166e+15
## poly(X1, X2, degree = 5)1.1  1.739213e+16
## poly(X1, X2, degree = 5)2.1  7.024907e+15
## poly(X1, X2, degree = 5)3.1  4.391183e+15
## poly(X1, X2, degree = 5)4.1  1.050510e+16
## poly(X1, X2, degree = 5)0.2  1.431455e+16
## poly(X1, X2, degree = 5)1.2  5.957502e+16
## poly(X1, X2, degree = 5)2.2  8.165323e+15
## poly(X1, X2, degree = 5)3.2 -2.428738e+16
## poly(X1, X2, degree = 5)0.3 -2.900987e+15
## poly(X1, X2, degree = 5)1.3 -9.580223e+15
## poly(X1, X2, degree = 5)2.3 -1.460529e+16
## poly(X1, X2, degree = 5)0.4  1.988873e+15
## poly(X1, X2, degree = 5)1.4  3.106163e+16
## poly(X1, X2, degree = 5)0.5  2.498039e+15
```

```
cbind(summary(glmDAT2)$coef[, 1])
```

```
##                                  [,1]
## (Intercept)                  11.72238
## poly(X1, X2, degree = 2)1.0 -49.65662
## poly(X1, X2, degree = 2)2.0  57.77664
## poly(X1, X2, degree = 2)0.1  50.18399
## poly(X1, X2, degree = 2)1.1 157.19889
## poly(X1, X2, degree = 2)0.2  96.31060
```

The magnitude of each coefficient in the 2 degree polynomial model is much larger than the corresponding coefficient in the 5 degree polynomial model, which indicates that each attribute in the lower degree model is more impactful than any attribute in the higher degree model. This implies that the higher degree model overfits to the sample data, which implies that fresh data from the population will have greater variance from the 5 degree model than the 2 degree model. The benefit of that overfitting is that the higher degree model theoretically offers lower bias, however whether that is a material difference remains to be seen. The 2 degree model seems to be an excellent fit so far.

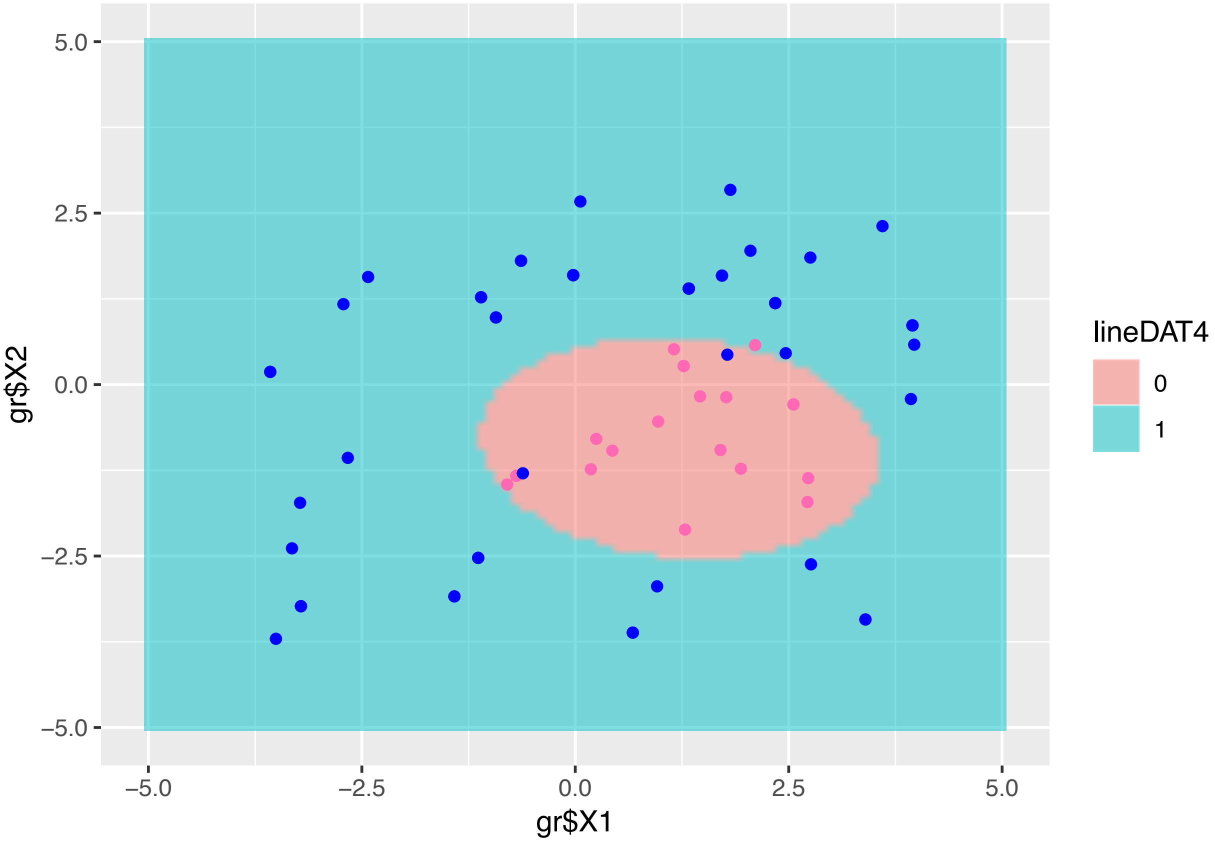
f) (**131 extra credit**) Create 3 bootstrap replicates of the original dataset. Fit the linear model and the 5th order polynomial to each of the bootstrap replicates. Plot class predictions on the grid of values for each of both linear and 5th order fits, from each of the bootstrap samples. There should be six plots total. Discuss what you see in the context of your answer to the previous question.

```
set.seed(1)
sampDAT <- NULL

sampDAT <- sample_n(dat, 72, replace = TRUE)
glmDAT4 <- glm(Y ~ poly(X1, X2, degree = 2), data = as.data.frame(sampDAT),
    family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

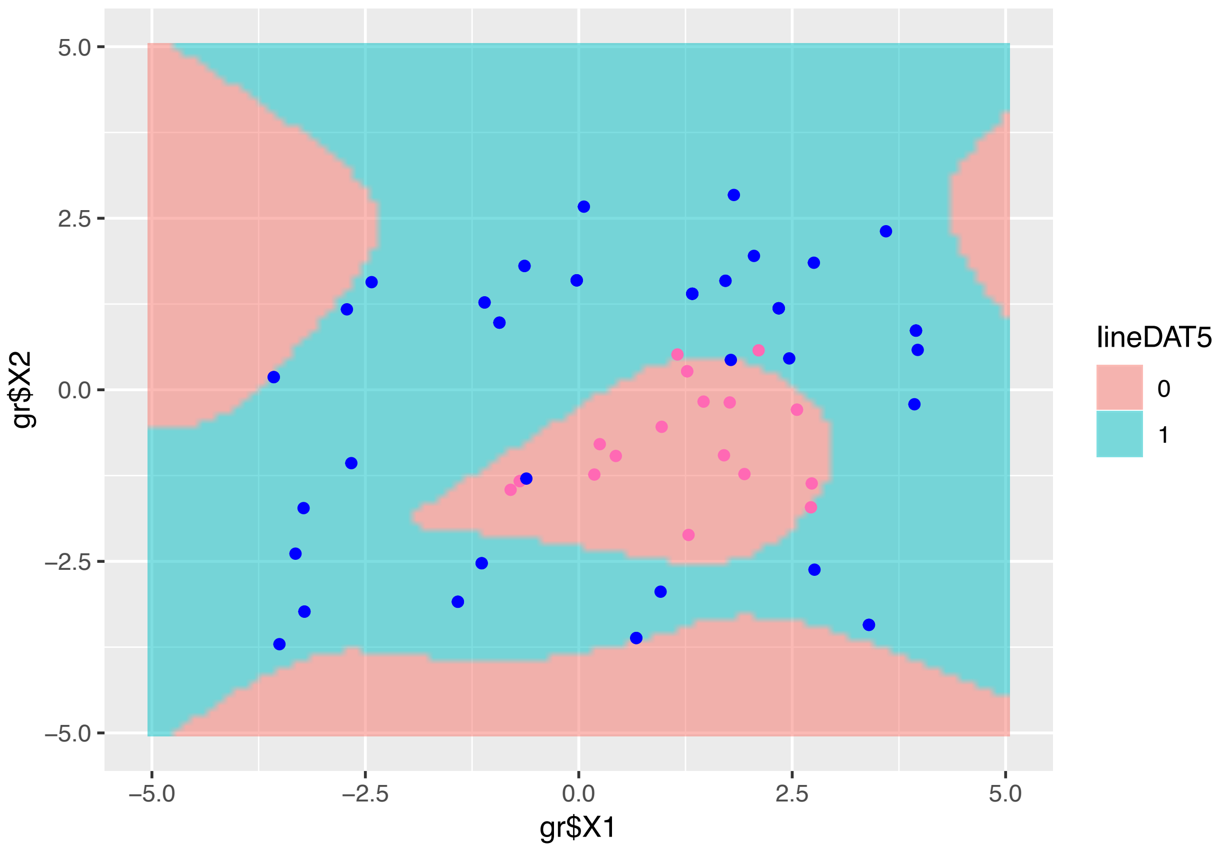
```
preDAT4 <- predict(glmDAT4, gr, type = "response")
lineDAT4 <- as.factor(ifelse(preDAT4 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT4),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```



```r
glmDAT5 <- glm(Y ~ poly(X1, X2, degree = 5), data = sampDAT,
    family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
preDAT5 <- predict(glmDAT5, gr, type = "response")
lineDAT5 <- as.factor(ifelse(preDAT5 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT5),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```

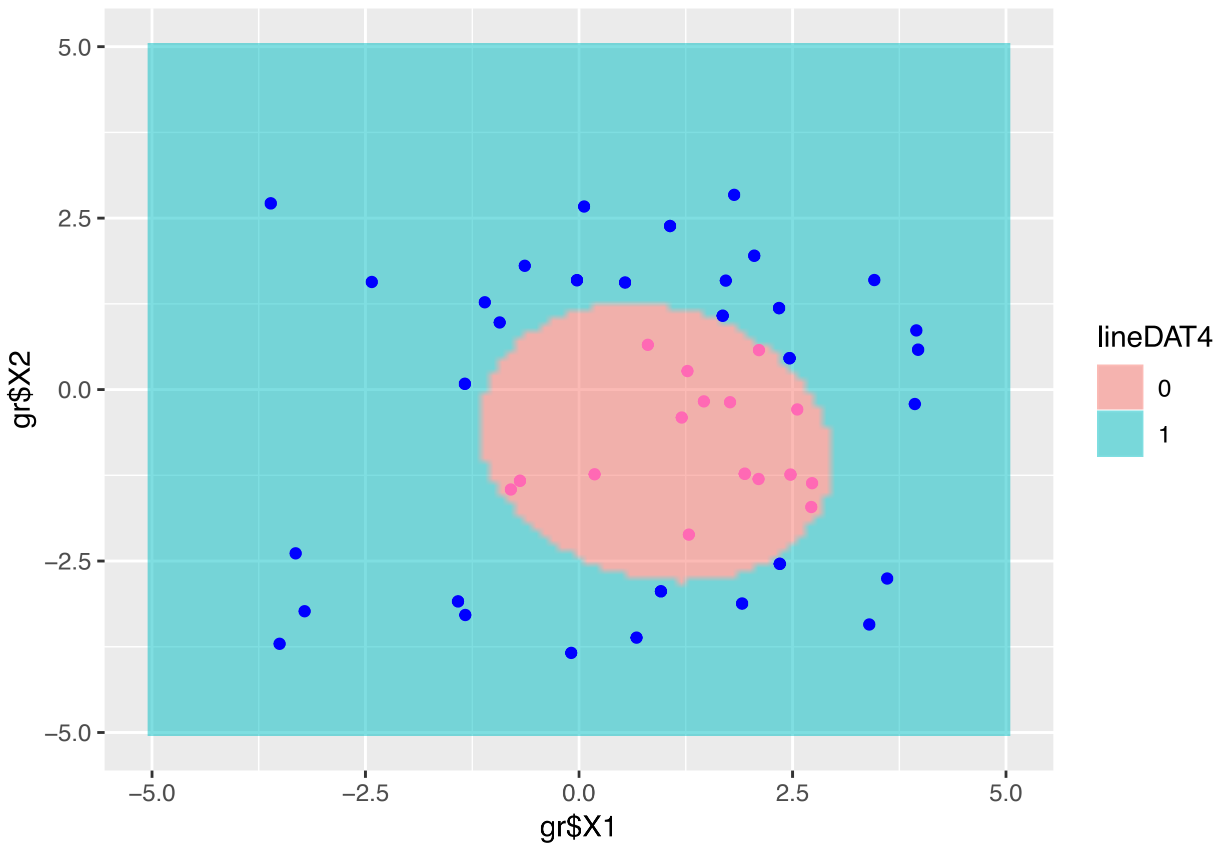


```
sampDAT <- sample_n(dat, 72, replace = TRUE)
glmDAT4 <- glm(Y ~ poly(X1, X2, degree = 2), data = as.data.frame(sampDAT),
    family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
preDAT4 <- predict(glmDAT4, gr, type = "response")
lineDAT4 <- as.factor(ifelse(preDAT4 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT4),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```

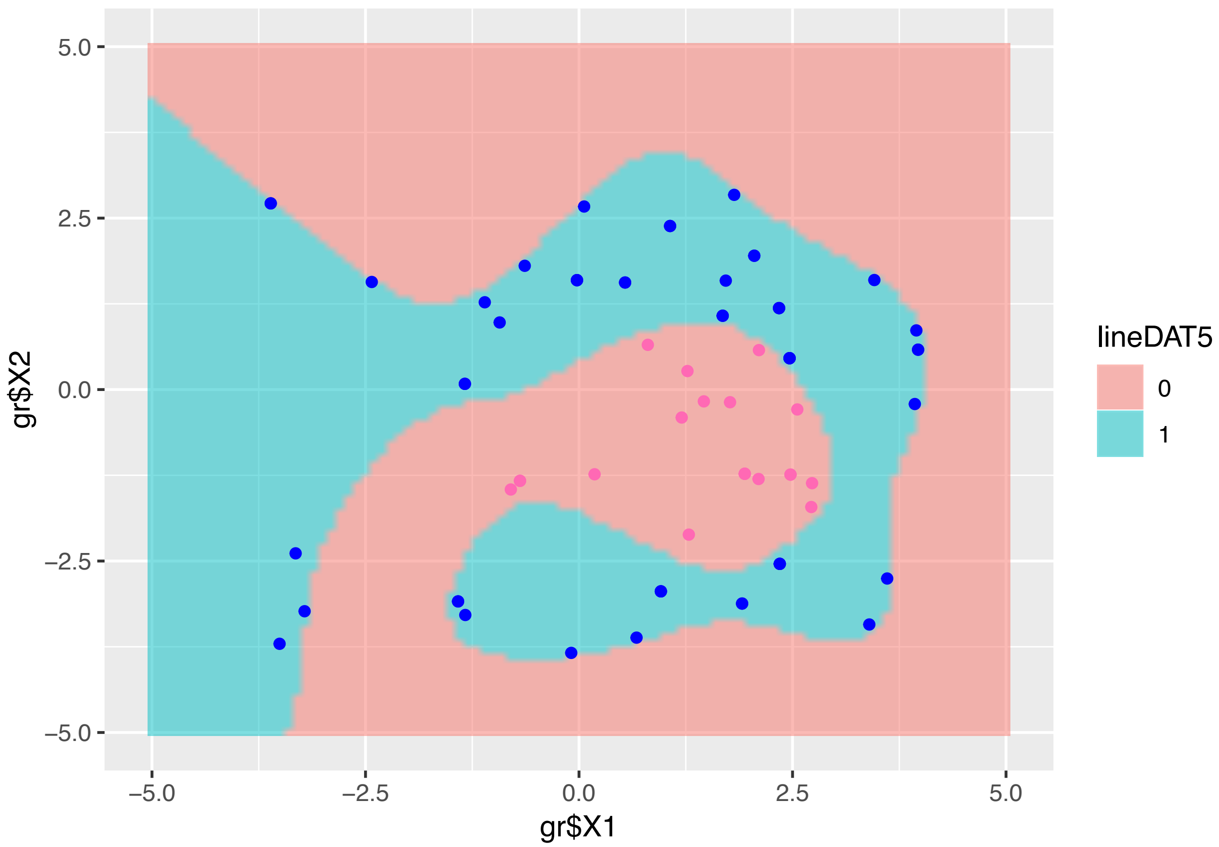


```
glmDAT5 <- glm(Y ~ poly(X1, X2, degree = 5), data = sampDAT,
    family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

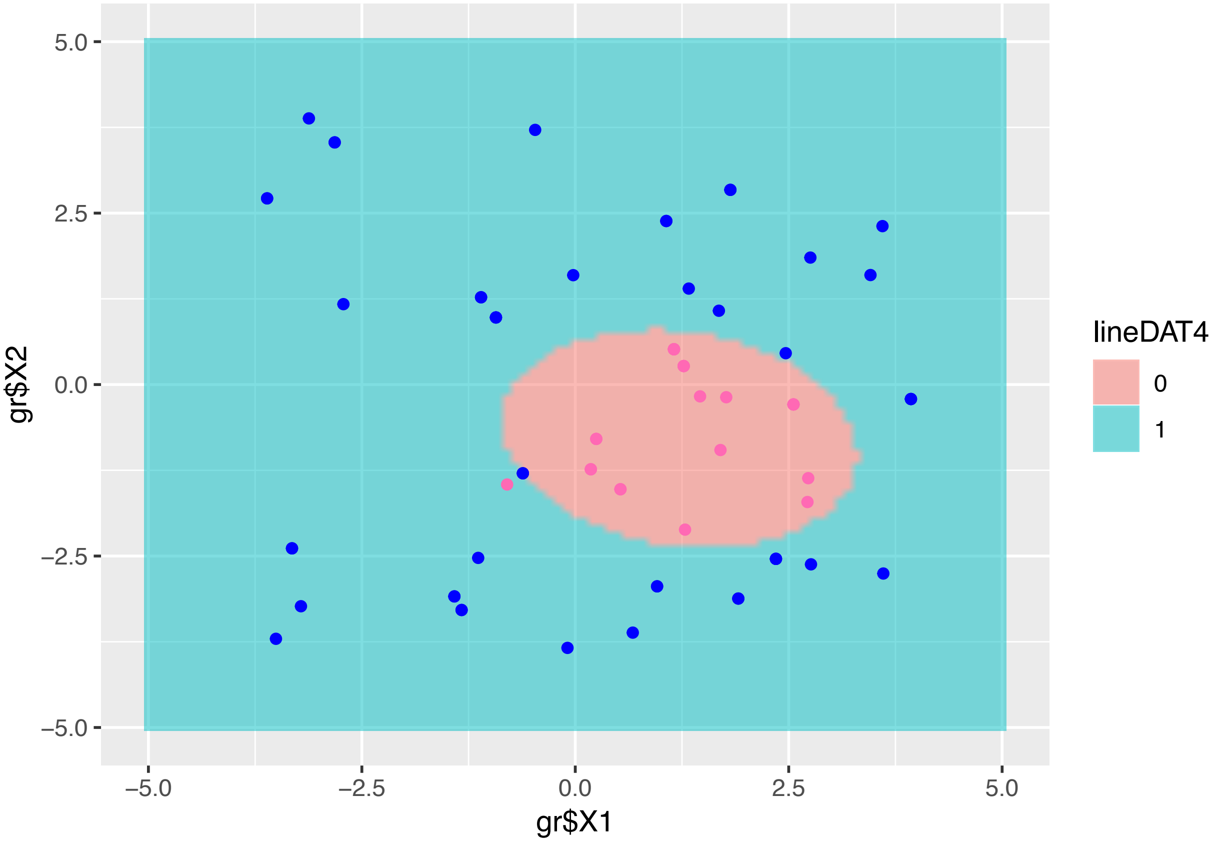
```
preDAT5 <- predict(glmDAT5, gr, type = "response")
lineDAT5 <- as.factor(ifelse(preDAT5 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT5),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```



```
sampDAT <- sample_n(dat, 72, replace = TRUE)
glmDAT4 <- glm(Y ~ poly(X1, X2, degree = 2), data = as.data.frame(sampDAT),
    family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

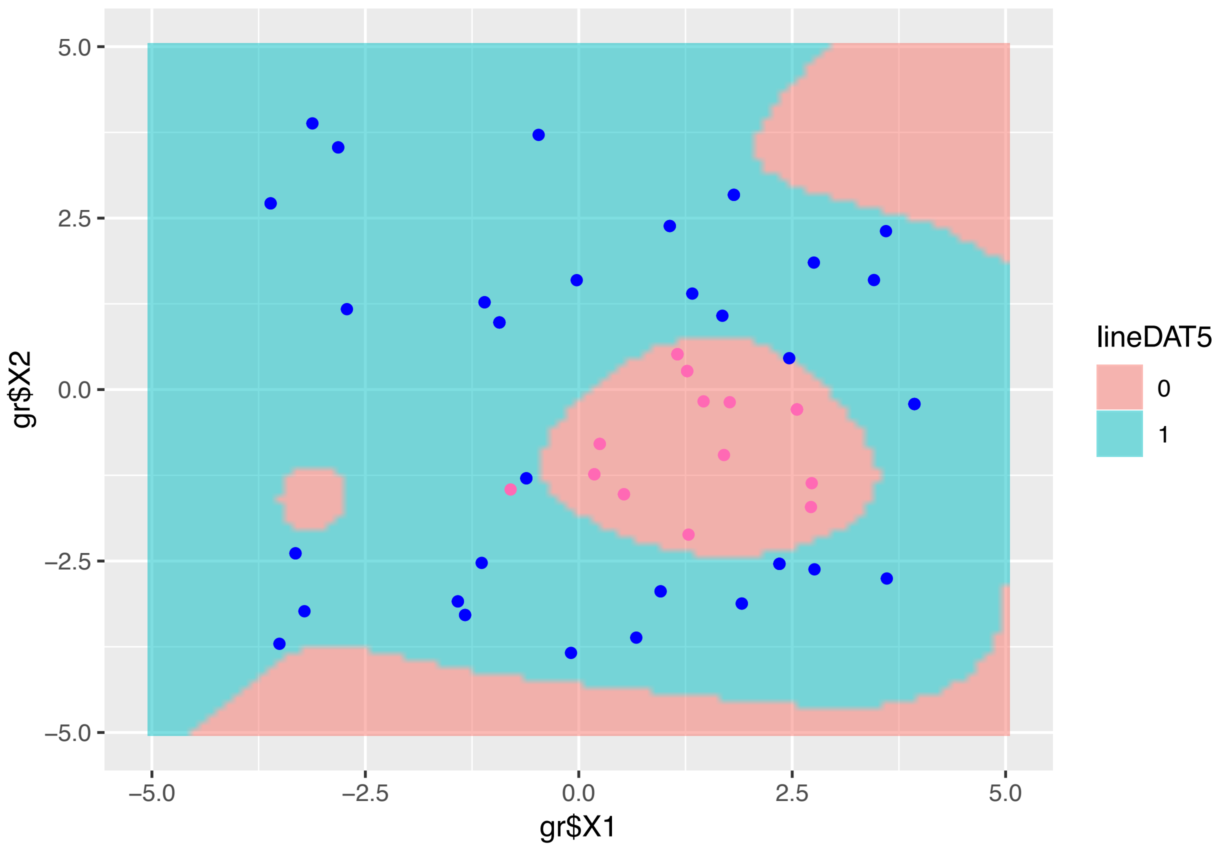
```
preDAT4 <- predict(glmDAT4, gr, type = "response")
lineDAT4 <- as.factor(ifelse(preDAT4 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT4),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```



```
glmDAT5 <- glm(Y ~ poly(X1, X2, degree = 5), data = sampDAT,
    family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
preDAT5 <- predict(glmDAT5, gr, type = "response")
lineDAT5 <- as.factor(ifelse(preDAT5 <= 0.5, 0, 1))
ggplot(gr, aes(gr$X1, gr$X2)) + geom_raster(aes(fill = lineDAT5),
    alpha = 0.5) + geom_point(aes(sampDAT$X1, sampDAT$X2), data = sampDAT,
    col = ifelse(sampDAT$Y == 0, "hotpink", 4))
```



The 5-degree models have boundaries that reflect the typical distance between blue and pink points, continuing to hug the blue points very closely, even in the absence of pink points at the other side. The shape of these models changes with each bootstrapped sample, showing very little consistency in the predictions. Hence we see the high variance that comes with overfitting, and excessive deliniation.
On the other hand, the 2-degree models offer great consistency, each plot looks highly similar to the others, and hence we believe this model is a better fit for the population.

4. Predicting insurance policy purchases

This question involves the use of the “Caravan” data set, which contains 5822 real customer records. Each record consists of 86 variables, containing sociodemographic data (variables 1-43) and product ownership (variables 44-86), grouped by zip code. In this problem we will focus on predicted the variable “Purchase” which indicates whether the customer purchased a caravan insurance policy. For more information see http://www.liacs.nl/~putten/library/cc2000/data.html.

a) When you load the “ISLR” library, the variable `Caravan` is automatically loaded into your environment. Split `Carvan` into a training set consisting of the first 1000 observations and a test set consisting of the remaining observations.

```
library(ISLR)
caravan_train <- head(Caravan, 1000)
caravan_test <- tail(Caravan, -1000)
print(dim(caravan_train))
```

```
## [1] 1000   86
```

```
print(dim(caravan_test))
```

```
## [1] 4822   86
```

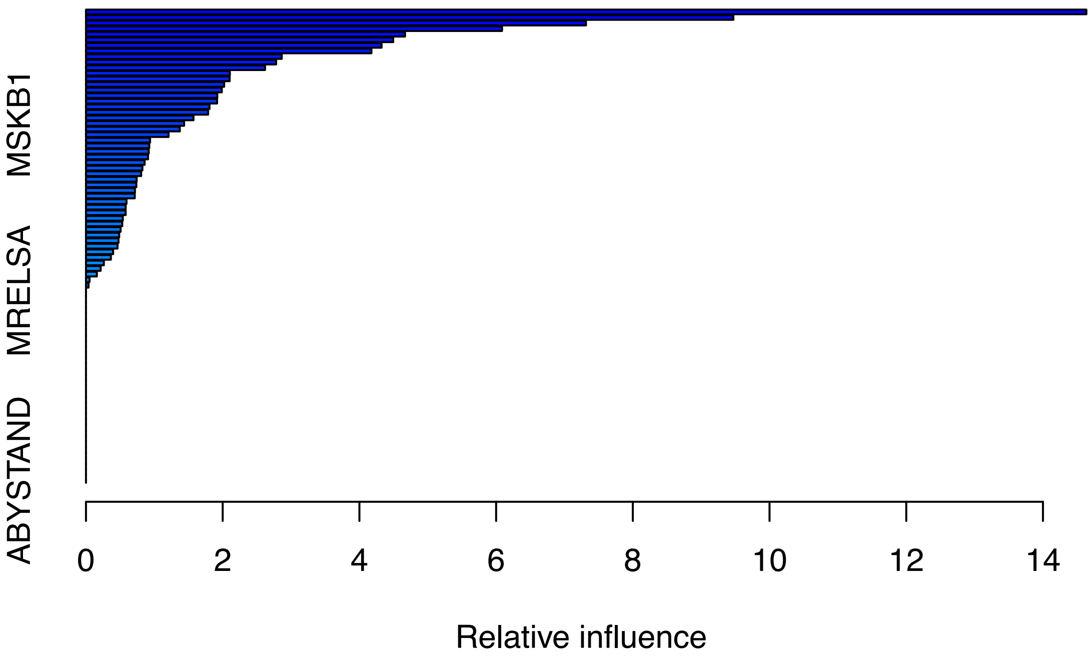
b) Fit a boosting model to the training set with `Purchase` as the response and the other variables as predictors. Use the `gbm` to fit a 1,000 tree boosted model and set the shrinkage value of 0.01. Which predictors appear to be the most important (Hint: use the `summary` function)?

```
set.seed(1)
boost.caravan = gbm(ifelse(Purchase == "Yes", 1, 0) ~ ., data = caravan_train,
    n.trees = 1000, shrinkage = 0.01, distribution = "bernoulli")
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 50: PVRAAUT has no variation.

## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 71: AVRAAUT has no variation.
```

```
summary(boost.caravan)
```



```
##               var     rel.inf
## PPERSAUT PPERSAUT 14.63504779
## MKOOPKLA MKOOPKLA  9.47091649
## MOPLHOOG MOPLHOOG  7.31457416
## MBERMIDD MBERMIDD  6.08651965
## PBRAND     PBRAND  4.66766122
## MGODGE     MGODGE  4.49463264
## ABRAND     ABRAND  4.32427755
## MINK3045 MINK3045  4.17590619
## MOSTYPE   MOSTYPE  2.86402583
## PWAPART   PWAPART  2.78191075
## MAUT1       MAUT1  2.61929152
## MBERARBG MBERARBG  2.10480508
## MSKA         MSKA  2.10185152
## MAUT2       MAUT2  2.02172510
## MSKC         MSKC  1.98684345
## MINKGEM   MINKGEM  1.92122708
## MGODPR     MGODPR  1.91777542
## MBERHOOG MBERHOOG  1.80710618
## MGODOV     MGODOV  1.78693913
## PBYSTAND PBYSTAND  1.57279593
## MSKB1       MSKB1  1.43551401
## MFWEKIND MFWEKIND  1.37264255
## MRELGE     MRELGE  1.20805179
## MOPLMIDD MOPLMIDD  0.93791970
## MINK7512 MINK7512  0.92590720
## MINK4575 MINK4575  0.91745993
## MGODRK     MGODRK  0.90765539
## MFGEKIND MFGEKIND  0.85745374
## MZPART     MZPART  0.82531066
## MRELOV     MRELOV  0.80731252
## MINKM30   MINKM30  0.74126812
## MHKOOP     MHKOOP  0.73690793
## MZFONDS   MZFONDS  0.71638323
## MAUT0       MAUT0  0.71388052
## MHHUUR     MHHUUR  0.59287247
## APERSAUT APERSAUT  0.58056986
## MOSHOOFD MOSHOOFD  0.58029563
## MSKB2       MSKB2  0.53885275
## PLEVEN     PLEVEN  0.53052444
## MINK123M MINK123M  0.50660603
## MBERARBO MBERARBO  0.48596479
## MGEMOMV   MGEMOMV  0.47614792
## PMOTSCO   PMOTSCO  0.46163590
## MSKD         MSKD  0.39735297
## MBERBOER MBERBOER  0.36417546
## MGEMLEEF MGEMLEEF  0.26166240
## MFALLEEN MFALLEEN  0.21448118
## MBERZELF MBERZELF  0.15906143
## MOPLLAAG MOPLLAAG  0.05263665
## MAANTHUI MAANTHUI  0.03766014
## MRELSA     MRELSA  0.00000000
## PWABEDR   PWABEDR  0.00000000
## PWALAND   PWALAND  0.00000000
## PBESAUT   PBESAUT  0.00000000
## PVRAAUT   PVRAAUT  0.00000000
## PAANHANG PAANHANG  0.00000000
## PTRACTOR PTRACTOR  0.00000000
## PWERKT     PWERKT  0.00000000
## PBROM       PBROM  0.00000000
## PPERSONG PPERSONG  0.00000000
## PGEZONG   PGEZONG  0.00000000
## PWAOREG   PWAOREG  0.00000000
## PZEILPL   PZEILPL  0.00000000
## PPLEZIER PPLEZIER  0.00000000
## PFIETS     PFIETS  0.00000000
## PINBOED   PINBOED  0.00000000
## AWAPART   AWAPART  0.00000000
## AWABEDR   AWABEDR  0.00000000
## AWALAND   AWALAND  0.00000000
## ABESAUT   ABESAUT  0.00000000
## AMOTSCO   AMOTSCO  0.00000000
## AVRAAUT   AVRAAUT  0.00000000
## AAANHANG AAANHANG  0.00000000
## ATRACTOR ATRACTOR  0.00000000
## AWERKT     AWERKT  0.00000000
## ABROM       ABROM  0.00000000
## ALEVEN     ALEVEN  0.00000000
## APERSONG APERSONG  0.00000000
## AGEZONG   AGEZONG  0.00000000
## AWAOREG   AWAOREG  0.00000000
## AZEILPL   AZEILPL  0.00000000
## APLEZIER APLEZIER  0.00000000
## AFIETS     AFIETS  0.00000000
## AINBOED   AINBOED  0.00000000
## ABYSTAND ABYSTAND  0.00000000
```

c) Now fit a random forest model to the same training set from the previous problem. Set `importance=TRUE` but use the default parameter values for all other inputs to the `randomForest` function. Print the random forest object returned by the random forest function. What is the out-of-bag estimate of error? How many variables were subsampled at each split in the trees? How many trees were used to fit the data? Look at the variable importance. Is the order of important variables similar for both boosting and random forest models?

```r
set.seed(1)
rf.caravan = randomForest(Purchase ~ ., data = caravan_train,
    importance = TRUE)
rf.caravan
```

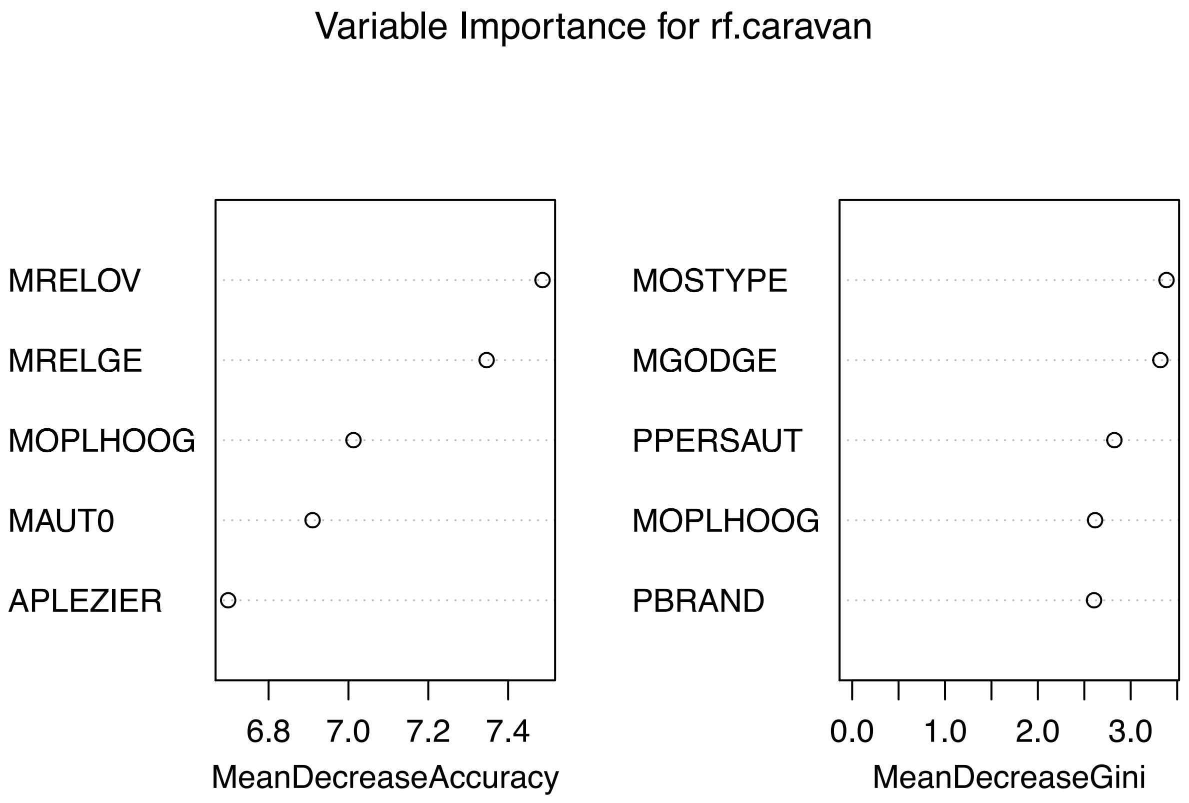
```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = caravan_train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 6.1%
## Confusion matrix:
##      No Yes class.error
## No  937   4 0.004250797
## Yes  57   2 0.966101695
```

The out-of-bag estimate of error is 6.1%. The number of variables subsampled at each split is 9. 500 trees were used to fit the data.

```r
# Looking at the importance variable
importance(rf.caravan)
```

```
##                   No          Yes MeanDecreaseAccuracy MeanDecreaseGini
## MOSTYPE   2.45154591  2.244153819            3.0134507     3.3855613917
## MAANTHUI  1.60618826 -0.290842854            1.4538716     0.6143667918
## MGEMOMV   3.38493902 -2.110795715            2.9068779     1.0071021582
## MGEMLEEF  2.55973289  1.473727894            2.9719722     1.0794485718
## MOSHOOFD  3.36995556  4.426788277            4.4229915     1.9738057922
## MGODRK    4.65845598  0.737034753            4.7409588     1.2387385868
## MGODPR    4.47209929  0.908551685            4.4602577     2.3366146232
## MGODOV    2.50722507 -0.487383289            2.3893495     1.5670299537
## MGODGE    1.28224362  5.639817197            3.4991594     3.3194250040
## MRELGE    7.36265693  0.269752923            7.3461530     1.9588421148
## MRELSA    4.70194669  0.452166675            4.6543739     1.3126338786
## MRELOV    7.46288700  0.496195580            7.4861695     1.7935856470
## MFALLEEN  6.11338942 -0.244730246            5.8823286     1.5595153222
## MFGEKIND  3.55145579 -0.416353528            3.3704460     2.0285631796
## MFWEKIND  1.77176524 -1.029635115            1.5662212     2.2381811379
## MOPLHOOG  5.11644375  6.411930730            7.0125794     2.6160759960
## MOPLMIDD  6.38345887 -0.367223968            6.1696129     2.3197585934
## MOPLLAAG  5.03880399 -0.204079688            4.9848255     1.7436716439
## MBERHOOG  2.84911348  0.640220247            3.1271066     1.8173572113
## MBERZELF  1.81181008  1.029947948            2.0175598     0.7595009487
## MBERBOER  0.24276831  0.500299080            0.3745657     0.4528499609
## MBERMIDD  4.58228798  4.226428926            5.6409776     2.4014556209
## MBERARBG  4.07720716 -1.126025271            3.7824374     2.3930576749
## MBERARBO  4.70824017  0.507995051            4.8136532     2.0635646903
## MSKA      2.85153495  2.869038666            3.7018567     2.0505017986
## MSKB1     2.02495419  2.811995244            2.7990426     1.9649351466
## MSKB2     3.63933294 -1.140039549            3.2819700     1.9153830076
## MSKC      1.62816010  2.248348320            2.2101073     2.2403166260
## MSKD      1.18991796  0.204816577            1.2780817     1.0344871416
## MHHUUR    2.17495496  4.339104961            3.4333327     2.0183314106
## MHKOOP    2.14404191  4.794262902            3.4505946     2.1058102134
## MAUT1     0.85874384 -0.745278178            0.7330822     1.9019071060
## MAUT2     2.49271799  1.897193762            2.9121809     1.7206310469
## MAUT0     7.06300777 -0.860204512            6.9101043     1.7461401474
## MZFONDS   5.99837468  0.482100327            6.3134388     2.0298872877
## MZPART    5.87997698  0.265375941            5.7507661     1.8976048036
## MINKM30   3.30483243  1.053640398            3.4325492     1.7861887910
## MINK3045  1.42380610  0.694473622            1.5478581     2.1300576007
## MINK4575  2.22746442  1.000307996            2.4605940     1.6754786425
## MINK7512  6.20976748  1.628500118            6.5538078     1.8321676293
## MINK123M -0.98857123  0.530151554           -0.7682253     0.3776091219
## MINKGEM   2.42617883  1.077510437            2.7107917     1.4937252558
## MKOOPKLA  4.16470811  3.618032553            5.1828355     2.4658506352
## PWAPART  -3.02481682  4.903157985           -1.2806744     2.0032603259
## PWABEDR   0.44448411 -1.001001503            0.2224323     0.1697765794
## PWALAND   1.50266675 -1.001001503            1.2555966     0.0893057652
## PPERSAUT  2.09263491  5.367475427            3.4587027     2.8240358520
## PBESAUT   0.00000000  0.000000000            0.0000000     0.0110000000
## PMOTSCO  -1.58856509 -0.914425433           -1.8397010     0.8164202523
## PVRAAUT   0.00000000  0.000000000            0.0000000     0.0000000000
## PAANHANG -0.05105565 -1.001001503           -0.2110083     0.2010860321
## PTRACTOR  1.34110446  0.000000000            1.3315425     0.2069953044
## PWERKT    0.00000000  0.000000000            0.0000000     0.0001025641
## PBROM     5.14931162 -1.728419270            4.5362542     0.4886392196
## PLEVEN   -0.09157408  0.005157702           -0.1001994     0.6850565571
## PPERSONG  0.00000000  0.000000000            0.0000000     0.0050000000
## PGEZONG  -0.99997680 -0.660121070           -1.1215069     0.7265914692
## PWAOREG   3.50037985  2.555469067            3.7534286     0.8830211907
## PBRAND   -3.46618600  3.065705615           -2.3895592     2.6050140703
## PZEILPL   0.00000000  0.000000000            0.0000000     0.3101046631
## PPLEZIER  3.41283872  5.519866934            5.3475104     1.9481523655
## PFIETS   -1.60373164 -1.001001503           -1.6410071     0.1434584923
## PINBOED   0.33600315 -1.415913679           -0.2448524     0.0622960673
## PBYSTAND  1.62312319  1.116455430            1.7928666     0.7890283949
## AWAPART   0.07474160  5.113356495            2.1664576     1.2282489984
## AWABEDR   1.64885562  0.000000000            1.6533327     0.0886151003
## AWALAND   1.30485240  1.001001503            1.5086644     0.0799057637
## APERSAUT  0.90083143  0.166343909            0.8674609     2.0385345213
## ABESAUT   0.00000000  0.000000000            0.0000000     0.0094285714
## AMOTSCO   0.75583812 -1.979465167            0.3016325     0.9055590600
## AVRAAUT   0.00000000  0.000000000            0.0000000     0.0000000000
## AAANHANG -2.22662144 -1.984140947           -2.5742611     0.1799960163
## ATRACTOR  1.90298837  0.000000000            1.9031806     0.0827505956
## AWERKT    0.00000000  0.000000000            0.0000000     0.0000000000
## ABROM     4.58525585 -2.944440951            3.5474461     0.3956193901
## ALEVEN   -0.35600815  0.096028467           -0.3229689     0.2689196899
## APERSONG  0.00000000  0.000000000            0.0000000     0.0022666667
## AGEZONG   0.01725175 -2.121294784           -0.5451891     0.4079733602
## AWAOREG   3.42266626  1.976805350            3.9407140     0.8752068226
## ABRAND   -0.86663442  0.401244715           -0.7153231     1.9137179129
## AZEILPL   0.00000000  0.000000000            0.0000000     0.3081311861
## APLEZIER  3.18370968  7.770210840            6.6986818     1.5981874502
## AFIETS   -1.68152736 -0.227241349           -1.6378835     0.2249215377
## AINBOED   0.29965226 -1.327312597           -0.4106905     0.0644346995
## ABYSTAND  1.03929978  1.529758423            1.4094141     0.5114897627
```

```
varImpPlot(rf.caravan, sort = T, main = "Variable Importance for rf.caravan",
    n.var = 5)
```



The order of important variables is not the same for Random Forest and Boosting models.

d) Use both models to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20 %. Print the confusion matrix for both the boosting and random forest models. In the random forest model, what fraction of the people predicted to make a purchase do in fact make one? Note: use the `predict` function with `type="prob"` for random forests and `type="resonpse"` for the boosting algorithm.

```
# Predictions with Boosting model
yhat.boost = predict(boost.caravan, newdata = caravan_test, n.trees = 1000,
    type = "response")
yhat.boostprob = as.factor(ifelse(yhat.boost >= 0.2, "Yes", "No"))

# Confusion matrix
boost.err = table(predicted = yhat.boostprob, true = caravan_test$Purchase)
test.boost.err = 1 - sum(diag(boost.err))/sum(boost.err)
test.boost.err
```

```
## [1] 0.07859809
```

Of the people predicted to make a purchase, 92.15% of them actually make the purchase.

```
# Predictions with Random Forest Model
yhat.rf = predict(rf.caravan, newdata = caravan_test, type = "prob")
yhat.rfprob = as.factor(ifelse(yhat.rf[, "Yes"] >= 0.2, "Yes",
    "No"))

# Confusion matrix
rf.err = table(predicted = yhat.rfprob, true = caravan_test$Purchase)
```

```r
test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)
test.rf.err
```

```
## [1] 0.1038988
```

Of the people predicted to make a purchase, 89.62% of them actually make the purchase.

5. An SVMs prediction of drug use

In this problem we return to an analysis of the drug use dataset. Load the drug use data using `read_csv`:

```r
drug_use <- read_csv("drug.csv", col_names = c("ID", "Age", "Gender",
    "Education", "Country", "Ethnicity", "Nscore", "Escore",
    "Oscore", "Ascore", "Cscore", "Impulsive", "SS", "Alcohol",
    "Amphet", "Amyl", "Benzos", "Caff", "Cannabis", "Choc", "Coke",
    "Crack", "Ecstasy", "Heroin", "Ketamine", "Legalh", "LSD",
    "Meth", "Mushrooms", "Nicotine", "Semer", "VSA"))
```

a) Split the data into training and test data. Use a random sample of 1500 observations for the training data and the rest as test data. Use a support vector machine to predict `recent_cannabis_use` using only the subset of predictors between `Age` and `SS` variables as on homework 3. Unlike homework 3, do not bother mutating the features into factors. Use a "radial" kernel and a cost of 1. Generate and print the confusion matrix of the predictions against the test data.

```r
train_index <- sample(nrow(drug_use), 1500)

drug_use <- drug_use %>% mutate(recent_cannabis_use = factor(ifelse(Cannabis >=
    "CL3", "Yes", "No"), levels = c("No", "Yes")))

drug_use_train <- drug_use[train_index, ]
drug_use_test <- drug_use[-train_index, ]

# SVM
svmfit = svm(drug_use_train$recent_cannabis_use ~ ., data = drug_use_train[c((2:13),
    33)], kernel = "radial", cost = 1, scale = FALSE)
drug_pred = predict(svmfit, drug_use_test[c((2:13), 33)], type = "class")

# Confusion Matrix
conf.test = table(predicted = drug_pred, true = drug_use_test$recent_cannabis_use)
conf.test
```

```
##          true
## predicted  No Yes
##       No  138  32
##       Yes  42 173
```

b) Use the `tune` function to perform cross validation over the set of cost parameters: `cost=c(0.001, 0.01, 0.1, 1,10,100)`. What is the optimal cost and corresponding cross validated training error for this model? Print the confusion matrix for the best model. The best model can be found in the `best.model` variable returned by `tune`.

```r
drug_use_train[33]
```

```
## # A tibble: 1,500 x 1
##    recent_cannabis_use
##    <fct>
##  1 No
##   2 Yes
##   3 Yes
##   4 Yes
##   5 No
##   6 Yes
##   7 No
##   8 No
##   9 Yes
## 10 No
## # ... with 1,490 more rows
```

```
set.seed(1)
tune.out = tune(svm, recent_cannabis_use ~ ., data = drug_use_train[c((2:13),
    33)], kernel = "radial", ranges = list(cost = c(0.001, 0.01,
    0.1, 1, 5, 10, 100), scale = FALSE))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost scale
##    0.1 FALSE
##
## - best performance: 0.1893333
##
## - Detailed performance results:
##     cost scale     error dispersion
## 1 1e-03 FALSE 0.4706667 0.03786917
## 2 1e-02 FALSE 0.2086667 0.03665656
## 3 1e-01 FALSE 0.1893333 0.03502380
## 4 1e+00 FALSE 0.1906667 0.04124004
## 5 5e+00 FALSE 0.2000000 0.03030707
## 6 1e+01 FALSE 0.2060000 0.03684469
## 7 1e+02 FALSE 0.2593333 0.03561800
```

```
# Optimal cost is 0.1 Training error for optimal cost model
# is 0.1826667

svmDRUG = svm(recent_cannabis_use ~ ., data = drug_use_train[c((2:13),
    33)], kernel = "radial", cost = 0.1, scale = FALSE)
preDRUG = predict(svmDRUG, drug_use_test[c((2:13), 33)], type = "class")

# Confusion Matrix
conf.test = table(predicted = preDRUG, true = drug_use_test$recent_cannabis_use)
conf.test
```

```
##          true
## predicted  No Yes
##       No  140  30
##       Yes  40 175
```