compgen2021: Week 3 exercises

Yuna Son

Exercises for Week 3

```
# have to load the data due to the slow knitting process...
load("week3.RData")
```

Classification

For this set of exercises we will be using the gene expression and patient annotation data from the glioblastoma patient. You can read the data as shown below:

```
library(compGenomRData)
# get file paths
fileLGGexp=system.file("extdata",
                       "LGGrnaseq.rds",
                       package="compGenomRData")
fileLGGann=system.file("extdata",
                       "patient2LGGsubtypes.rds",
                       package="compGenomRData")
# gene expression values
gexp=readRDS(fileLGGexp)
# patient annotation
patient=readRDS(fileLGGann)
```

1. Our first task is to not use any data transformation and do classification. Run the k-NN classifier on the data without any transformation or scaling. What is the effect on classification accuracy for k-NN predicting the CIMP and noCIMP status of the patient? [Difficulty: **Beginner**]

solution:

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.5

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.0.5

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.5
```

```r
set.seed(3031) # set the random number seed for reproducibility


# transpose the data set
tgexp <- t(gexp)

# merge the patient and gexp data.
tgexp=merge(patient,tgexp,by="row.names")
row.names(tgexp) <- tgexp[, 1]
tgexp <- tgexp[, -1]

# get indices for 70% of the data set
#intrain <- createDataPartition(y = tgexp[,1], p= 0.7)[[1]]
# seperate test and training sets
#training <- tgexp[intrain,]
#testing <- tgexp[-intrain,]
```

```r
knnFit=knn3(x=tgexp[,-1], # data set
            y=tgexp[,1], # data set class labels
            k=5)


# predictions on the data set
trainPred=predict(knnFit,tgexp[,-1],type="class")

# compare the predicted labels to real labels
# get different performance metrics
confusionMatrix(data=tgexp[,1],reference=trainPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     87      5
##     noCIMP   16     76
##
##                Accuracy : 0.8859
##                  95% CI : (0.8308, 0.9279)
##     No Information Rate : 0.5598
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7717
##
##  Mcnemar's Test P-Value : 0.0291
##
##             Sensitivity : 0.8447
##             Specificity : 0.9383
##          Pos Pred Value : 0.9457
##          Neg Pred Value : 0.8261
##              Prevalence : 0.5598
##          Detection Rate : 0.4728
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8915
```

```
##
##        'Positive' Class : CIMP
##
```

```
# predictions on the test set, return class labels
testPred=predict(knnFit,tgexp[,-1],type="class")

# compare the predicted labels to real labels
# get different performance metrics
confusionMatrix(data=tgexp[,1],reference=testPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     87      5
##     noCIMP   17     75
##
##                Accuracy : 0.8804
##                  95% CI : (0.8246, 0.9235)
##     No Information Rate : 0.5652
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7609
##
##  Mcnemar's Test P-Value : 0.01902
##
##             Sensitivity : 0.8365
##             Specificity : 0.9375
##          Pos Pred Value : 0.9457
##          Neg Pred Value : 0.8152
##              Prevalence : 0.5652
##          Detection Rate : 0.4728
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8870
##
##        'Positive' Class : CIMP
##
```

```
### k values = 2 ###
knnFit=knn3(x=tgexp[,-1], # data set
            y=tgexp[,1], # data set class labels
            k=2)


# predictions on the data set
trainPred=predict(knnFit,tgexp[,-1],type="class")

# compare the predicted labels to real labels
# get different performance metrics
confusionMatrix(data=tgexp[,1],reference=trainPred)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     84      8
##     noCIMP   12     80
##
##                Accuracy : 0.8913
##                  95% CI : (0.8371, 0.9323)
##     No Information Rate : 0.5217
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7826
##
##  Mcnemar's Test P-Value : 0.5023
##
##             Sensitivity : 0.8750
##             Specificity : 0.9091
##          Pos Pred Value : 0.9130
##          Neg Pred Value : 0.8696
##              Prevalence : 0.5217
##          Detection Rate : 0.4565
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8920
##
##        'Positive' Class : CIMP
##
```

```
# predictions on the test set, return class labels
testPred=predict(knnFit,tgexp[,-1],type="class")

# compare the predicted labels to real labels
# get different performance metrics
confusionMatrix(data=tgexp[,1],reference=testPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     85      7
##     noCIMP   11     81
##
##                Accuracy : 0.9022
##                  95% CI : (0.8498, 0.941)
##     No Information Rate : 0.5217
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8043
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.8854
##             Specificity : 0.9205
##          Pos Pred Value : 0.9239
##          Neg Pred Value : 0.8804
```

```
##              Prevalence : 0.5217
##          Detection Rate : 0.4620
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9029
##
##        'Positive' Class : CIMP
##
```

k values is affecting the accuracy. When k value is changed, the accuracy is also changed.

2. Bootstrap resampling can be used to measure the variability of the prediction error. Use bootstrap resampling with k-NN for the prediction accuracy. How different is it from cross-validation for different ks? [Difficulty: **Intermediate**]

solution:

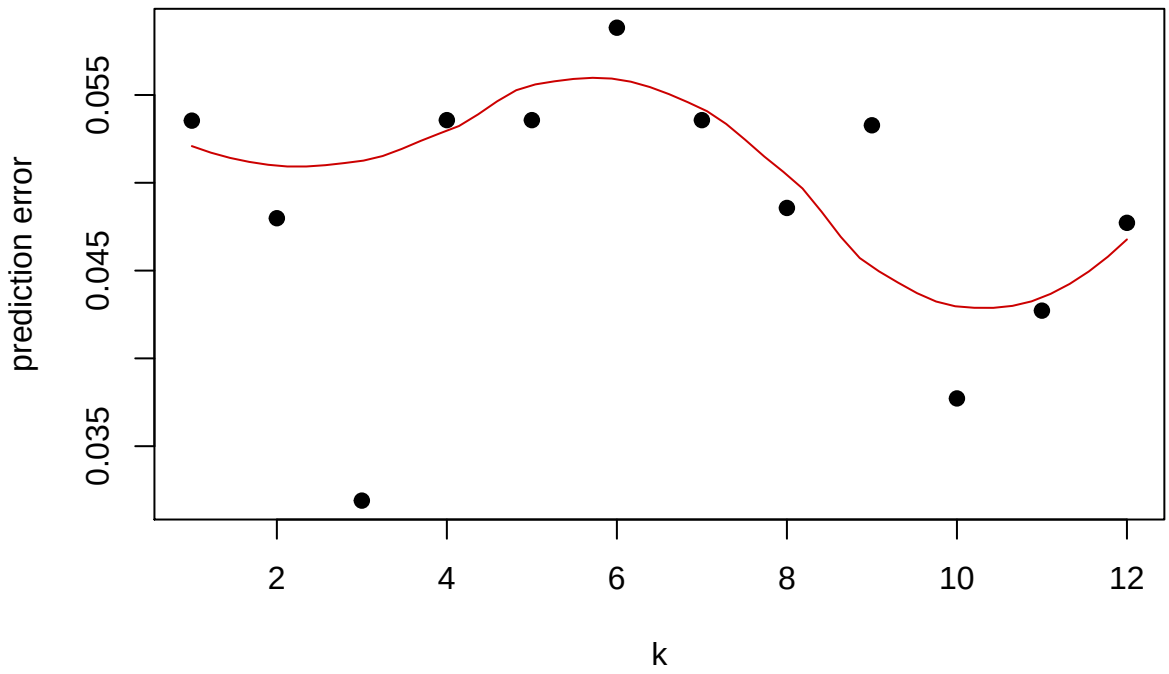
```
# Data transformation
gexp=log10(gexp+1)
# transpose the data set
tgexp <- t(gexp)

### MEMORY ISSUES ###
# Filter the data (choose to take the top 1000 variable predictors.)

SDs=apply(tgexp,2,sd )
topPreds=order(SDs,decreasing = TRUE)[1:1000]
tgexp=tgexp[,topPreds]


# Center the data and scale the data
processCenter=preProcess(tgexp, method = c("center"))
tgexp=predict(processCenter,tgexp)

# create a filter for removing higly correlated variables
# if two variables are highly correlated only one of them
# is removed
corrFilt=preProcess(tgexp, method = "corr",cutoff = 0.9)
tgexp=predict(corrFilt,tgexp)



# make a single data frame
tgexp=merge(patient,tgexp,by="row.names")

# push sample ids back to the row names
rownames(tgexp)=tgexp[,1]
tgexp=tgexp[,-1]


# get indices for 70% of the data set
#intrain <- createDataPartition(y = tgexp[,1], p= 0.7)[[1]]
# seperate test and training sets
#training <- tgexp[intrain,]
#testing <- tgexp[-intrain,]
```

We are doing Cross-validation first.

```
set.seed(17)
# this method controls everything about training
# we will just set up 10-fold cross validation
trctrl <- trainControl(method = "cv",number=10)

# we will now train k-NN model
knn_fit <- train(subtype~., data = tgexp,
                 method = "knn",
                 trControl=trctrl,
                 tuneGrid = data.frame(k=1:12))

# best k value by cross-validation accuracy
knn_fit$bestTune
```

```
##   k
## 3 3
```

```
# plot k vs prediction error
plot(x=1:12,1-knn_fit$results[,2],pch=19,
     ylab="prediction error",xlab="k")
lines(loess.smooth(x=1:12,1-knn_fit$results[,2],degree=2),
      col="#CC0000")
```



```r
# Accuracy from different k values (from k = 1 to k = 12)
knn_fit$results[,2]
```

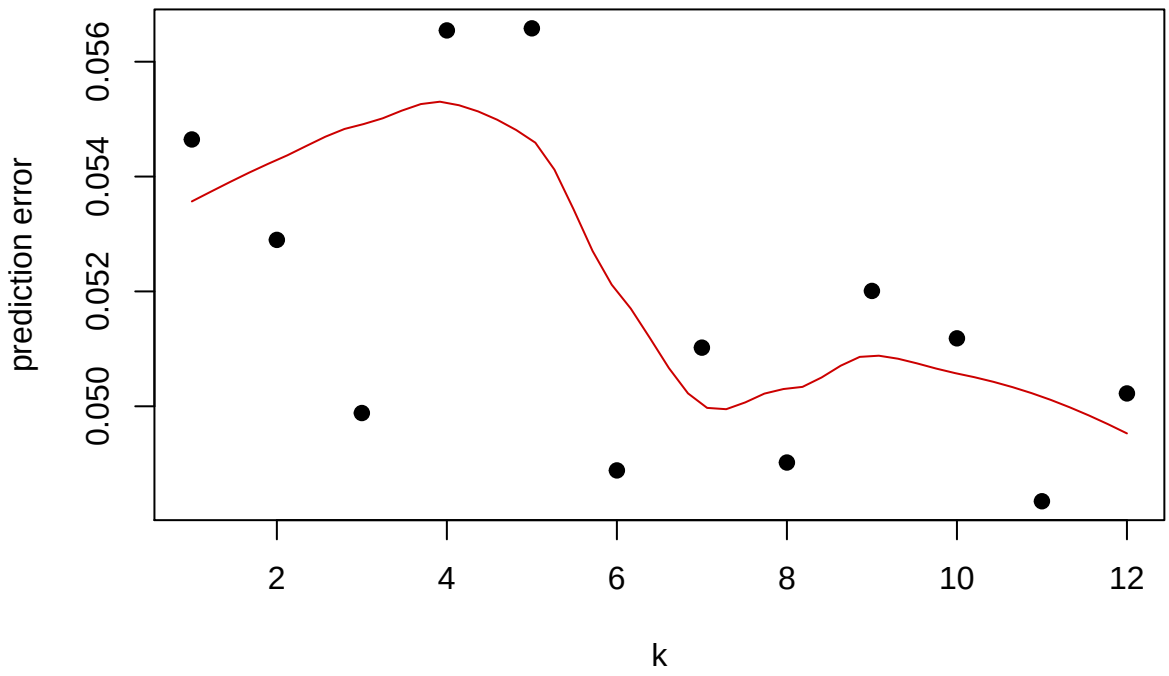
```
##  [1] 0.9464620 0.9520175 0.9680994 0.9464327 0.9464327 0.9411696 0.9464327
##  [8] 0.9514327 0.9467251 0.9622807 0.9572807 0.9522807
```

And, we are doing Boostrap resampling and check the prediction error.

```r
set.seed(17)
# this method controls everything about training
# we will just set up 100 bootstrap samples and for each
# bootstrap OOB samples to test the error
trctrl <- trainControl(method = "boot",number=20,
                       returnResamp="all")

# we will now train k-NN model
knn_fit2 <- train(subtype~., data = tgexp,
                  method = "knn",
                  trControl=trctrl,
                  tuneGrid = data.frame(k=1:12))



# best k value by bootstrap resampling accuracy
knn_fit2$bestTune
```

```
##     k
## 11 11
```

```r
# plot k vs prediction error
plot(x=1:12,1-knn_fit2$results[,2],pch=19,
     ylab="prediction error",xlab="k")
lines(loess.smooth(x=1:12,1-knn_fit2$results[,2],degree=2),
      col="#CC0000")
```




```
# Accuracy from different k values (from k = 1 to k = 12)
knn_fit2$results[,2]
```

```
##  [1] 0.9453541 0.9471026 0.9501179 0.9434556 0.9434194 0.9511168 0.9489789
##  [8] 0.9509796 0.9479917 0.9488172 0.9516535 0.9497762
```

The two methods showed very different k values for the optimized prediction errors.

3. There are a number of ways to get variable importance for a classification problem. Run random forests on the classification problem above. Compare the variable importance metrics from random forest and the one obtained from DALEX applied on the random forests model. How many variables are the same in the top 10? [Difficulty: **Advanced**]

solution:

```
# I don't know why it is not working. other data set from the example in the DALEX documents seem to wo

### RANDOM FOREST Model ###

set.seed(17)
library(DALEX)
```

```
## Warning: package 'DALEX' was built under R version 4.0.5
```

```
## Welcome to DALEX (version: 2.3.0).
## Find examples and detailed introduction at: http://ema.drwhy.ai/
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.5
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

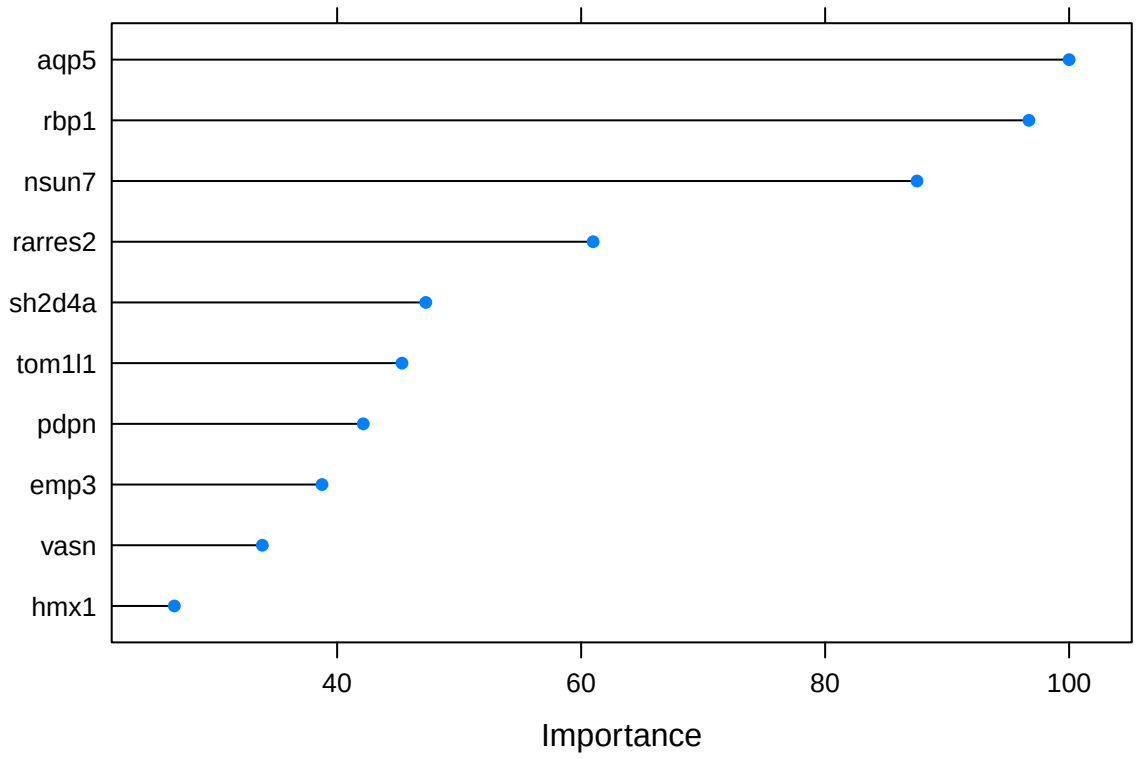
```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# Had to format column names due to some issues - not recognized by certain functions.
tgexp2 <- janitor::clean_names(tgexp)

#options(digits = 5)
# Random forest model on the data
#rfFit <- randomForest(subtype ~., data = tgexp2, importance = TRUE)

# DALEX explain function
#explained_rf2 <- explain(rfFit, data =tgexp2[, -1], y=as.numeric(tgexp2[, 1]))

# Calculate variable importance
#vi_dalex=feature_importance(explained_rf2,n_sample=50,type="difference")

#set.seed(1980)
#result <- model_parts(explainer = explained_rf2,
#        loss_function = loss_root_mean_square,
#                      B = 1)
#head(vi_dalex, 11)

# Order by the drop out loss levels
t <- vi_dalex[order(vi_dalex$dropout_loss, decreasing = TRUE),]
# top 10 variables
head(t$variable, 11)
```

```
##  [1] "_full_model_" "usp9y"        "xist"         "gstt1"        "hoxa7"
##  [6] "hoxc10"       "shox2"        "tsix"         "c5orf38"      "dao"
## [11] "hoxd10"
```

```
# train random forest
set.seed(519)
# this method controls everything about training
# we will just set up 5-fold cross validation
```

```
trctrl <- trainControl(method = "cv",number=5)
# we will now train random forest model
rfFit2 <- train(subtype~.,
                data = tgexp2,
                method = "ranger",
                trControl=trctrl,
                importance="permutation", # calculate importance
                tuneGrid = data.frame(mtry=100,
                                      min.node.size = 1,
                                      splitrule="gini")
                 )

# default mode of variable importance from ranger

varImport_direct <- varImp(rfFit2)
varImport_direct <- varImport_direct$importance
plot(varImp(rfFit2),top=10)
```



They are not matched genes for top 10 variable importances from two methods.

4. Come up with a unified importance score by normalizing importance scores from random forests and DALEX, followed by taking the average of those scores. [Difficulty: **Advanced**]

solution:

```r
# Merge the data by the gene names
vi_dalex_2 <- vi_dalex[vi_dalex$permutation == 10,]
row.names(vi_dalex_2) <- vi_dalex_2$variable
vi_dalex_2 <- vi_dalex_2[, c(-1, -2)]
merged_imp <- merge(varImport_direct, vi_dalex_2, all = T, by = 'row.names')
merged_imp <- merged_imp[!is.na(merged_imp$Overall),]


# Get the total importance scores from random forest
a <- sum(merged_imp$Overall)

# Get the total importance scores from DALEX
b <- sum(merged_imp$dropout_loss)

# Add column for calculating random forest and DALEX normalized scores
merged_imp$rf <- with(merged_imp, merged_imp$Overall / a)
merged_imp$dalex <- with(merged_imp, merged_imp$dropout_loss / b)


# Calculate the unified scores
merged_imp$uni <- with(merged_imp, (merged_imp$rf + merged_imp$dalex) / 2)
```

Regression

For this set of problems we will use the regression data set where we tried to predict the age of the sample from the methylation values. The data can be loaded as shown below:

```r
# file path for CpG methylation and age
fileMethAge=system.file("extdata",
                        "CpGmeth2Age.rds",
                        package="compGenomRData")
# read methylation-age table
ameth=readRDS(fileMethAge)
```

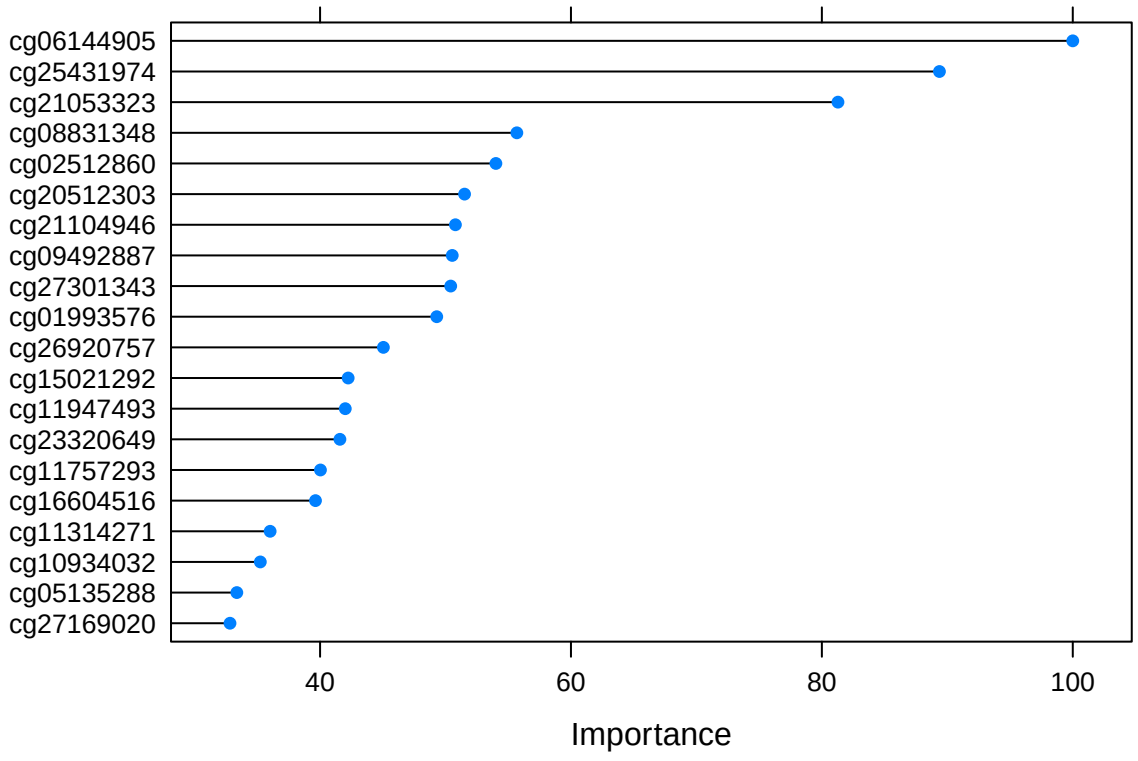
1. Run random forest regression and plot the importance metrics. [Difficulty: **Beginner**]

solution:

```r
#### Overflow issues - needed to remove variables #####

#There are 27000 predictor variables. We can remove the ones that have low variation across samples. In
#between  0 and 1. The CpGs that have low variation are not likely to have any association with age;
#they could simply be technical variation of the experiment. We will remove CpGs that have less than 0.

ameth = ameth[, c(TRUE, matrixStats::colSds(as.matrix(ameth[, -1])) > 0.1)]
dim(ameth)
```

```
## [1]  108 2290
```

```
set.seed(18)

par(mfrow=c(1,2))

# we are not going to do any cross-validation
# and rely on OOB error
trctrl <- trainControl(method = "none")

# we will now train random forest model
rfregFit <- train(Age~.,
                  data = ameth,
                  method = "ranger",
                  trControl=trctrl,
                  # calculate importance
                  importance="permutation",
                  tuneGrid = data.frame(mtry=50,
                                        min.node.size = 5,
                                        splitrule="variance")
                  )

# plot the importance metrics
plot(varImp(rfregFit), top = 20)
```



2. Split 20% of the methylation-age data as test data and run elastic net regression on the training portion to tune parameters and test it on the test portion. [Difficulty: **Intermediate**]

solution:

```
set.seed(3031) # set the random number seed for reproducibility

# get indices for 80% of the data set
intrain <- createDataPartition(y = ameth[,1], p= 0.8)[[1]]

# seperate test and training sets
training2 <- ameth[intrain,]
testing2 <- ameth[-intrain,]
```

```
set.seed(17)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.5
```

```
## Loading required package: Matrix
```

```
## Warning: package 'Matrix' was built under R version 4.0.5
```

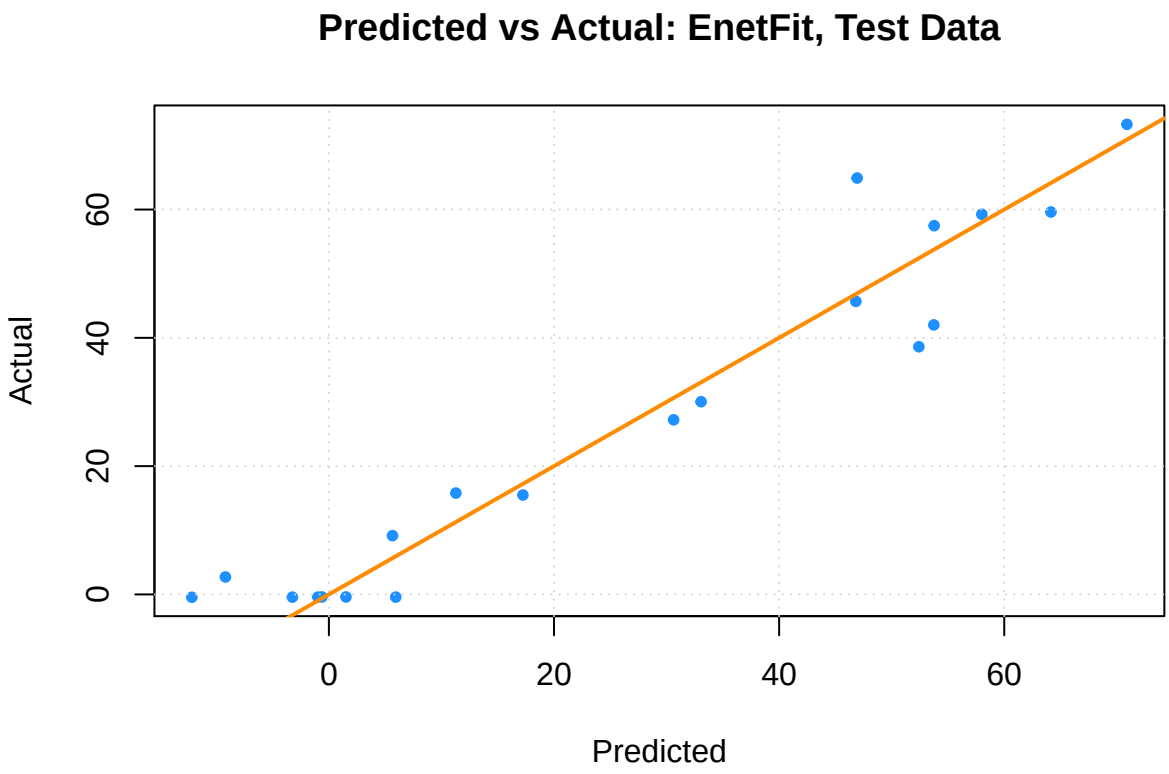
```
## Loaded glmnet 4.1-2
```

```
# this method controls everything about training
# we will just set up 10-fold cross validation
trctrl <- trainControl(method = "cv",number=10)

# we will now train elastic net model
# it will try
enetFit <- train(Age~., data = training2,
                 method = "glmnet",
                 trControl=trctrl,
                  # alpha and lambda paramters to try
                 tuneGrid = data.frame(alpha=0.5,
                                       lambda=seq(0.1,0.7,0.05)))


#enetFit
# best alpha and lambda values by cross-validation accuracy
enetFit$bestTune
```

```
##    alpha lambda
## 13   0.5    0.7
```

```
# predict on testing data
class.res=predict(enetFit,testing2[,-1])
```

```
plot(class.res, testing2[, 1],
     xlab = "Predicted", ylab = "Actual",
     main = "Predicted vs Actual: EnetFit, Test Data",
     col = "dodgerblue", pch = 20)
grid()
abline(0, 1, col = "darkorange", lwd = 2)
```




```
# summarize accuracy

rsq_enet <- cor(testing2[, 1], class.res)^2
rsq_enet
```

```
## [1] 0.9253791
```

```
mse <- mean((testing2[, 1] - class.res)^2)
mse
```

```
## [1] 54.12746
```

3. Run an ensemble model for regression using the **caretEnsemble** or **mlr** package and compare the results with the elastic net and random forest model. Did the test accuracy increase? **HINT:** You need to install these extra packages and learn how to use them in the context of ensemble models. [Difficulty: **Advanced**]

solution:

```
#install.packages("caretEnsemble")
library(caretEnsemble)
```

```
## Warning: package 'caretEnsemble' was built under R version 4.0.5
```

```
##
## Attaching package: 'caretEnsemble'

## The following object is masked from 'package:ggplot2':
##
##     autoplot
```

```
trctrl <- trainControl(method = "cv",
                       number=3,
                       returnResamp = "all",
                       savePredictions = "final",
                       classProbs = FALSE,
                       index = createMultiFolds(training2$Age, k = 3, times = 1))


set.seed(222)
#model_list <- caretList(training2[, -1],
#                        training2[, 1],
#                        trControl = trctrl,
#                        methodList = c('enet', 'rf'),
#                        tuneList = NULL,
#                        continue_on_fail = FALSE,
#                        preProcess = c('center','scale'))
#options(digits = 3)

#train_ctrl <- trainControl(method = "cv", number = 3, classProbs = FALSE, savePredictions = TRUE, indes

#model_results <- data.frame(
# RF = min(model_list$rf$results$RMSE),
# ENET = min(model_list$enet$results$RMSE)
# )
#print(model_results)

resamples <- resamples(model_list)
```

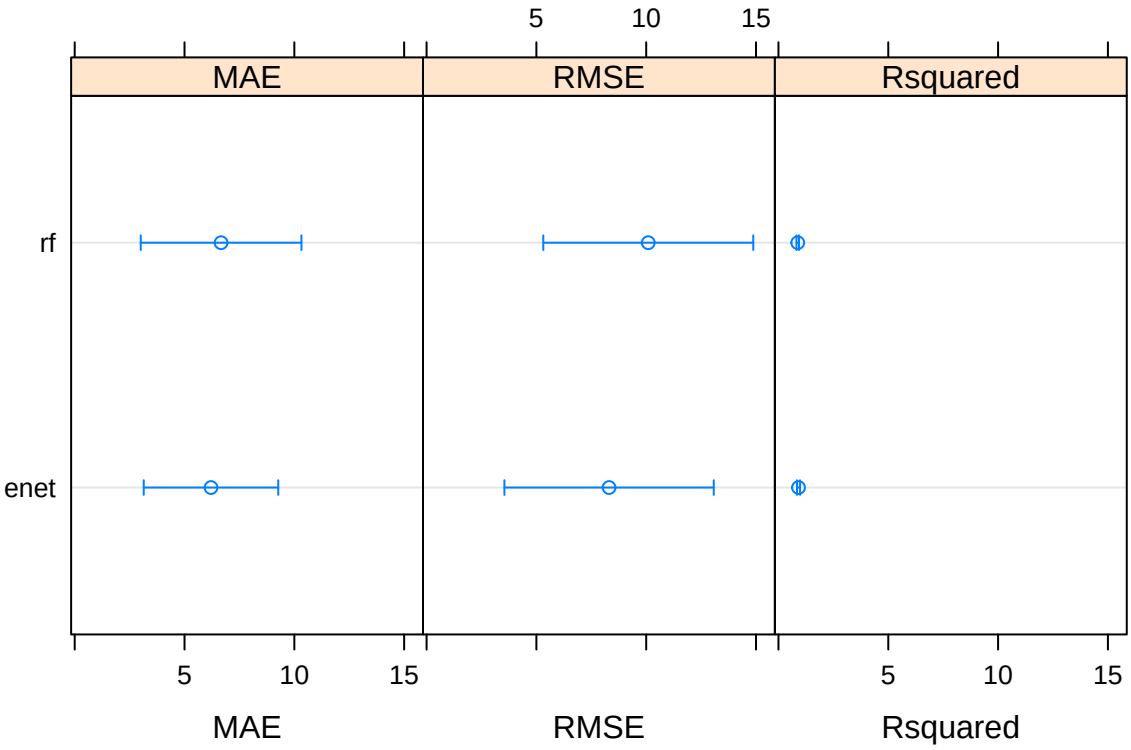
```
## Warning in resamples.default(model_list): 'enet' did not have
## 'returnResamp="final"; the optimal tuning parameters are used

## Warning in resamples.default(model_list): 'rf' did not have
## 'returnResamp="final"; the optimal tuning parameters are used
```

```
summary(resamples)
```

```
##
## Call:
## summary.resamples(object = resamples)
##
## Models: enet, rf
## Number of resamples: 3
##
## MAE
##          Min. 1st Qu.   Median     Mean  3rd Qu.     Max. NA's
```

```
## enet 4.838013 5.69450 6.550987 6.206215 6.890316 7.229646    0
## rf   5.184458 5.93317 6.681881 6.665470 7.405976 8.130071    0
##
## RMSE
##          Min.  1st Qu.   Median      Mean   3rd Qu.     Max. NA's
## enet 6.279104 7.417742 8.556381  8.309116  9.324122 10.09186    0
## rf   8.341965 9.065155 9.788345 10.094755 10.971150 12.15396    0
##
## Rsquared
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## enet 0.8922893 0.8981211 0.9039529 0.9137827 0.9245295 0.9451060    0
## rf   0.8631456 0.8650971 0.8670487 0.8783064 0.8858867 0.9047248    0
```

```
dotplot(resamples)
```



Confidence Level: 0.95

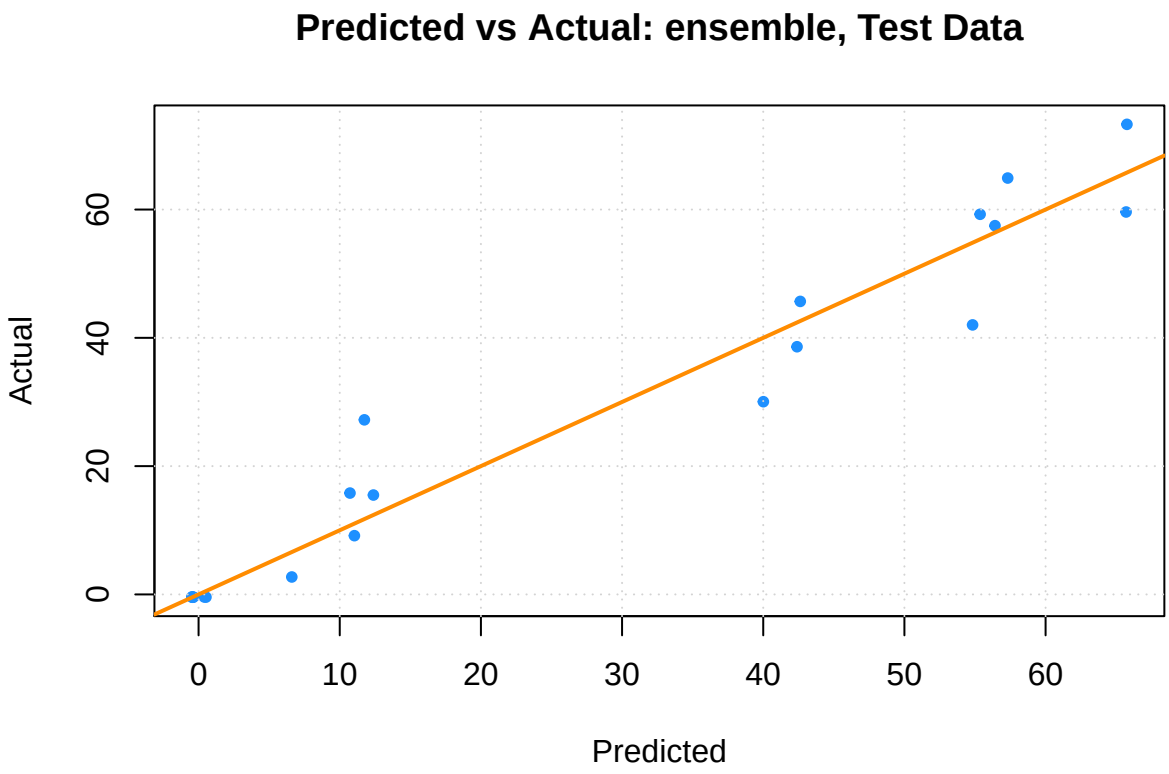
```
set.seed(222)
#ensemble_1 <- caretEnsemble(model_list,
#                             metric = 'RMSE',
#                             trControl = trctrl)

ensemble <- caretStack(model_list, method = "rf", metic = "Accuracy", trControl = trctrl)
```

```
## note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .
```

```
ensemble
```

```
## A rf ensemble of 2 base models: enet, rf
##
## Ensemble results:
## Random Forest
##
## 88 samples
##  2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 60, 57, 59
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   9.838201  0.9202594  6.388586
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

```
pred = predict(ensemble, testing2[,-1])

plot(pred, testing2[, 1],
     xlab = "Predicted", ylab = "Actual",
     main = "Predicted vs Actual: ensemble, Test Data",
     col = "dodgerblue", pch = 20)
grid()
abline(0, 1, col = "darkorange", lwd = 2)
```




```
# summarize accuracy

rsq_ensemble <- cor(testing2[, 1], pred)^2
rsq_ensemble
```

```
## [1] 0.9429972
```

```
mse_ensemble <- mean((testing2[, 1] - pred)^2)
mse_ensemble
```

```
## [1] 37.47664
```

```
sqrt(mse_ensemble)
```

```
## [1] 6.121817
```

Ensemble model may have better performance than individual models.