Biomedical Data Science - Assignment 2

Yile Shi (s2168022)

2022/3/26

Assignment 2

Biomedical Data Science

Due on Tuesday 5th April 2022, 5:00pm

The assignment is marked out of 100 points, and will contribute to 30% of your final mark. Please knit this document in PDF format and submit using the gradescope link on Learn. If you can't knit to PDF directly, knit it to word and you should be able to either convert to PDF or print it and scan to PDF using a scanning app on your phone. If you have any code that doesn't run you won't be able to knit the document so comment it as you might still get some grades for partial code. Clear and reusable code will be rewarded so pay attention to indentation, choice of variable identifiers, comments, error checking, etc. An initial code chunk is provided after each subquestion but create as many chunks as you feel is necessary to make a clear report. Add plain text explanations in between the chunks as and when required and any comments necessary within code chunks to make it easier to follow your code/reasoning.

Problem 1 (27 points)

File wdbc2.csv (available from the accompanying zip folder on Learn) refers to a study of breast cancer where the outcome of interest is the type of the tumour (benign or malignant, recorded in column "diagnosis"). The study collected 30 imaging biomarkers on 569 patients.

Problem 1.a (7 points)

Using package caret, create a data partition so that the training set contains 70% of the observations (set the random seed to 984065 beforehand). Fit both a ridge regression model and a lasso model which uses cross-validation on the training set to diagnose the type of tumour from the 30 biomarkers. Then use a plot to help identify the penalty parameter λ that maximizes the AUC. Note: There is no need to use the prepare.glmnet() function from lab 4, using as.matrix() with the required columns is sufficient.

Solution:

After reading the dataset into R, we find that response variable `diagnosis` consists of strings. Though `glmnet()` function can deal with strings in `diagnosis`, `glm()` function failed to fit regressions on the response variable with strings in 1.c. Therefore, before any further handling, we encoded `diagnosis` with 0 and 1, where 0 represents "malignant" and 1 represents "benign".

We use package `caret` to create training and testing sets with a ratio of 7 : 3. For model reproducibility and comparison, we set the random seed to 984065 beforehand. Since `glmnet()` function requires matrices as its input, we use `as.matrix()` function to transform data tables for 30 biomarkers to matrices.

After data pre-processing, we fit a Ridge regression and a Lasso regression which use cross-validation on the training data. To identify the optimal penalty parameter which maximises the AUC for each model, for each model, we make an auxiliary plot which display the AUC in red with bars corresponding to standard errors. The leftmost dotted line in each plot corresponds to the λ that maximises AUC, i.e. `lambda_min` in the regression object. The right dotted line corresponds to the largest value of λ such that the error is within one standard error from the minimum, i.e. `lambda_1se` in the fitted object.

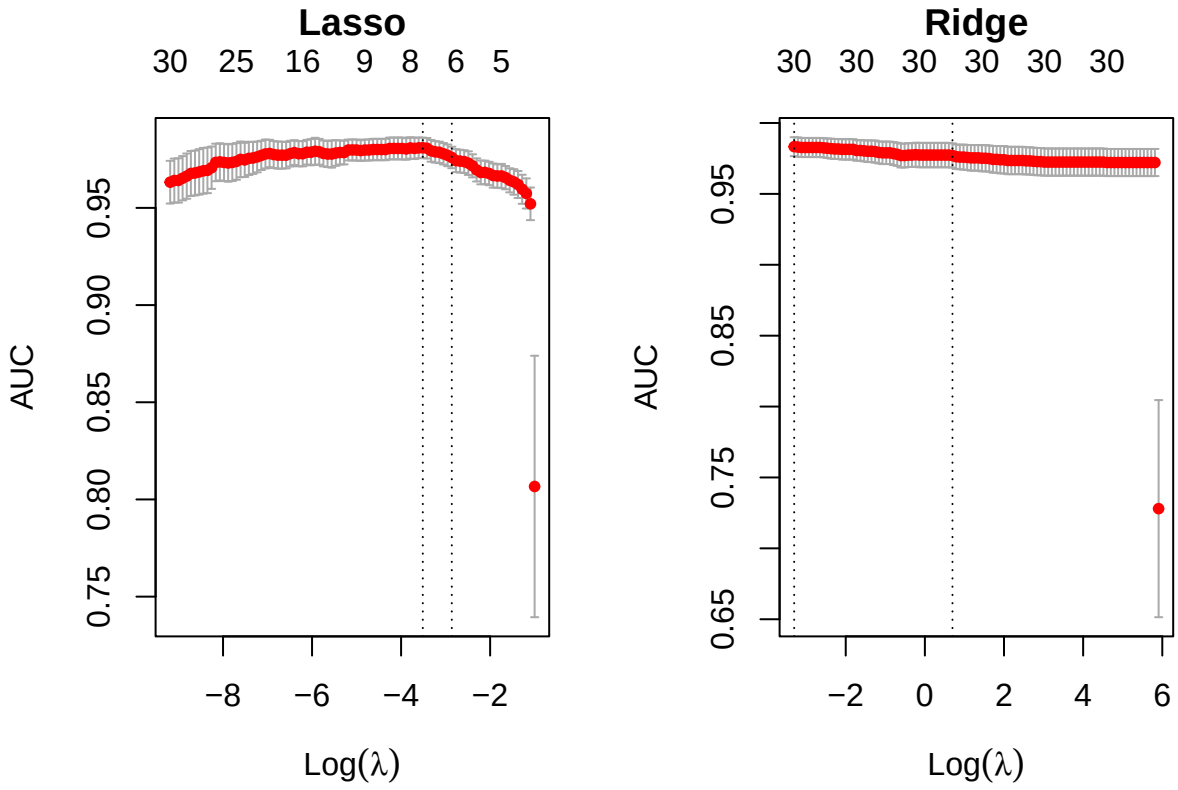
Finally, we report the optimal λs which maximise the AUC for two models:

- Lasso regression: $\lambda = 0.02982835$
- Ridge regression: $\lambda = 0.03677378$

```
# Read dataset into R
wdbc <- fread("assignment2/wdbc2.csv")

# convert strings in 'diagnosis' to numeric
wdbc$diagnosis <- ifelse(wdbc$diagnosis == 'malignant', 1, 0)

# set random seed beforehand
set.seed(984065)

# create training and testing sets
train.idx <- createDataPartition(y = wdbc$diagnosis, p = 0.7)$Resample1
train.wdbc <- wdbc[train.idx, -c(1)]
test.wbdc <- wdbc[-train.idx, -c(1)]

# define x and y matrices in the regression function
train.x <- train.wdbc[, -c(1)]
test.x <- test.wbdc[, -c(1)]
train.x.mat <- as.matrix(train.x)
test.x.mat <- as.matrix(test.x)

train.y.mat <- train.wdbc$diagnosis
test.y.mat <- test.wbdc$diagnosis
```

```
# fit a Ridge regression and a Lasso regression using CV on training set
set.seed(984065)
fit.cv.lasso <- cv.glmnet(train.x.mat, train.y.mat,
                          family = "binomial", type.measure = "auc")
fit.cv.ridge <- cv.glmnet(train.x.mat, train.y.mat, alpha = 0,
                          family = "binomial", type.measure = "auc")
```

```
# make plots to find the optimal lambdas that maximise the AUC
par(mfrow = c(1,2), mar = c(4,4,5,2))
plot(fit.cv.lasso, main = "Lasso")
plot(fit.cv.ridge, main = "Ridge")
```




```
# the values of optimal lambdas
cat("The optimal lambda in Lasso regression: ", fit.cv.lasso$lambda.min, "\n")
```

```
## The optimal lambda in Lasso regression:  0.02982835
```

```
cat("The optimal lambda in Ridge regression: ", fit.cv.ridge$lambda.min)
```

```
## The optimal lambda in Ridge regression:  0.03677378
```

Problem 1.b (2 points)

Create a data table that for each value of 'lambda.min' and 'lambda.1se' for each model fitted in problem 1.a reports: * the corresponding AUC, * the corresponding model size. Use 3 significant digits for floating point values and comment on these results. Hint: The AUC values are stored in the field called 'cvm'.

Solution:

We construct a data frame to display the value of λ in each model and corresponding AUC and model size, using 3 significant digits on floating point values. (The `index` attribute of regression stores the indices of both `lambda_min` and `lambda_1se`.)

From the results, we draw the following conclusions:

1. Low values of λ except Ridge regression using `lambda_1se` indicates the parameters do not get penalized hard.
2. Both Lasso regression and Ridge regression using `lambda_min` have higher AUC scores than corresponding models using `lambda_1se`. This is reasonable as `lambda_min` maximises the AUC.
3. Using same kind of penalty parameters, Lasso regression has smaller values for both λ and AUC than Ridge regression. In terms of AUC, we may consider that Ridge regression with `lambda_min` preforms best on the training set (0.983), out of 4 models.
4. The model sizes of Lasso regression models, 8 and 6, are significantly smaller than the model sizes of Ridge regression models, which keep constant at 30. This is because Ridge regression shrinks penalized coefficients near 0 while Lasso regression shrinks them to 0, based on the penalty parameter λ.

```
# now we create the required data table
rep.dt <- data.table(model = c("Lasso(lambda.min)", "Lasso(lambda.1se)",
                               "Ridge(lambda.min)", "Ridge(lambda.1se)"),
                     lambda = c(signif(fit.cv.lasso$lambda.min, 3),
                                signif(fit.cv.lasso$lambda.1se, 3),
                                signif(fit.cv.ridge$lambda.min, 3),
                                signif(fit.cv.ridge$lambda.1se, 3)),
                     AUC = c(signif(fit.cv.lasso$cvm[fit.cv.lasso$index[1]], 3),
                             signif(fit.cv.lasso$cvm[fit.cv.lasso$index[2]], 3),
                             signif(fit.cv.ridge$cvm[fit.cv.ridge$index[1]], 3),
                             signif(fit.cv.ridge$cvm[fit.cv.ridge$index[2]], 3)),
                     modelsize = c(fit.cv.lasso$nzero[fit.cv.lasso$index[1]],
                                   fit.cv.lasso$nzero[fit.cv.lasso$index[2]],
                                   fit.cv.ridge$nzero[fit.cv.ridge$index[1]],
                                   fit.cv.ridge$nzero[fit.cv.ridge$index[2]]))
rep.dt
```

```
##                model lambda   AUC modelsize
## 1: Lasso(lambda.min) 0.0298 0.981         8
## 2: Lasso(lambda.1se) 0.0572 0.976         6
## 3: Ridge(lambda.min) 0.0368 0.983        30
## 4: Ridge(lambda.1se) 2.0100 0.977        30
```

Problem 1.c (7 points)

Perform both backward (we'll later refer to this as model B) and forward (model S) stepwise selection on the same training set derived in problem 1.a. Report the variables selected and their standardized regression coefficients in decreasing order of the absolute value of their standardized regression coefficient. Discuss the results and how the different variables entering or leaving the model influenced the final result.

Solution:

To perform backward and forward stepwise selection, we pre-define a full model including all 30 biomarkers and a null model with only intercept term.

We use `stepAIC()` function to perform both backward and forward stepwise selection and use `lm.beta()` function in `lm.beta` package to get the standardized regression coefficients of selected variables. Note that since the number of variables is large, we set `trace` option in `stepAIC` to 'FALSE' and hide the selection steps to save space.

Finally, for each model, we report the selected variables with their standardized coefficients in decreasing order of the absolute values of the standardized regression coefficients.

We can observe that besides the intercepts, most variables selected in both backward and forward stepwise method are same, though their regression coefficients are different. This suggests that these biomarkers do have significant effects on the outcome of interest, the type of the tumour. Specifically, these biomarkers are: `texture.worst`, `radius.stderr`, `smoothness.worst`, `radius`, `concavity.worst`, `area.worst`, `compactness.worst`, `perimeter`, `radius.worst` and `texture.stderr`.

We consider the performance of a stepwise selection model based on its AIC. Generally, the lower AIC, the better performance of the model. We discuss the influence of entering or leaving variables on the model result in terms of AIC:

1. For backward stepwise selection, we start with the full model including all variables, together with an initial AIC. We select and drop insignificant variables and the AIC reduces until it reaches the minimum. Note that dropping variables after AIC meets the minimum will increase AIC in backward stepwise method.
2. For forward stepwise selection, on the contrary, we start with a null model with intercept only. Significant variables enter the model and improve model performance by reducing AIC until we reach the optimal AIC. After that, adding redundant variables increases the AIC of the model.

```
# define the full model and null model
full.model <- glm(diagnosis ~ ., data = train.wdbc, family = "binomial")
null.model <- glm(diagnosis ~ 1, data = train.wdbc, family = "binomial")
```

```
# backward stepwise selection
model.B <- stepAIC(full.model, scope = list(lower = null.model),
                   direction = "backward", trace = FALSE)

# forward stepwise selection
model.S <- stepAIC(null.model, scope = list(upper = full.model),
                   direction = "forward", trace = FALSE)
```

```
# selected variables with standardized regression coefficients in model.B
# get the standardized coefficients
std.coef.B <- lm.beta(model.B)$standardized.coefficients
```

```r
# order the coefficients in decreasing order of their absolute values
std.coef.B <- std.coef.B[order(abs(std.coef.B), decreasing = TRUE)]

cat("selected variables in model.B: \n")
```

```
## selected variables in model.B:
```

```r
std.coef.B
```

```
##        radius.worst          area.worst           perimeter     concavity.worst
##           53.047377          -40.610055          -18.500646            8.472730
##       concavepoints              radius       radius.stderr       texture.worst
##            8.398555            7.378246            7.371562            5.605099
##   compactness.worst concavepoints.worst      texture.stderr    smoothness.worst
##           -5.320776           -3.831640           -3.722656            2.008449
##         (Intercept)
##            0.000000
```

```r
# selected variables with standardized regression coefficients in model.S
# get the standardized coefficients
std.coef.S <- lm.beta(model.S)$standardized.coefficients

# order the coefficients in decreasing order of their absolute values
std.coef.S <- std.coef.S[order(abs(std.coef.S), decreasing = TRUE)]

cat("selected variables in model.S: \n")
```

```
## selected variables in model.S:
```

```r
std.coef.S
```

```
##        area.worst      radius.worst   perimeter.worst         perimeter
##        -44.347062         39.117707         16.396414        -13.329464
##     radius.stderr compactness.worst            radius         concavity
##         10.235291         -5.930017          5.544839          5.364296
##     texture.worst  perimeter.stderr   concavity.worst    texture.stderr
##          4.932050         -4.761290          4.299930         -3.028769
##  smoothness.worst       area.stderr       (Intercept)
##          2.661371          2.278462          0.000000
```

Problem 1.d (3 points)

Compare the goodness of fit of model B and model S in an appropriate way.

Solution:

Here we consider using AIC as an appropriate criterion to compare the goodness-of-fit to the training data of two models. As mentioned before, in general, model with lower AIC value has better performance. Thus, according to the following result, we consider the backward stepwise selection, i.e. model.B, performs better than the forward stepwise method.

We also apply χ^2 test to both methods to compare their model significance to check our conclusions drawn from AIC values. As we expect, model.B has a lower χ^2 statistics than model.S, indicating backward stepwise selection is more significant than forward stepwise selection, which is consistent with our previous conclusions.

Note that if we look at the number of parameters in each model from 1.(c), we observe model.B has fewer parameters than model.S. With this in mind, we are more favour of backward stepwise selection as it is a more efficient model with fewer parameters but better performance.

```
data.frame(model.B = c(model.B$aic,
                       pchisq(model.B$null.deviance - model.B$deviance,
                              df = model.B$df.null - model.B$df.residual,
                              lower.tail = FALSE)),
           model.S = c(model.S$aic,
                       pchisq(model.S$null.deviance - model.S$deviance,
                              df = model.S$df.null - model.S$df.residual,
                              lower.tail = FALSE)),
           row.names = c("AIC", "chisq"))
```

```
##            model.B      model.S
## AIC   9.947075e+01 1.052948e+02
## chisq 1.462963e-89 1.350598e-87
```

Problem 1.e (2 points)

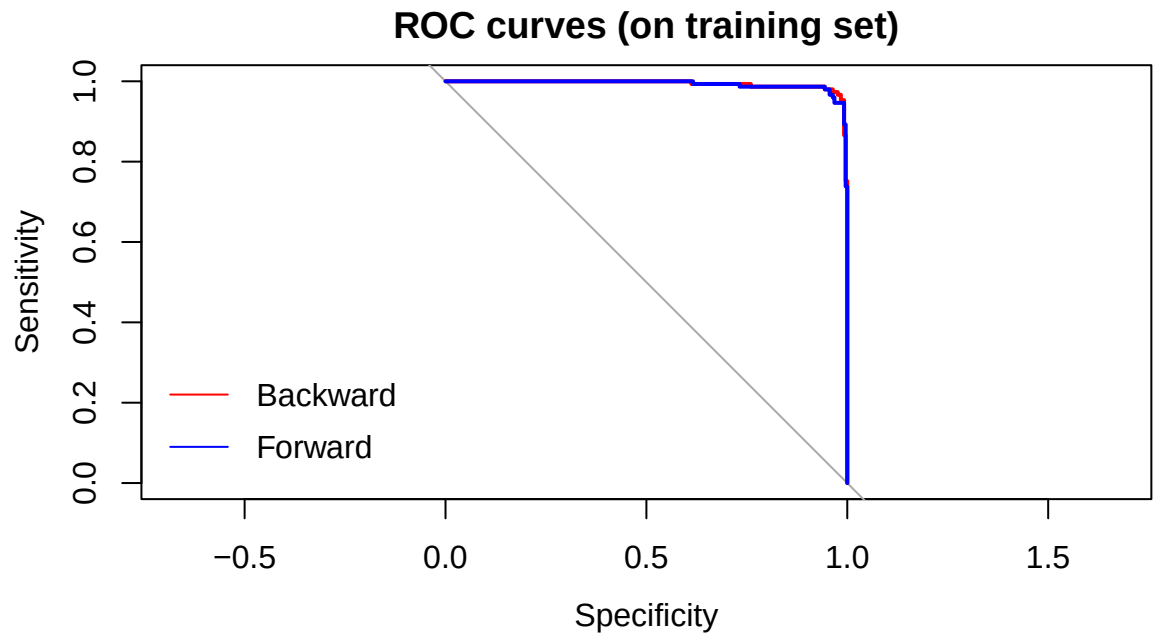
Compute the training AUC for model B and model S.

Solution:

To compute the training AUC for two stepwise models, we first obtain the predictive values of training set using model.B and model.S. With predictive values, we use `roc()` function from `pROC` package to calculate AUC of training set for model.B and model.S.

From the result, we can find that the AUC scores for both models are quite high as well as well performed ROC curves. Backward stepwise selection performs slightly better than forward stepwise method, which is consistent with the conclusions from previous question.

```
# predict values on training set
pred.B <- predict(model.B, newdata = train.wdbc, type = "response")
pred.S <- predict(model.S, newdata = train.wdbc, type = "response")

# AUC and ROC curves
invisible({capture.output({
auc.B <- roc(train.y.mat ~ pred.B, plot = TRUE, col = "red", xlim = c(0, 1),
             main = "ROC curves (on training set)")$auc
auc.S <- roc(train.y.mat ~ pred.S, plot = TRUE, add = TRUE, col = "blue")$auc
legend("bottomleft", legend = c("Backward", "Forward"),
       col = c("red", "blue"), lwd = c(1, 1), bty = "n")
})})
```




```
data.frame(model = c("model.B", "model.S"), AUC = c(auc.B, auc.S))
```

```
##     model       AUC
## 1 model.B 0.9936376
## 2 model.S 0.9929396
```

Problem 1.f (6 points)

Use the four models to predict the outcome for the observations in the test set (use the lambda at 1 standard error for the penalised models). Plot the ROC curves of these models (on the sameplot, using different colours) and report their test AUCs. Compare the training AUCs obtained in problems 1.b and 1.e with the test AUCs and discuss the fit of the different models.

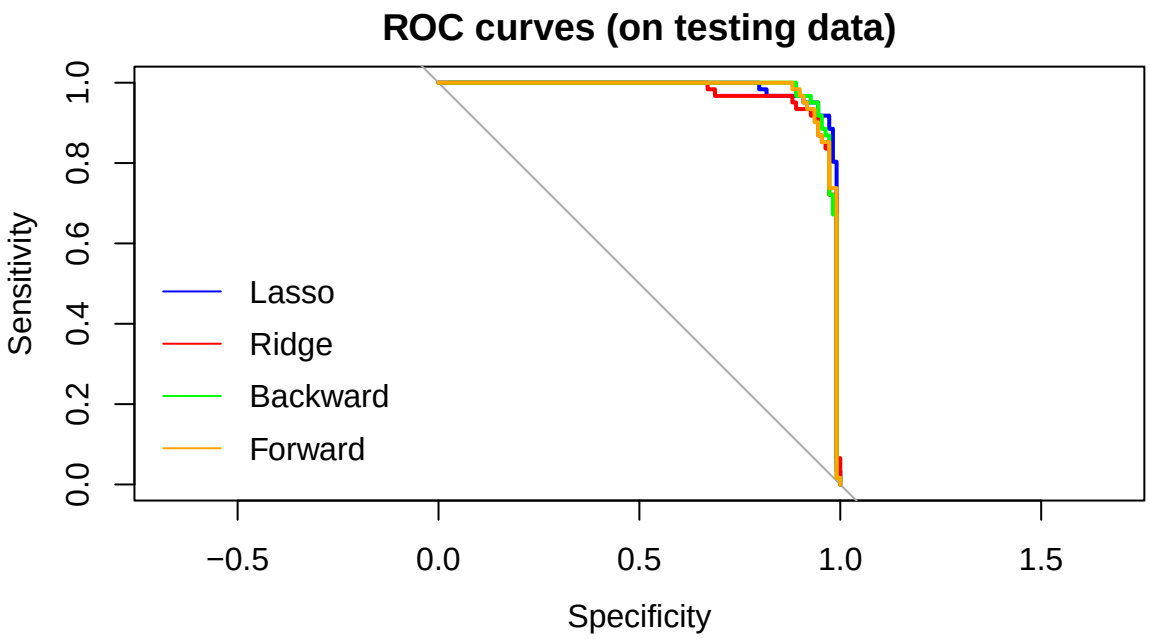
Solution:

Similar with 1.e, to get the testing AUC of each model, we need to calculate the predictive values of testing set for each model. After that, again, we use `roc()` function to plot the testing ROC curves of each model in the same graph. Moreover, we report the testing AUC for each model.

Looking at the ROC curves on testing set, we consider good performances of 4 models with AUCs over 0.95. Specifically, Lasso regression has higher testing AUC than its training AUC, while the other 3 models has higher training AUCs than their testing AUCs. Moreover, basically stepwise models performs better on training set, while penalised models have better performances on testing model. Note that we may not consider the problem of over-fitting for penalised models as the idea of regularization reduces over-fitting. However, we possibly consider over-fitting happens to stepwise models.

In conclusion, we consider Lasso regression as the best model among 4 mentioned models.

```
# predict values of testing set
pred.lasso.test <- predict(fit.cv.lasso, newx = test.x.mat,
                      s = fit.cv.lasso$lambda.1se, type = "response")
pred.ridge.test <- predict(fit.cv.ridge, newx = test.x.mat,
                      s = fit.cv.ridge$lambda.1se, type = "response")
pred.B.test <- predict(model.B, newdata = test.x, type = "response")
pred.S.test <- predict(model.S, newdata = test.x, type = "response")

# plot ROC curves of four models
invisible({capture.output({
roc.lasso <- roc(test.y.mat, pred.lasso.test,
                 plot = TRUE, col = "blue", xlim = c(0, 1),
                 main = "ROC curves (on testing data)")
roc.ridge <- roc(test.y.mat, pred.ridge.test,
                 plot = TRUE, col = "red", add = TRUE, xlim = c(0, 1))
roc.B <- roc(test.y.mat, pred.B.test,
                 plot = TRUE, col = "green", add = TRUE, xlim = c(0, 1))
roc.S <- roc(test.y.mat, pred.S.test,
                 plot = TRUE, col = "orange",add = TRUE,  xlim = c(0, 1))
})})

# add legend
legend("bottomleft", legend = c("Lasso", "Ridge", "Backward", "Forward"),
       col = c("blue", "red", "green", "orange"),
       lty = c(1, 1, 1, 1), bty = "n")
```




```
# data frame to report the AUC of each model on both training and testing sets
data.frame(train.AUC = c(fit.cv.lasso$cvm[fit.cv.lasso$index[2]],
                         fit.cv.ridge$cvm[fit.cv.ridge$index[2]],
                         auc.B, auc.S),
           test.AUC = c(roc.lasso$auc, roc.ridge$auc,
                        roc.B$auc, roc.S$auc),
           row.names = c("Lasso", "Ridge", "Backward", "Forward"))
```

```
##          train.AUC  test.AUC
## Lasso    0.9759440 0.9808994
## Ridge    0.9770664 0.9712739
## Backward 0.9936376 0.9802978
## Forward  0.9929396 0.9790946
```

Problem 2 (40 points)

File GDM.raw.txt (available from the accompanying zip folder on Learn) contains 176 SNPs to be studied for association with incidence of gestational diabetes (a form of diabetes that is specific to pregnant women). SNP names are given in the form "rs1234_X" where "rs1234" is the official identifier (rsID), and "X" (one of A, C, G, T) is the reference allele.

Problem 2.a (3 points)

Read file GDM.raw.txt into a data table named gdm.dt. Impute missing values in gdm.dt according to SNP-wise median allele count.

Solution:

By looking inside the dataset, we observe that only SNP columns contains missing values. We define a function `impute.to.median()` to impute the missing values of each SNP column with its median. After imputation, we count the number of missing terms and the result of 0 indicates there is no missing values in the imputed dataset.

```
# read data into R
gdm.dt <- fread("assignment2/GDM.raw.txt")
```

```
# define a function to impute missing values in each column with median
impute.to.median <- function(x){
  na.idx <- is.na(x)
  x[na.idx] <- median(x, na.rm = TRUE)
  return(x)
}

# impute the missing values using the function
gdm.dt.imputed <- gdm.dt[, (colnames(gdm.dt)) := lapply(.SD, impute.to.median),
                         .SDcols = colnames(gdm.dt)]

# check if there is missing values after imputation
sum(is.na(gdm.dt.imputed))
```

```
## [1] 0
```

Problem 2.b (8 points)

Write function univ.glm.test <- function(x, y, order = FALSE) where x is a data table of SNPs, y is a binary outcome vector, and order is a boolean. The function should fit a logistic regression model for each SNP in x, and return a data table containing SNP names, regression coefficients, odds ratios, standard errors and p-values. If order is set to TRUE, the output data table should be ordered by increasing p-value.

Solution:

```
univ.glm.test <- function(x, y, order = FALSE){
  # initialize lists for output
  name <- NULL
  coef <- NULL
  OR <- NULL
  se <- NULL
  p.value <- NULL

  # get the names of columns in x
  x.cols <- colnames(x)

  # logistic regression for each SNP in x
  for (i in 1:length(x.cols)){
    logreg <- glm(y ~ x[[i]], family = "binomial")
    reg.sum <- summary(logreg)
    name[i] <- x.cols[i]
    coef[i] <- reg.sum$coefficients[2, 1]
    OR[i] <- exp(coef[i])
    se[i] <- reg.sum$coefficients[2, 2]
    p.value[i] <- reg.sum$coefficients[2, 4]
  }

  # create a data table including the values
  dt <- data.table(name, coef, OR, se, p.value)

  # if order = TRUE ...
  if (order == TRUE){
    return(setorder(dt, p.value))
  }
  else{
    return(dt)
  }
}
```

Problem 2.c (5 points)

Using function univ.glm.test(), run an association study for all the SNPs in gdm.dt against having gestational diabetes (column "pheno"). For the SNP that is most strongly associated to increased risk of gestational diabetes and the one with most significant protective effect, report the summary statistics from the GWAS as well as the 95% and 99% confidence intervals on the odds ratio.

Solution:

With pre-defined function `univ.glm.test()`, we run logistic regression over each SNP in gdm.dt (imputed) separately and obtained the result data table, rep. To ensure the consistence of the order of SNPs between rep and original dataset, we set the `order` option to "FALSE" as default.

As required, we find the SNP with the smallest regression p-value, i.e. $rs12243326\_A$, which is considered to be mostly strongly associated to increased risk of gestational diabetes. Besides statistics in `rep`, we also report its 95% and 99% confidence intervals. Since the confidence intervals do not include 1, we have sufficient evidence to conclude that $rs12243326\_A$ is strongly associated with the outcome of interest, consistent with our previous conclusion.

In addition, we report statistics including 95% and 99% confidence intervals for the SNP with most significant protective effect on gestational diabetes, i.e. $rs11575839\_C$, which has the most negative regression coefficient. Note that the confidence intervals include 1, which suggests insignificance of $rs11575839\_C$, consistent with its large p-value of 0.1090394.

```
# apply pre-defined function to the imputed dataset
rep <- univ.glm.test(x = gdm.dt.imputed[, -c(1, 2, 3)],
                     y = gdm.dt.imputed$pheno)
head(rep, 5)
```

```
##           name        coef       OR        se   p.value
## 1: rs7513574_T 0.002157494 1.002160 0.1051372 0.9836280
## 2: rs1627238_A 0.114637864 1.121467 0.1138224 0.3138559
## 3: rs1171278_C 0.121409445 1.129087 0.1138073 0.2860628
## 4: rs1137100_A 0.060104751 1.061948 0.1104238 0.5862285
## 5: rs2568958_A 0.149379896 1.161114 0.1233800 0.2259989
```

```
#The SNP mostly strongly associated to increased risk of gestational diabetes is:
SNP.msa <- rep[which.min(rep$p.value)]
print(SNP.msa)
```

```
##            name      coef       OR        se      p.value
## 1: rs12243326_A 0.6454198 1.906787 0.1583787 4.598104e-05
```

```
# The 95% and 99% confidence intervals of corresponding odds ratio are:
CI.95.msa <- exp(SNP.msa$coef + qnorm(0.975) * SNP.msa$se * c(-1, 1))
CI.99.msa <- exp(SNP.msa$coef + qnorm(0.995) * SNP.msa$se * c(-1, 1))
data.frame(lower = c(round(CI.95.msa[1], 3), round(CI.99.msa[1], 3)),
           upper = c(round(CI.95.msa[2], 3), round(CI.99.msa[2], 3)),
           row.names = c("95%", "99%"))
```

```
##     lower upper
## 95% 1.398 2.601
## 99% 1.268 2.867
```

```r
#The SNP with most significant protective effect is:
SNP.msp <- rep[which.min(rep$coef)]
print(SNP.msp)
```

```
##            name       coef        OR        se   p.value
## 1: rs11575839_C -0.6022542 0.5475759 0.3758156 0.1090394
```

```r
# The 95% and 99% confidence intervals of corresponding odds ratio are:
CI.95.msp <- exp(SNP.msp$coef + qnorm(0.975) * SNP.msp$se * c(-1, 1))
CI.99.msp <- exp(SNP.msp$coef + qnorm(0.995) * SNP.msp$se * c(-1, 1))
data.frame(lower = c(round(CI.95.msp[1], 3), round(CI.99.msp[1], 3)),
           upper = c(round(CI.95.msp[2], 3), round(CI.99.msp[2], 3)),
           row.names = c("95%", "99%"))
```

```
##     lower upper
## 95% 0.262 1.144
## 99% 0.208 1.442
```

Problem 2.d (4points)

Merge your GWAS results with the table of gene names provided in file GDM.annot.txt (available from the accompanying zip folder on Learn). For SNPs that have p-value $< 10^{-4}$ (hit SNPs) report SNP name, effect allele, chromosome number and corresponding gene name. Separately, report for each 'hit SNP' take names of the genes that are within a 1Mb window from the SNP position on the chromosome. Note: That's genes that fall within +/- 1,000,000 positions using the 'pos' column in the dataset.

Solution:

Since SNP names in rep, i.e. `name` terms, consist of the official identifiers of SNPs and the reference alleles, we need to split `name` into `snp` and `allele`. Then we merge two dataset by mutual column `snp`.

After merging, we report the 'hit SNPs' (p-value $< 1e-4$) $rs12243326$ and $rs2237897$, with effect allele of A and T, chromosome number of 10 and 11 and gene name of $TCF7L2$ and $KCNQ1$ respectively. We also make an auxiliary Manhattan plot to confirm the 'hit.SNPs' we found.

Moreover, for each 'hit SNP', we report the gene names that are within a 1Mb window from the SNP position on the chromosome:

- `rs12243326`: $TCF7L2$
- `rs2237897`: TH, $KCNQ1$, $ACNA2D4$, $SMG6$

People with these genes are more likely to be diagnosed with malignant tumour.

```
# read dataset into R
gdm.annot.dt <- data.table(read.table("assignment2/GDM.annot.txt",
                                      sep = "\t", header = TRUE))

# merge previous results with gene names in gdm.annot.dt
# before merging, split 'name' in res to SNP names and effect allele
rep$snp <- sapply(rep$name, function(x) strsplit(x, split = "_")[[1]][1])
rep$allele <- sapply(rep$name, function(x) strsplit(x, split = "_")[[1]][2])

# now merge the datasets by 'snp'
rep.merged <- merge(rep, gdm.annot.dt, by = "snp")
```

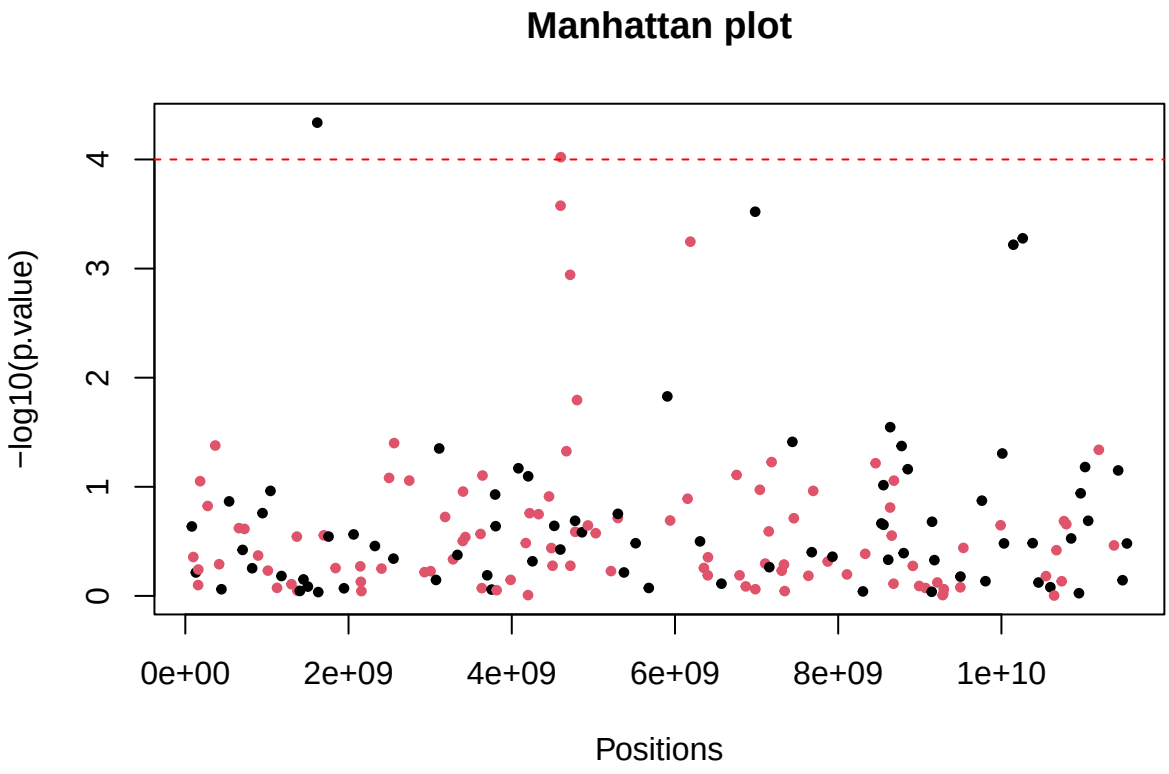
```
# find the hit SNPs and report
idx.hit.SNP <- which(rep.merged$p.value < 1e-4)
rep.hit.SNP = rep.merged[idx.hit.SNP, ]
rep.hit.SNP[, .(snp, allele, chrom, gene)]
```

```
##           snp allele chrom   gene
## 1: rs12243326      A    10 TCF7L2
## 2:  rs2237897      T    11  KCNQ1
```

```
# manhattan plot
rep.merged$pos <- as.numeric(rep.merged$pos)
rep.merged$cum.pos <- cumsum(rep.merged$pos)
rep.merged$chrom <- as.numeric(rep.merged$chrom)
```

```
## Warning: NAs introduced by coercion
```

```
chrom.cols <- 1 + rep.merged$chrom %% 2
with(rep.merged, plot(cum.pos, -log10(p.value),
                      col = chrom.cols, pch = 19, cex = 0.6,
                      main = "Manhattan plot", xlab = "Positions"))
abline(h = -log10(1e-4), lty = 2, col = "red")
```




```
# 1Mb window for 1st SNP in hit.SNP
unique(rep.merged[abs(rep.merged$pos - rep.hit.SNP$pos[1]) <= 10^6
                  & rep.merged$pos != rep.hit.SNP$pos[1], "gene"])
```

```
##      gene
## 1: TCF7L2
```

```
# 1Mb window for 2nd SNP in hit.SNP
unique(rep.merged[abs(rep.merged$pos - rep.hit.SNP$pos[2]) <= 10^6
                      & rep.merged$pos != rep.hit.SNP$pos[2], "gene"])
```

```
##        gene
## 1:       TH
## 2:    KCNQ1
## 3: CACNA2D4
## 4:     SMG6
```

Problem 2.e (8 points)

Build a weighted genetic risk score that includes all SNPs with p-value $< 10^{-4}$, a score with all SNPs with p-value $< 10^{-3}$, and a score that only includes SNPs on the FTO gene (hint: ensure that the ordering of SNPs is respected). Add the three scores as columns to the gdm.dt data table. Fit the three scores in separate logistic regression models to test their association with gestational diabetes, and for each report odds ratio, 95% confidence interval and p-value.

Solution:

Generally, we construct a weighted genetic risk score for SNPs by their regression coefficients, so that both direction and contribution of the effect to final score for a SNP are considered.

Once we obtain the scores, we add them as new columns into dataset and apply logistic regression to them in separate. We report the odds ratio, 95% confidence interval and p-value of each score.

Looking the results, from the p-values, we consider the risk scores for both SNPs with p-values $< 1e-4$ and SNPs with p-values $< 1e-3$ have significant associations with gestational diabetes as p-values for two scores are relatively low with confidence intervals excluding 1. However, for SNPs on the FTO genes, the corresponding score has a p-value > 0.05 and its confidence interval covers 1, implying an insignificant contribution to the gestational diabetes.

```
# subset of SNPs with p-value < 1e-4
gdm.hit.SNP <- gdm.dt.imputed[, .SD, .SDcols = rep.hit.SNP$name]

# subset of SNPs with p-value < 1e-3
idx.p3.SNP <- which(rep.merged$p.value < 1e-3)
rep.p3.SNP <- rep.merged[idx.p3.SNP, ]
gdm.p3.SNP <- gdm.dt.imputed[, .SD, .SDcols = rep.p3.SNP$name]

# subset of SNPs on the FTO gene
idx.FTO.SNP <- which(rep.merged$gene == 'FTO')
rep.FTO.SNP <- rep.merged[idx.FTO.SNP, ]
gdm.FTO.SNP <- gdm.dt.imputed[, .SD, .SDcols = rep.FTO.SNP$name]
```

```
# weighted genetic risk score for each group of SNPs
score.hit.SNP <- as.matrix(gdm.hit.SNP) %*% rep.hit.SNP$coef
score.p3.SNP <- as.matrix(gdm.p3.SNP) %*% rep.p3.SNP$coef
score.FTO.SNP <- as.matrix(gdm.FTO.SNP) %*% rep.FTO.SNP$coef

# add weighted scores to the original data table
gdm.dt.imputed$score.hit <- score.hit.SNP
gdm.dt.imputed$score.p3 <- score.p3.SNP
gdm.dt.imputed$score.FTO <- score.FTO.SNP
```

```
# univariate logistic regressions
logreg.hit <- glm(pheno ~ score.hit, data = gdm.dt.imputed, family = "binomial")
logreg.p3 <- glm(pheno ~ score.p3, data = gdm.dt.imputed, family = "binomial")
logreg.FTO <- glm(pheno ~ score.FTO, data = gdm.dt.imputed, family = "binomial")

# use a data frame to report
invisible({capture.output({
data.frame(name = c("score.hit", "socre.p3", "score.FTO"),
           OR = exp(c(summary(logreg.hit)$coefficients[2, 1],
```

```
                      summary(logreg.p3)$coefficients[2, 1],
                      summary(logreg.FTO)$coefficients[2, 1])),
          CI.lower = exp(c(confint(logreg.hit)[2, 1],
                           confint(logreg.p3)[2, 1],
                           confint(logreg.FTO)[2, 1])),
          CI.upper = exp(c(confint(logreg.hit)[2, 2],
                           confint(logreg.p3)[2, 2],
                           confint(logreg.FTO)[2, 2])),
          p.value = c(summary(logreg.hit)$coefficients[2, 4],
                      summary(logreg.p3)$coefficients[2, 4],
                       summary(logreg.FTO)$coefficients[2, 4]))
})})
```

Problem 2.f (4 points)

File GDM.test.txt (available from the accompanying zip folder on Learn) contains genotypes of another 40 pregnant women with and without gestational diabetes (assume that the reference allele is the same one that was specified in file GDM.raw.txt). Read the file into variable gdm.test. For the set of patients in gdm.test, compute the three genetic risk scores as defined in problem 2.e using the same set of SNPs and corresponding weights. Add the three scores as columns to gdm.test (hint: use the same columnnames as before).

Solution:

As required, we use the same set of SNPs and corresponding weights to construct score columns in gdm.test.

```
# read data into R
gdm.test <- fread("assignment2/GDM.test.txt")

# subset of each group
gdm.test.hit.SNP <- gdm.test[, .SD, .SDcols = rep.hit.SNP$snp]
gdm.test.p3.SNP <- gdm.test[, .SD, .SDcols = rep.p3.SNP$snp]
gdm.test.FTO.SNP <- gdm.test[, .SD, .SDcols = rep.FTO.SNP$snp]

# compute the score of each group
score.test.hit.SNP <- as.matrix(gdm.test.hit.SNP) %*% rep.hit.SNP$coef
score.test.p3.SNP <- as.matrix(gdm.test.p3.SNP) %*% rep.p3.SNP$coef
score.test.FTO.SNP <- as.matrix(gdm.test.FTO.SNP) %*% rep.FTO.SNP$coef

# add scores to gdm.test
gdm.test$score.hit <- score.test.hit.SNP
gdm.test$score.p3 <- score.test.p3.SNP
gdm.test$score.FTO <- score.test.FTO.SNP

head(gdm.test[, c(180:182)], 5)
```

```
##     score.hit   score.p3 score.FTO
## 1: -0.2334714 -1.0420490 0.0000000
## 2: -0.8788912 -1.6874688 0.4740752
## 3:  0.0000000  0.0000000 0.0000000
## 4: -0.4394456 -0.4394456 0.4740752
## 5:  0.0000000  0.0000000 0.0000000
```

Problem 2.g (4 points)

Use the logistic regression models fitted in problem 2.e to predict the outcome of patients in gdm.test. Compute the test log-likelihood for the predicted probabilities from the three genetic risk score models.

Solution:

We use pre-fitted logistic regression models of each score to predict the outcomes of patients in gdm.test. Based on the predicted probabilities from each genetic risk score model, we calculate the log density of each observation in gdm.test, which follows a Bernoulli distribution, and obtain the test log-likelihood.

```
# use pre-fitted logistic regressions to predict
predict.test.hit.SNP <- predict(logreg.hit, newdata = gdm.test,
                                type = "response")
predict.test.p3.SNP <- predict(logreg.p3, newdata = gdm.test,
                               type = "response")
predict.test.FTO.SNP <- predict(logreg.FTO, newdata = gdm.test,
                                type = "response")

# test log-likelihood for predicted probabilities
log.lld.test.hit <- sum(dbinom(gdm.test$pheno, size = 1,
                               prob = predict.test.hit.SNP, log = TRUE))
log.lld.test.p3 <- sum(dbinom(gdm.test$pheno, size = 1,
                              prob = predict.test.p3.SNP, log = TRUE))
log.lld.test.FTO <- sum(dbinom(gdm.test$pheno, size = 1,
                               prob = predict.test.FTO.SNP, log = TRUE))

# report the test log-likelihood
data.frame(score.model = c("score.hit", "score.p3", "score.FTO"),
           test.log.likelihood = c(log.lld.test.hit, log.lld.test.p3,
                                   log.lld.test.FTO))
```

```
##   score.model test.log.likelihood
## 1   score.hit           -25.06824
## 2    score.p3           -24.77693
## 3   score.FTO           -28.05355
```

Problem 2.h (4points)

File GDM.study2.txt (available from the accompanying zip folder on Learn) contains the summary statistics from a different study on the same set of SNPs. Perform a meta-analysis with the results obtained in problem 2.c (hint: remember that the effect alleles should correspond) and produce a summary of the meta-analysis results for the set of SNPs with meta-analysis p-value $< 10^{-4}$ sorted by increasing p-value.

Solutions:

When genome-wide association studies of the same phenotype are performed in multiple independent studies, it is common to merge the results by performing a meta-analysis. This has the benefit of increasing the statistical power and reducing false-positive findings (type I error rate).

To preform the meta-analysis, we need to harmonize two datasets. Thus, we take out a subset of rep which only contains columns in gdm.study2, i.e. `snp`, `allele`, `coef` and `se`. Moreover, we rename some columns with the corresponding column names in gdm.study2. After that, we also ensure SNPs in two datasets correspond and order them by SNP and effect allele as SNP and effect allele are what matter in the meta-analysis.

We start with finding which SNPs have both alleles matching, and which would match if the alleles were flipped. The following result indicates that in this case, there are two SNPs whose alleles do not match even after swapping them. This could be caused by a gene-typing or imputation error, or because of some other discrepancy in the GWAS pipelines adopted by the different studies. As a result, these two SNPs cannot be meta-analysed, and we drop them from both datasets.

The effect sizes of the SNPs which we identified to have alleles flipped need to have their direction of effect swapped in one of the studies before entering the meta-analysis. Here, without loss of generality, we swap the sign for gdm.study2.

Now, with data pre-processing ready, we perform a fixed effect meta-analysis by using inverse variance weighting. According to the weights, we may consider that in general results from 2.c are more powered.

Moreover, we derive the meta-analysis effect size which is a weighted sum of effect sizes from two studies, based on weights obtained before. Finally, we compute the p-values for the meta-analysis and construct a meta-analysis summary for SNP terms with meta-analysis p-value $< 1e-4$, increasingly ordered by p-values.

By combining two studies together, we increase the statistical power of the study. We observe this improvement by plotting the meta-analysis p-values over the p-values from `rep` dataset, where most data points are left skewed.

```
# read data into R
gdm.study2 <- fread("assignment2/GDM.study2.txt")

# make the variable names in rep consistent with gdm.study2
rep.meta <- rep[, .(snp, allele, coef, se)]
setnames(rep.meta, c("allele", "coef"), c("effect_allele", "beta"))

# ensure the SNPs in two datasets correspond
rep.meta <- rep.meta[snp %in% gdm.study2$snp]
gdm.study2 <- gdm.study2[snp %in% rep.meta$snp]

# order two datasets by SNP and effect allele
rep.meta <- rep.meta[order(snp, effect_allele)]
gdm.study2 <- gdm.study2[order(snp, effect.allele)]
stopifnot(all.equal(rep.meta$snp, gdm.study2$snp))
```

```r
# SNPs with alleles matching or flipped alleles matching
both.ok <- rep.meta$effect_allele == gdm.study2$effect.allele
flipped <- rep.meta$effect_allele == gdm.study2$other.allele
table(both.ok, flipped)
```

```
##        flipped
## both.ok FALSE TRUE
##   FALSE     2   27
##   TRUE    147    0
```

```r
# drop SNPs with non-matching alleles
idx.nomatching <- which(both.ok == FALSE & flipped == FALSE)
rep.meta <- rep.meta[-idx.nomatching, ]
gdm.study2 <- gdm.study2[-idx.nomatching, ]

# update both.ok and flipped
both.ok <- both.ok[-idx.nomatching]
flipped <- flipped[-idx.nomatching]
table(both.ok, flipped)
```

```
##        flipped
## both.ok FALSE TRUE
##   FALSE     0   27
##   TRUE    147    0
```

```r
# swap signs of betas of SNPs with alleles matching after flipped in gdm.study2
beta1 <- rep.meta$beta
beta2 <- gdm.study2$beta
beta2[flipped] <- -beta2[flipped]
```

```r
# perform a fixed effect meta-analysis
weight.rep.meta <- 1 / rep.meta$se^2
weight.gdm.study2 <- 1 / gdm.study2$se^2
head(weight.rep.meta)
```

```
## [1] 95.95663 48.35218 51.12222 80.72417 32.39329 80.53452
```

```r
head(weight.gdm.study2)
```

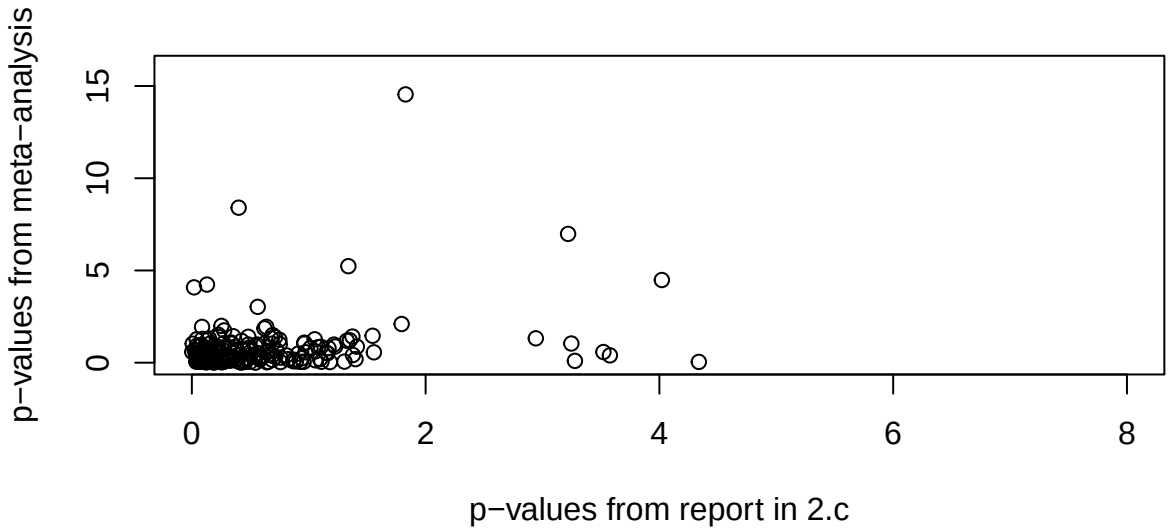
```
## [1]  9.141470  6.919489  7.069651 10.430712  3.328963  9.751846
```

```r
# computation of meta-analysis effect size
beta.meta <- (weight.rep.meta*beta1+weight.gdm.study2*beta2)/
  (weight.rep.meta+weight.gdm.study2)
se.meta <- sqrt(1/(weight.rep.meta+weight.gdm.study2))

# p-values of meta-analysis
p.value.meta <- 2 * pnorm(abs(beta.meta/se.meta), lower.tail = FALSE)
```

```
# SNPs with meta-analysis p-values < 1e-4
idx.p4.meta <- which(p.value.meta < 1e-4)

# report the meta-analysis result for SNPs with meta-analysis p-values < 1e-4
summary.p4.meta <- data.table(snp = rep.meta[idx.p4.meta, ]$snp,
                              # weight.rep.meta = weight.rep.meta[idx.p4.meta],
                              # weight.gdm.study2 = weight.gdm.study2[idx.p4.meta],
                              beta.meta = beta.meta[idx.p4.meta],
                              se.meta = se.meta[idx.p4.meta],
                              p.value.meta = p.value.meta[idx.p4.meta])

summary.p4.meta <- summary.p4.meta[order(p.value.meta)]
summary.p4.meta
```

```
##           snp  beta.meta   se.meta p.value.meta
## 1: rs12243326  0.8918988 0.1129379 2.851239e-15
## 2:  rs2237897 -0.5903702 0.1002930 3.945724e-09
## 3:  rs3786897  0.6069063 0.1141012 1.043301e-07
## 4:  rs2237892 -0.4834871 0.1066965 5.858759e-06
## 5:  rs4506565  0.5396749 0.1299622 3.287877e-05
## 6:  rs7903146  0.5353424 0.1331728 5.822054e-05
## 7:  rs7901695  0.5409567 0.1374089 8.256236e-05
```

```
# additional check on meta-analysis if it improves the statistical power
plot(-log10(rep[-idx.nomatching]$p.value), -log10(p.value.meta),
     xlim = c(0, 8), ylim = c(0, 16),
     xlab = "p-values from report in 2.c",
     ylab = "p-values from meta-analysis"
     )
```



Problem 3 (33 points)

File nki.csv (available from the accompanying zip folder on Learn) contains data for 144 breast cancer patients. The dataset contains a binary outcome variable ("Event", indicating the insurgence of further complications after operation), covariates describing the tumour and the age of the patient, and gene expressions for 70 genes found to be prognostic of survival.

Problem 3.a (6 points)

Compute the matrix of correlations between gene expression variables, and display it so that a block structure is highlighted. Discuss what you observe. Write some code to identify the unique pairs of (distinct) variables that have correlation coefficient greater than 0.80 in absolute value and report correlation coefficients.

Solution:

To compute the correlations between the gene expression variables, we first obtained a subset which only contains the gene expression variables and then computed the correlation matrix using `cor()` function. Moreover, we visualized the strength of correlation between the gene variables by creating a correlation plot.

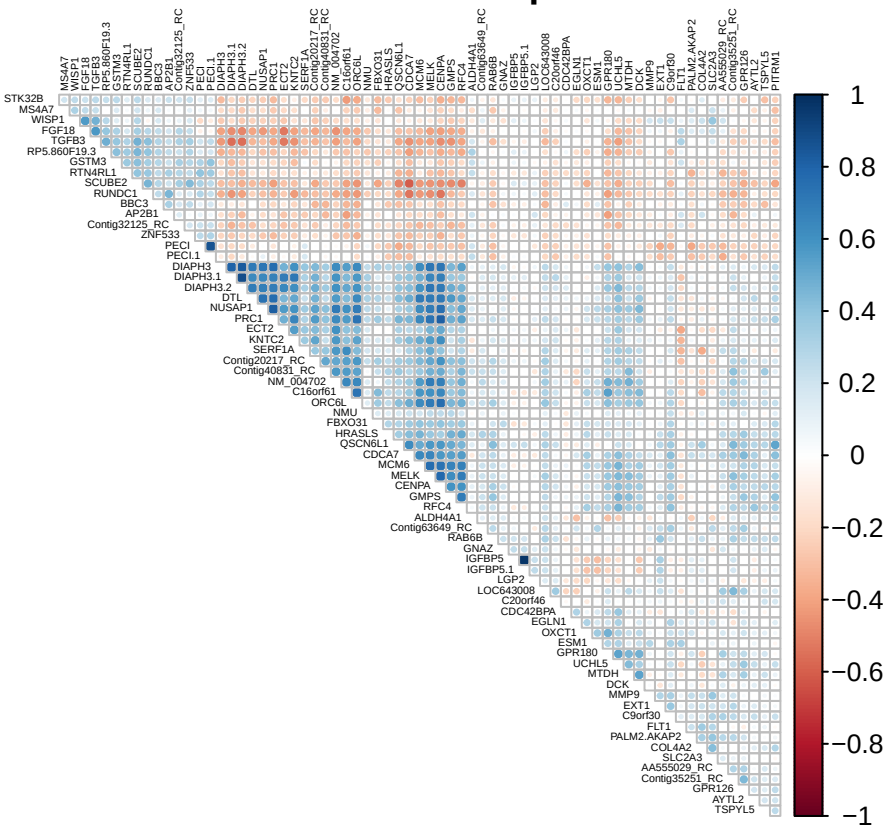
Since the number of gene expression variables is large (70 columns), we improved the default plot by using options including `order` and `type`.

Looking at the correlation plot above, we can see that there are strong correlations between variables with similar names, such as "IGFBP5" and "IGFBP5.1". It is plausible to think that variables with similar names contains similar information and have similar effect on the outcome variable. Generally, high correlation between variables indicates high strength of linear association. On the other hand, we may also consider the existence of multicollinearity among variables with relatively high correlations.

In particular, we found the unique pairs of distinct variables with correlation coefficients greater than 0.80 in absolute value and reported them in the following data frame. As we expected, high correlations appeared in variables with similar names as well as two other pairs corresponding to "PRC1".

```
# read data into R
nki <- read.csv("assignment2/nki.csv", header = TRUE)
```

```
# subset of gene expression variables
gene.expression <- nki[, -c(1 : 6)]

# correlations matrix
gene.expression.corr <- cor(gene.expression, use = "pairwise.complete")

# visualize the correlations using a heatmap
corrplot(gene.expression.corr,
         order = "hclust",      # order the variables by correlation clusters
         title = "Correlation Matrix of Gene Expression Variables",
         tl.col = "black", tl.cex = 0.3,  # set the colour and size of labels
         diag = FALSE,   # remove diagonal components as they are 1 trivially
         type = "upper",                    # display the upper triangle only
         mar = c(0, 0, 1, 0))                       # set the size of margins
```




```
# find the unique pairs of distinct variables with correlation > 0.8
# initialize an empty data frame
corr.08 <- data.frame()

# loop to get pairs with correlation > 0.8 except identity pairs
for (i in 2:ncol(gene.expression.corr)){
  for (j in 1:(i-1)){
    pair <- paste(rownames(gene.expression.corr)[i],
                  colnames(gene.expression.corr)[j], sep = ", ")
    corr <- gene.expression.corr[i, j]
    if (abs(corr) > 0.8){
      df <- data.frame(pair = pair, correlation = corr)
    corr.08 <- rbind(corr.08, df)
    }
  }
}
corr.08
```

```
##                  pair correlation
## 1   DIAPH3.1, DIAPH3   0.8031368
## 2   DIAPH3.2, DIAPH3   0.8338591
## 3 DIAPH3.2, DIAPH3.1   0.8868741
## 4       PECI.1, PECI   0.8697836
## 5   IGFBP5.1, IGFBP5   0.9775030
## 6       PRC1, NUSAP1   0.8298356
## 7        CENPA, PRC1   0.8175424
```

Problem 3.b (8 points)

Run PCA (only over the columns containing gene expressions), in order to derive a patient-wise summary of all gene expressions (dimensionality reduction). Decide which components to keep and justify your decision. Test if those principal components are associated with the outcome in unadjusted logistic regression models and in models adjusted for age, estrogen receptor and grade. Justify the difference in results between unadjusted and adjusted models.

Solution:

Using `prcomp()` function, we applied PCA on the subset of gene expression variables and derived a patient-wise summary of all gene expression.

The amount of variability explained by the components can be computed bearing in mind that the square root of the eigenvalues is stored in vector `sdev` of the PCA object. The variance explained by the principal components can be visualized through a scree plot. From the scree plot, we observe that, from the PC4, the variance explained by each principle component keeps at a relatively low level.

Furthermore, we construct a plot to show the cumulative proportion of the variance explained by principle components. In this plot, we also highlight the positions where the cumulative proportion is over 50% and 90%, i.e. $n = 6$ and $n = 33$. In general, to have better model accuracy, we usually extract the first n principle components so that their cumulative proportion of variance is at least 80%. However, in this case, the first 22 principle components are required, which is a large number resulting in a complex model.

Thus, for both model accuracy and conciseness, we consider using the first 3 principle components in further steps, though they only explain 37.7% of variance.

We test the association between the outcome and each principle component by fitting an unadjusted univariate logistic regression over the PC as well as a logistic regression adjusted by `Age`, `EstrogenReceptor` and `Grade`. According to the p-values from regression results, we find:

- PC2 is insignificant for both unadjusted and adjusted models
- The first principle component is significant in unadjusted model, while it becomes insignificant after adjusting on the aforementioned variables.
- Principle component 3 keeps its significance in both models.

Besides, we fit a logistic regression over 3 principle components by adjusting them for `Age`, `EstrogenReceptor` and `Grade`. We obtain a poor model performance where the third principle component is the only significant variable with $\alpha = 0.05$.

All in all, we do not think applying PCA is a good choice for our data as the percentage of variance explained by each principle component is not large. Moreover, the regression results which the first two are not significant adjusting for `Age`, `EstrogenReceptor` and `Grade` also suggest poor model performance of PCA. We will consider other models including penalized models and stepwise selections to fit the data in further steps.

```
# run PCA
pca.patients <- prcomp(gene.expression, center = TRUE, scale = TRUE)
summary(pca.patients)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     4.1171 2.30541 2.02437 1.78597 1.73982 1.68091 1.42309
```

```
## Proportion of Variance 0.2422 0.07593 0.05854 0.04557 0.04324 0.04036 0.02893
## Cumulative Proportion  0.2422 0.31808 0.37662 0.42219 0.46543 0.50580 0.53473
##                            PC8     PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     1.36441 1.29119 1.2715 1.24741 1.18388 1.15101 1.13883
## Proportion of Variance 0.02659 0.02382 0.0231 0.02223 0.02002 0.01893 0.01853
## Cumulative Proportion  0.56132 0.58514 0.6082 0.63046 0.65049 0.66941 0.68794
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     1.09473 1.07016 1.04187 1.00234 0.99086 0.94095 0.93322
## Proportion of Variance 0.01712 0.01636 0.01551 0.01435 0.01403 0.01265 0.01244
## Cumulative Proportion  0.70506 0.72142 0.73693 0.75128 0.76531 0.77796 0.79040
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.90727 0.89675 0.88859 0.86019 0.84462 0.82782 0.82368
## Proportion of Variance 0.01176 0.01149 0.01128 0.01057 0.01019 0.00979 0.00969
## Cumulative Proportion  0.80216 0.81364 0.82492 0.83549 0.84569 0.85548 0.86517
##                           PC29    PC30    PC31    PC32    PC33    PC34   PC35
## Standard deviation     0.78694 0.75594 0.73942 0.70569 0.69414 0.67129 0.6639
## Proportion of Variance 0.00885 0.00816 0.00781 0.00711 0.00688 0.00644 0.0063
## Cumulative Proportion  0.87401 0.88218 0.88999 0.89710 0.90399 0.91042 0.9167
##                           PC36    PC37    PC38    PC39    PC40    PC41    PC42
## Standard deviation     0.63815 0.61964 0.59947 0.58447 0.57195 0.55097 0.53820
## Proportion of Variance 0.00582 0.00549 0.00513 0.00488 0.00467 0.00434 0.00414
## Cumulative Proportion  0.92254 0.92802 0.93316 0.93804 0.94271 0.94705 0.95118
##                           PC43    PC44    PC45    PC46    PC47    PC48    PC49
## Standard deviation     0.52029 0.51211 0.49533 0.48712 0.47079 0.44565 0.41879
## Proportion of Variance 0.00387 0.00375 0.00351 0.00339 0.00317 0.00284 0.00251
## Cumulative Proportion  0.95505 0.95880 0.96230 0.96569 0.96886 0.97170 0.97420
##                           PC50    PC51   PC52    PC53    PC54    PC55    PC56
## Standard deviation     0.40556 0.39328 0.3925 0.38502 0.36669 0.36205 0.33734
## Proportion of Variance 0.00235 0.00221 0.0022 0.00212 0.00192 0.00187 0.00163
## Cumulative Proportion  0.97655 0.97876 0.9810 0.98308 0.98500 0.98687 0.98850
##                           PC57    PC58    PC59    PC60    PC61    PC62    PC63
## Standard deviation     0.32150 0.30744 0.28898 0.28186 0.27274 0.25622 0.24118
## Proportion of Variance 0.00148 0.00135 0.00119 0.00113 0.00106 0.00094 0.00083
## Cumulative Proportion  0.98998 0.99133 0.99252 0.99365 0.99472 0.99565 0.99649
##                           PC64    PC65    PC66    PC67    PC68   PC69    PC70
## Standard deviation     0.23024 0.21442 0.19886 0.19371 0.17927 0.1677 0.09833
## Proportion of Variance 0.00076 0.00066 0.00056 0.00054 0.00046 0.0004 0.00014
## Cumulative Proportion  0.99724 0.99790 0.99846 0.99900 0.99946 0.9999 1.00000
```

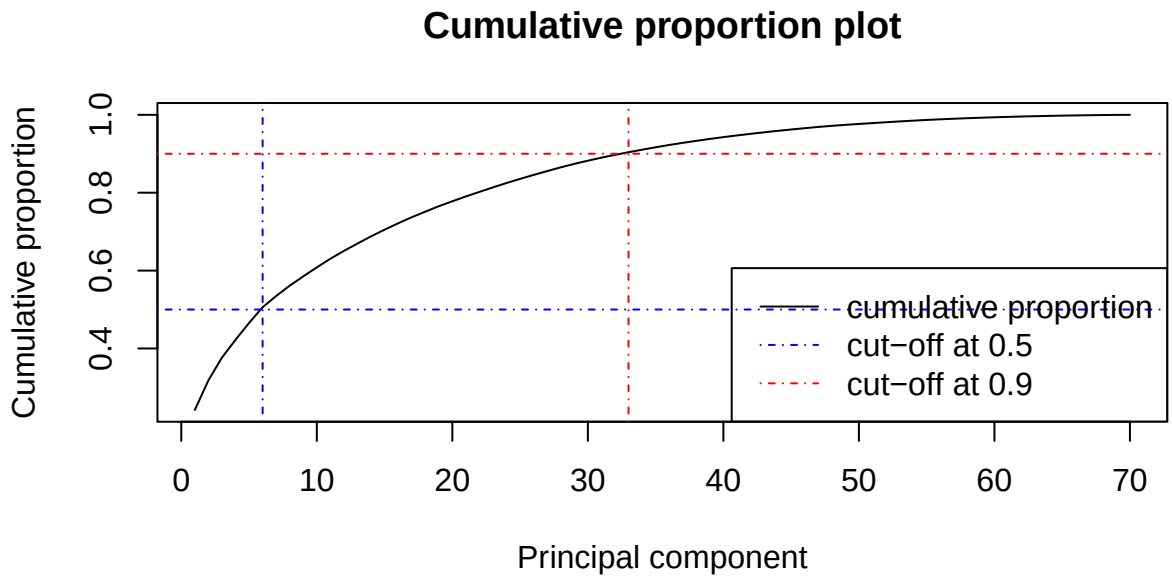
```
# the proportion explained by the first 3 PCs
perc.expl <- pca.patients$sdev^2 / sum(pca.patients$sdev^2)
sum(perc.expl[1:3])
```

```
## [1] 0.3766231
```

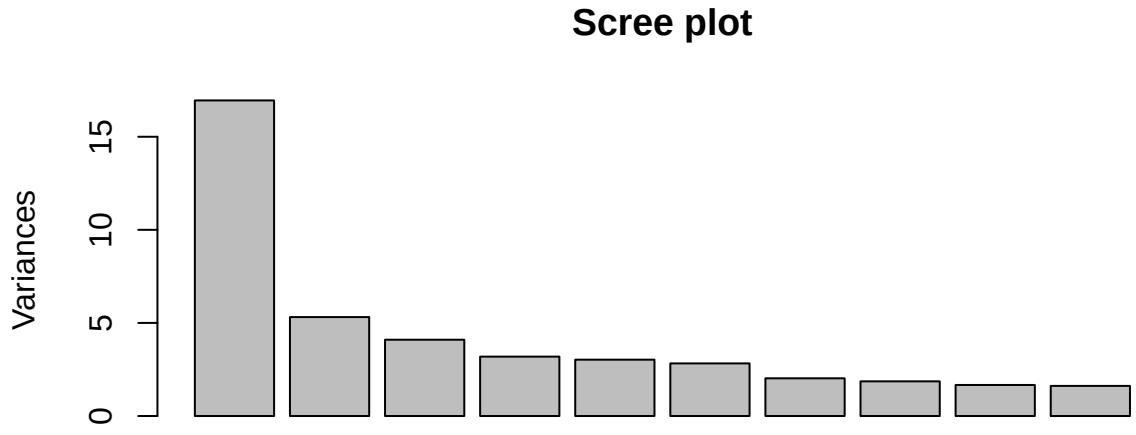
```
# cumulative proportion plot
plot(summary(pca.patients)$importance[3, ],
     type = "line", main = "Cumulative proportion plot",
     xlab = "Principal component", ylab = "Cumulative proportion")
```

```
## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
## character
```

```
abline(h = 0.5, col = "blue", lty = 4)
abline(h = 0.9, col = "red", lty = 4)
abline(v = 6, col = "blue", lty = 4)
abline(v = 33, col = "red", lty = 4)
legend("bottomright",
       legend = c("cumulative proportion", "cut-off at 0.5", "cut-off at 0.9"),        col = c("black",
```




```
# scree plot
screeplot(pca.patients, main = "Scree plot")
```



```r
# subset of the first 3 PCs
pca.components <- pca.patients$x[, 1:3]

# logistic regressions on each PC without adjustment on other variables
fit.pc1 <- glm(nki$Event ~ pca.components[, 1],
               family = "binomial"(link = "logit"))

fit.pc2 <- glm(nki$Event ~ pca.components[, 2],
               family = "binomial"(link = "logit"))

fit.pc3 <- glm(nki$Event ~ pca.components[, 3],
               family = "binomial"(link = "logit"))

# logistic regression on each PC with adjustments
fit.pc1.adj <- glm(nki$Event ~ pca.components[, 1]
                   + nki$EstrogenReceptor + nki$Grade + nki$Age,
                   family = "binomial"(link = "logit"))

fit.pc2.adj <- glm(nki$Event ~ pca.components[, 2]
                   + nki$EstrogenReceptor + nki$Grade + nki$Age,
                   family = "binomial"(link = "logit"))

fit.pc3.adj <- glm(nki$Event ~ pca.components[, 3]
                   + nki$EstrogenReceptor + nki$Grade + nki$Age,
                   family = "binomial"(link = "logit"))
```

```r
# regression coefficients and p-values in unadjusted and adjusted models
# principal component 1
data.frame(coef = c(fit.pc1$coefficients[2], fit.pc1.adj$coefficients[2]),
           p.value = c(summary(fit.pc1)$coefficients[2, 4],
                       summary(fit.pc1.adj)$coefficients[2, 4]),
           row.names = c("unadjusted", "adjuested"))
```

```
##                  coef     p.value
## unadjusted 0.11768352 0.009425039
## adjuested  0.07215917 0.272381200
```

```r
# principal component 2
data.frame(coef = c(fit.pc2$coefficients[2], fit.pc2.adj$coefficients[2]),
           p.value = c(summary(fit.pc2)$coefficients[2, 4],
                       summary(fit.pc2.adj)$coefficients[2, 4]),
           row.names = c("unadjusted", "adjuested"))
```

```
##                    coef   p.value
## unadjusted -0.067166804 0.3885231
## adjuested   0.005164421 0.9550636
```

```r
# principal component 3
data.frame(coef = c(fit.pc3$coefficients[2], fit.pc3.adj$coefficients[2]),
           p.value = c(summary(fit.pc3)$coefficients[2, 4],
                       summary(fit.pc3.adj)$coefficients[2, 4]),
           row.names = c("unadjusted", "adjuested"))
```

```
##                  coef    p.value
## unadjusted 0.2435485 0.00863000
## adjuested  0.2183706 0.02454557
```

```
fit.pca.adj <- glm(nki$Event ~ pca.components[, 1] + pca.components[, 2]
                   + pca.components[, 3] + nki$EstrogenReceptor
                   + nki$Grade + nki$Age, family = "binomial"(link = "logit"))
summary(fit.pca.adj)
```

```
##
## Call:
## glm(formula = nki$Event ~ pca.components[, 1] + pca.components[,
##     2] + pca.components[, 3] + nki$EstrogenReceptor + nki$Grade +
##     nki$Age, family = binomial(link = "logit"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8178  -0.8782  -0.5907   1.1119   2.1029
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    2.24591    1.69668   1.324   0.1856
## pca.components[, 1]            0.09426    0.07333   1.285   0.1986
## pca.components[, 2]           -0.04621    0.10305  -0.448   0.6539
## pca.components[, 3]            0.22688    0.09899   2.292   0.0219 *
## nki$EstrogenReceptorPositive   0.09780    0.69101   0.142   0.8875
## nki$GradePoorly diff           0.15228    0.48885   0.312   0.7554
## nki$GradeWell diff            -0.63836    0.55341  -1.153   0.2487
## nki$Age                       -0.06788    0.03632  -1.869   0.0616 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 183.32  on 143  degrees of freedom
## Residual deviance: 162.25  on 136  degrees of freedom
## AIC: 178.25
##
## Number of Fisher Scoring iterations: 4
```

Problem 3.c (8 points)

Use plots to compare with the correlation structure observed in problem 2.a and to examine how well the dataset may explain your outcome. Discuss your findings and suggest any further steps if needed.

Solutions:

We visualize the first 3 principle components using `fviz_pca_ind()` and `fviz_pca_biplot()` functions in `factoextra` package.

We start with analysing the results from individual scatter plots. The data points in these plots are the individual counties. They are artificially grouped and coloured by quartile. The x-axis in the first plot is the first principle component and the value between brackets is the proportion of the variance it explains. The y-axis in the first plot is the second component and in the second graph the counties are projected on the plane formed by the second and third components. Same applies for the third plot.

Looking at all plots we can see that the ellipses overlap. To overcome this we could just look at the largest and lowest quartiles. Furthermore, each component simply describes the variation of the dataset. A dataset which contains a few variables that are associated with the outcome and a lot more variables that aren't won't be very good at separating individual observations of the outcome. This isn't entirely the case with our dataset as while the ellipses overlap for the first two principal components and first and third principal components show that they do separate slightly in the right order along the x and y axes.

Looking in closely, centroids are separated in the right order along the x-axis for the first principal component and marginally so along the y-axis for the second principal component. However, in the second graph the ellipses completely overlap. Removing a few variables which weren't associated with our outcome of interest in the correlation matrix can probably improve this matter. Finally, first and second components and first and third components contain the large variation in our data it is therefore a good idea to examine the subset of most extreme values for the first two components as they are likely to contain outliers.

Biplot is a visual representation of the linear composition of the components. The dots are the same data points then in the previous graph but without colour and each arrow represents the direction of each variable in the the 2D plane of the two components. Here we make 3 biplots for each two 2 of the first 3 principle components.

For the first component we would look at the variables with greatest magnitude along the x-axis in the first biplot, i.e. furthest to the left and furthest to the right. In this case, first component mainly assigns large positive values to `MELK` and `RFC4`, and large negative values to `STK32B`, `SCUBE2` and `TGFB3`.

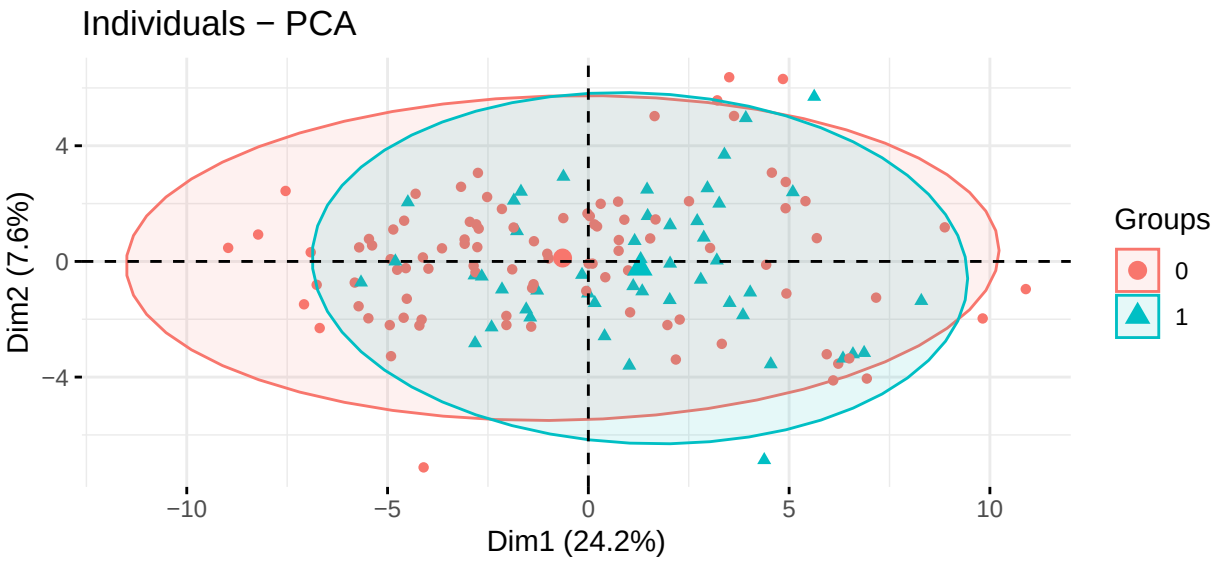
For the second component we look at the variables with the greatest magnitude along the y-axis in the first biplot, i.e. the highest and lowest variables. Here, for this component we observe that `SLC2A3`, `COL4A2`, `PECI` and `PECI.1` carry the most weight.

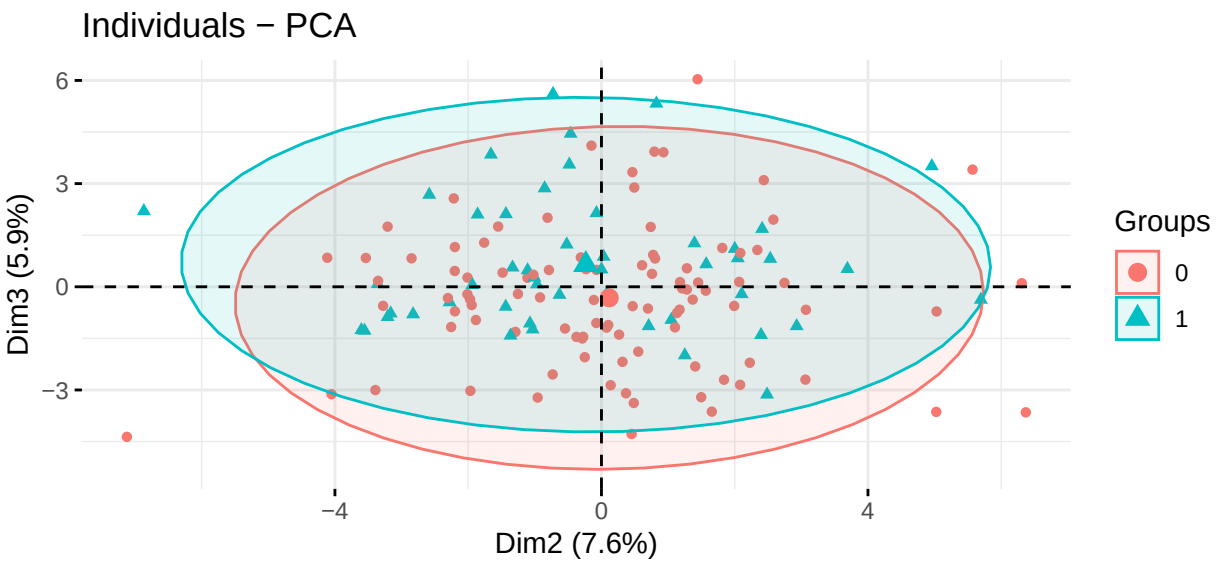
For the third component we look at the variables with the greatest magnitude along the y-axis in the second biplot, i.e. the highest and lowest variables. `IGFBP5`, `IGFBP5.1`, `OXCT1` and `GPR180` are observed to carry the most weight for this component.

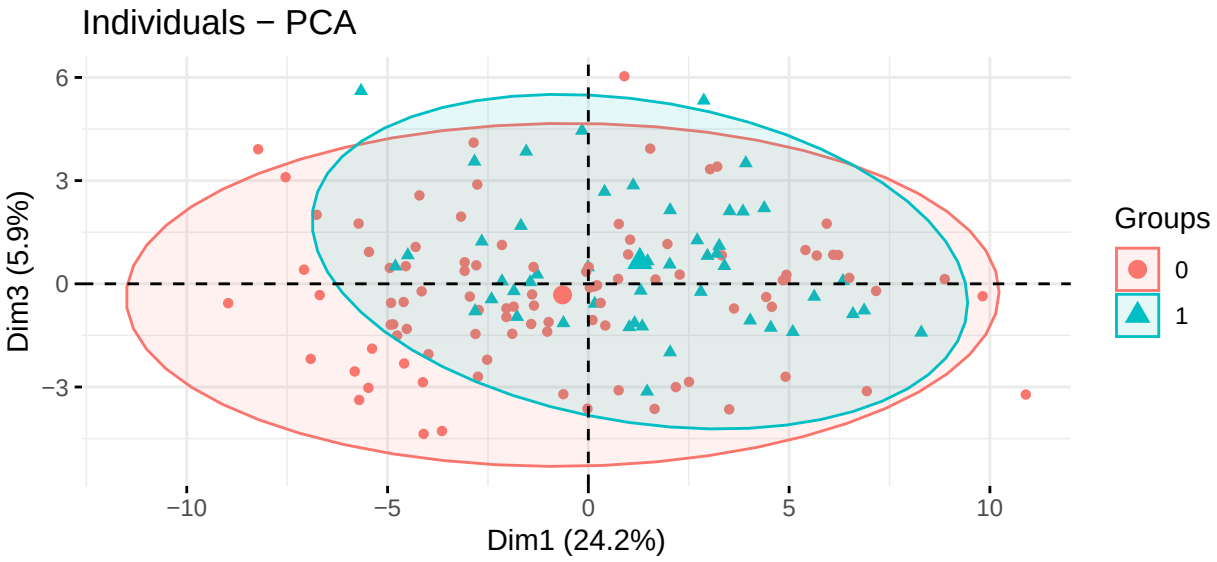
The results from different biplots are consistent with each other. They roughly follow the blocks that we observed in the correlation plot which isn't surprising given that the components are the eigenvectors of the variance-covariance matrix of the the predictor variables. We also observe genes with similar names always have similar directions and weights, which agrees with our findings from yje correlation plot in 3.c.

```
# individual scatter plots
# PC1 vs PC2
plot1 <- fviz_pca_ind(pca.patients, geom = 'point',
                      habillage = nki$Event, addEllipses = TRUE)
# PC2 vs PC3
```

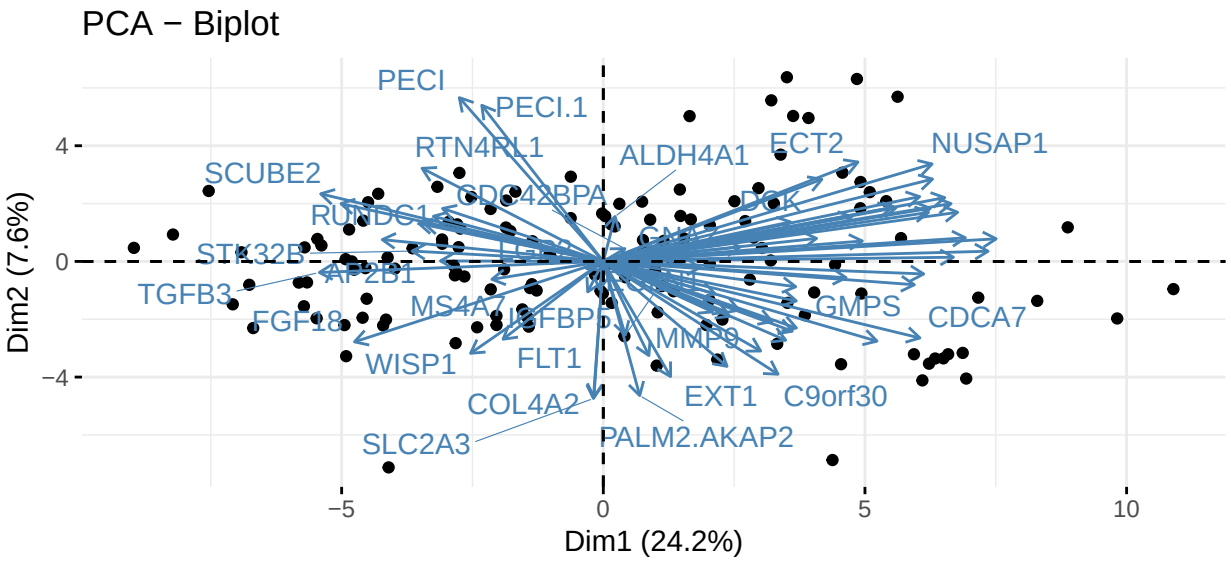
```
plot3 <- fviz_pca_ind(pca.patients, geom = 'point', axes = c(2, 3),
                      habillage = nki$Event, addEllipses = TRUE)
# PC1 vs PC3
plot5 <- fviz_pca_ind(pca.patients, geom = 'point', axes = c(1, 3),
                      habillage = nki$Event, addEllipses = TRUE)

grid.arrange(plot1, plot3, plot5)
```

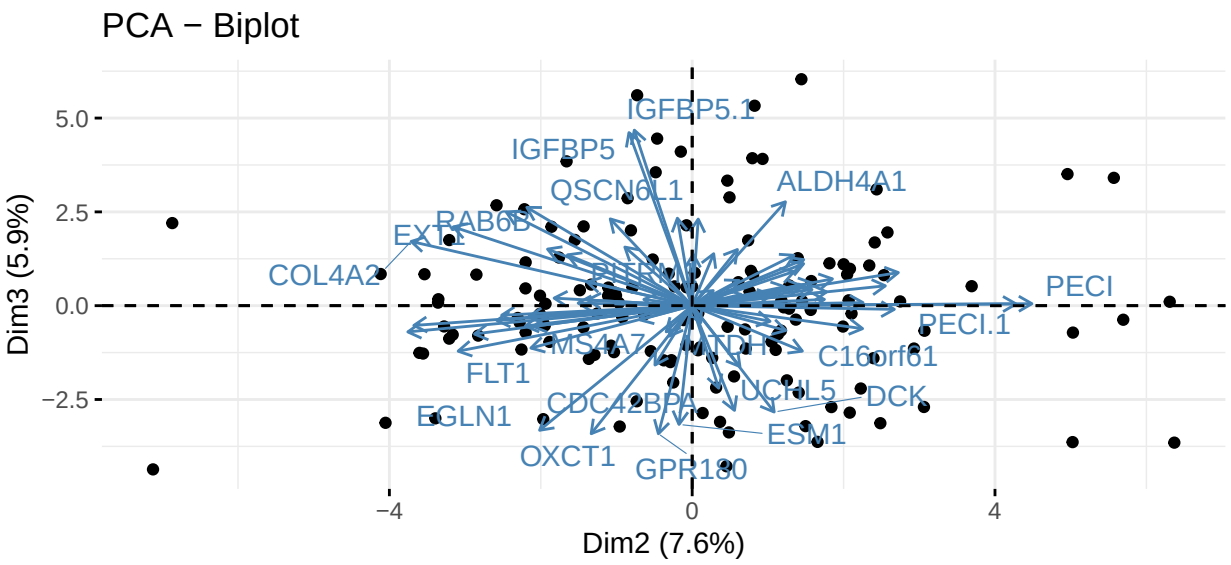

Individuals − PCA
Dim2 (7.6%)
Dim1 (24.2%)
Groups
0
1

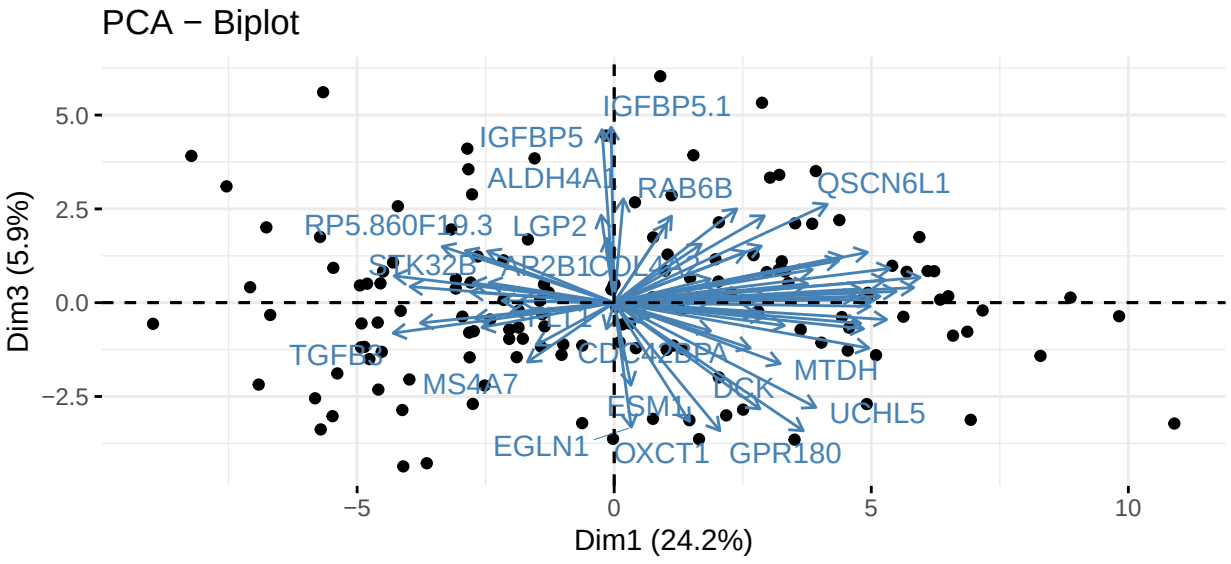

Individuals − PCA
Dim3 (5.9%)
Dim2 (7.6%)
Groups
0
1


Individuals − PCA
Dim3 (5.9%)
Dim1 (24.2%)
Groups
0
1

```r
# biplots
# PC1 vs PC2
plot2 <- fviz_pca_biplot(pca.patients, geom = 'point', repel = TRUE)
# PC2 vs PC3
plot4 <- fviz_pca_biplot(pca.patients, geom = 'point', axes = c(2, 3),
                         repel = TRUE)
# PC1 vs PC3
plot6 <- fviz_pca_biplot(pca.patients, geom = 'point', axes = c(1, 3),
                         repel = TRUE)

grid.arrange(plot2, plot4, plot6)
```


PCA − Biplot
PECI
PECI.1
RTN4RL1
ALDH4A1
ECT2
NUSAP1
SCUBE2
CDC42BPA
DCK
RUNDC1
STK32B
AP2B1
TGFB3
MS4A7
GMPS
CDCA7
FGF18
IGFBP5
MMP9
WISP1
FLT1
COL4A2
EXT1
C9orf30
SLC2A3
PALM2.AKAP2
Dim2 (7.6%)
Dim1 (24.2%)


PCA − Biplot
IGFBP5.1
IGFBP5
QSCN6L1
ALDH4A1
RAB6B
EXT1
COL4A2
PECI
PECI.1
MS4A7
MTDH
C16orf61
FLT1
UCHL5
DCK
CDC42BPA
EGLN1
ESM1
OXCT1
GPR180
Dim3 (5.9%)
Dim2 (7.6%)


PCA − Biplot
IGFBP5.1
IGFBP5
ALDH4A1
RAB6B
QSCN6L1
RP5.860F19.3
LGP2
STK32B
TGFB3
CDC42BPA
MTDH
MS4A7
DCK
ESM1
UCHL5
EGLN1
OXCT1
GPR180
Dim3 (5.9%)
Dim1 (24.2%)

Problem 3.d (11 points)

Based on the models we examined in the labs, fit an appropriate model with the aim to provide the most accurate prognosis you can for patients. Discuss and justify your decisions.

Solution:

As mentioned in 3.b, PCA, with its poor model performance, is not an appropriate method for our data. Thus, in this part, we tend to apply different methods to fit a well-performed model which can provide the most accurate prognosis. The candidate methods are `Lasso regression`, `Ridge Regression`, `Backward stepwise selection` and `Forward stepwise selection`. Moreover, for Lasso regression and Ridge regression, we test two independent variable sets to determine if we should use gene expression variables only (4 models in total).

We start with Lasso regression and Ridge regression on two variables sets. Since there are factors in the independent variable set, we define and apply a helper function `prepare.gllmnet()` (from Lab 4) to covert the corresponding data table to a matrix.

Next, as what we did in Problem 1, we use `createDataPartion()` function to split the training and testing sets with a ratio of $7:3$. After that, we run Lasso and Ridge regressions using cross-validation over all independent variables and gene expression variables only. To compare the model performances, we get the maximised AUC of each model with the optimal λ.

Furthermore, we also compute the AUC scores of 4 regression models and plot ROC curves in the same graph, using `roc()` function. From both data frame of AUCs and the plot, we observe the performances of different model vary dramatically, where the Ridge regression over all variables beats other competitors. On the other hand, looking at Ridge regression over different variable sets, we conclude that it is better to take all variables into consideration rather than gene expression only, in terms of AUC.

Note that for Lasso regression over gene expression variables only, though it has the largest AUC on training set (0.786 in 3 decimal places), it obtains the lowest AUC on testing set, indicating the problem of over-fitting. As a result, we reject this model.

We repeat the same steps using stepwise selection models and compare their performances on testing set with penalized models using AUC and ROC curves. According to the results, both stepwise selection models have lower AUCs, implying worse model accuracy. Another problem of stepwise selection is the convergence of it doesn't meet during iterations.

In conclusion, we pick up **Ridge regression over all variables** as the most accurate model with respect to AUC score.

```
# helper function in Lab4
prepare.glmnet <- function(data, formula=~ .) {
                ## create the design matrix to deal correctly with factor variables,
                ## without losing rows containing NAs
                old.opts <- options(na.action='na.pass')
                x <- model.matrix(formula, data)
                options(old.opts)

                ## remove the intercept column, as glmnet will add one by default
                x <- x[, -match("(Intercept)", colnames(x))]
                return(x)
}
```

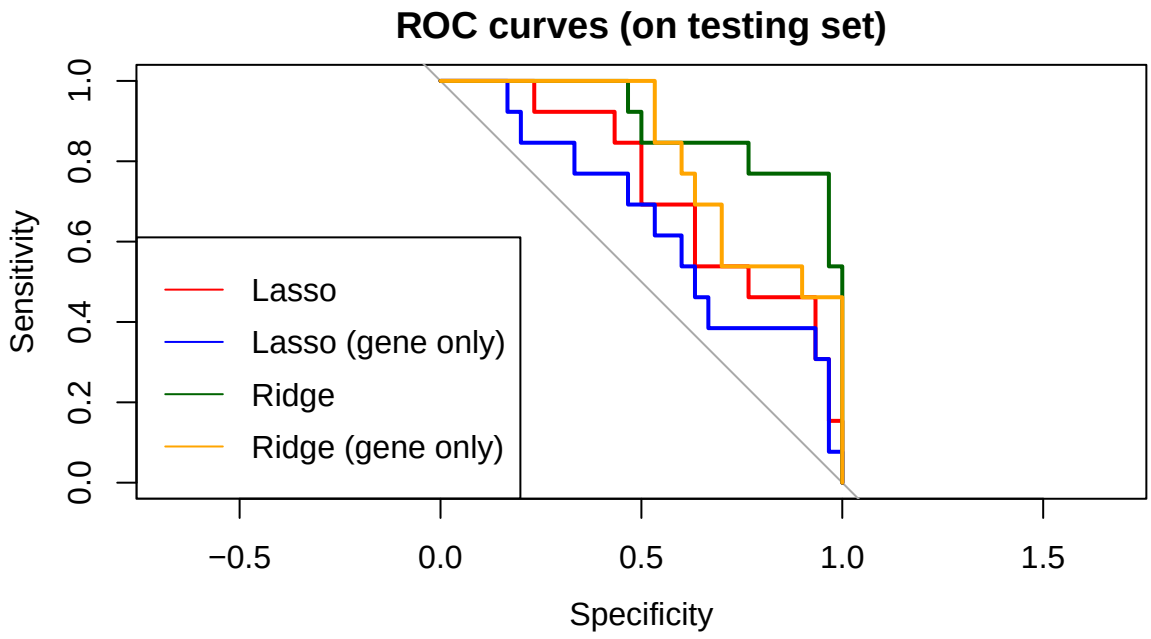
```
# define x and y matrices
x.mat <- prepare.glmnet(nki, ~ . - Event)
```

```r
y.mat <- nki$Event

# create training and testing matrices
set.seed(984065)
train.idx <- createDataPartition(y = nki$Event, p = 0.7)$Resample1
train.x.mat <- x.mat[train.idx, ]
test.x.mat <- x.mat[-train.idx, ]
train.y.mat <- y.mat[train.idx]
test.y.mat <- y.mat[-train.idx]
```

```r
# Lasso regressions using CV
set.seed(984065)
fit.cv.Lasso <- cv.glmnet(train.x.mat, train.y.mat,
                          family = "binomial", type.measure = "auc")
fit.cv.Lasso.gene <- cv.glmnet(train.x.mat[, -c(1:6)], train.y.mat,
                               family = "binomial", type.measure = "auc")

# AUCs with lambda.min
Lasso.auc <- fit.cv.Lasso$cvm[fit.cv.Lasso$index[1]]
Lasso.gene.auc <- fit.cv.Lasso.gene$cvm[fit.cv.Lasso.gene$index[1]]

# Ridge regression using CV
fit.cv.Ridge <- cv.glmnet(train.x.mat, train.y.mat, alpha = 0,
                          family = "binomial", type.measure = "auc")
fit.cv.Ridge.gene <- cv.glmnet(train.x.mat[, -c(1:6)], train.y.mat, alpha = 0,
                               family = "binomial", type.measure = "auc")

# AUCs with lambda.min
Ridge.auc <- fit.cv.Ridge$cvm[fit.cv.Ridge$index[1]]
Ridge.gene.auc <- fit.cv.Ridge.gene$cvm[fit.cv.Ridge.gene$index[1]]
```

```r
# predictive values of Lasso and Ridge regressions on testing set
pred.Lasso.test <- predict(fit.cv.Lasso, newx = test.x.mat,
                           s = fit.cv.Lasso$lambda.min, type = "response")
pred.Lasso.gene.test <- predict(fit.cv.Lasso.gene, newx = test.x.mat[, -c(1:6)],
                                s = fit.cv.Lasso.gene$lambda.min, type = "response")
pred.Ridge.test <- predict(fit.cv.Ridge, newx = test.x.mat,
                           s = fit.cv.Ridge$lambda.min, type = "response")
pred.Ridge.gene.test <- predict(fit.cv.Ridge.gene, newx = test.x.mat[, -c(1:6)],
                                s = fit.cv.Ridge.gene$lambda.min, type = "response")

# AUC and ROC curves
invisible({capture.output({
Lasso.test.auc <- roc(test.y.mat ~ pred.Lasso.test,
                      plot = TRUE, main = "ROC curves (on testing set)",
                      col = "red", xlim = c(0, 1))$auc
Lasso.gene.test.auc <- roc(test.y.mat ~ pred.Lasso.gene.test,
                           plot = TRUE, add = TRUE, col = "blue")$auc
Ridge.test.auc <- roc(test.y.mat ~ pred.Ridge.test,
                      plot = TRUE, add = TRUE, col = "darkgreen")$auc
Ridge.gene.test.auc <- roc(test.y.mat ~ pred.Ridge.gene.test,
                           plot = TRUE, add = TRUE, col = "orange")$auc
})})
```

```
legend("bottomleft", legend = c("Lasso", "Lasso (gene only)",
                                "Ridge", "Ridge (gene only)"),
       col = c("red", "blue", "darkgreen", "orange"),
       lwd = c(1, 1, 1, 1))
```

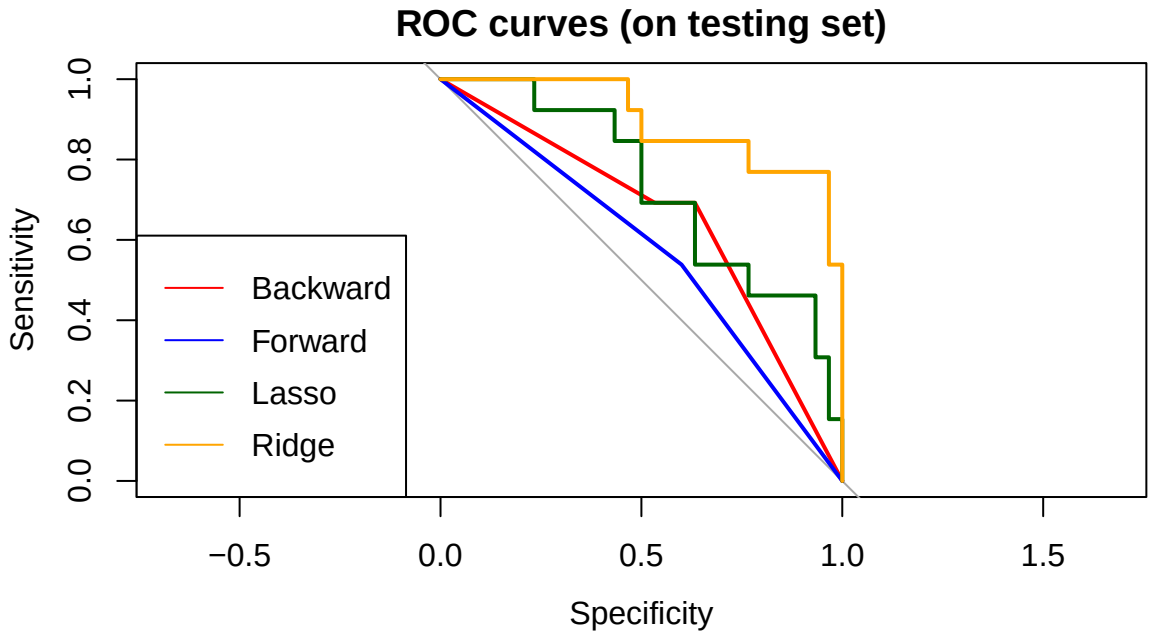



```
# report the AUCs
data.frame(train.AUC = c(Lasso.auc, Lasso.gene.auc, Ridge.auc, Ridge.gene.auc),
           modelsize = c(fit.cv.Lasso$nzero[fit.cv.Lasso$index[1]],
                         fit.cv.Lasso.gene$nzero[fit.cv.Lasso.gene$index[1]],
                         fit.cv.Ridge$nzero[fit.cv.Ridge$index[1]],
                         fit.cv.Ridge.gene$nzero[fit.cv.Ridge.gene$index[1]]),
           test.AUC = c(Lasso.test.auc, Lasso.gene.test.auc,
                        Ridge.test.auc, Ridge.gene.test.auc),
           row.names = c("Lasso", "Lasso (gene only)",
                         "Ridge", "Ridge (gene only)"))
```

```
##                   train.AUC modelsize  test.AUC
## Lasso             0.7050330        49 0.7307692
## Lasso (gene only) 0.7856554        46 0.6487179
## Ridge             0.7258242        76 0.8948718
## Ridge (gene only) 0.7203835        70 0.8153846
```

```
# stepwise selection
# training and testing sets
train.x <- nki[train.idx, -c(1)]
test.x <- nki[-train.idx, -c(1)]
train.y <- nki$Event[train.idx]
test.y <- nki$Event[-train.idx]

# define null and full models
```

```r
null.model <- glm(Event ~ 1, data = nki[train.idx, ], family = "binomial")
full.model <- glm(Event ~ ., data = nki[train.idx, ], family = "binomial")

# define stepwise selection models
fit.B <- stepAIC(full.model, scope = list(lower = null.model),
                 direction = "backward", trace = FALSE)
fit.S <- stepAIC(null.model, scope = list(upper = full.model),
                 direction = "forward", trace = FALSE)

# apply stepwise models on testing set
pred.B.test <- predict(fit.B, newdata = test.x, type = "response")
pred.S.test <- predict(fit.S, newdata = test.x, type = "response")

# AUCs of stepwise models on testing set
invisible({capture.output({
B.test.auc <- roc(test.y ~ pred.B.test, plot  = TRUE,
                  main = "ROC curves (on testing set)",
                  col = "red", xlim = c(0, 1))$auc
S.test.auc <- roc(test.y ~ pred.S.test, plot = TRUE,
                  add = TRUE, col = "blue")$auc
# add ROC curves for Lasso and Ridge regressions
roc(test.y.mat ~ pred.Lasso.test, plot = TRUE, add = TRUE, col = "darkgreen")
roc(test.y.mat ~ pred.Ridge.test, plot = TRUE, add = TRUE, col = "orange")
})})
legend("bottomleft", legend = c("Backward", "Forward", "Lasso", "Ridge"),
       col = c("red", "blue", "darkgreen", "orange"), lwd = c(1, 1, 1, 1))
```




```r
# report the AUCs for testing set
data.frame(modelsize = c(fit.cv.Lasso$nzero[fit.cv.Lasso$index[1]],
                         fit.cv.Ridge$nzero[fit.cv.Ridge$index[1]],
                         length(fit.B$coefficients) - 1,
```

```
                        length(fit.S$coefficients) - 1),
           test.AUC = c(Lasso.test.auc, Ridge.test.auc,
                        B.test.auc, S.test.auc),
           row.names = c("Lasso", "Ridge", "Backward", "Forward"))
```

```
##          modelsize  test.AUC
## Lasso           49 0.7307692
## Ridge           76 0.8948718
## Backward        21 0.6474359
## Forward         21 0.5692308
```