curatedMetagenomicData Data Application - Scenario 2

Load packages.

```r
library(curatedMetagenomicData)
library(plyr)
library(FSelector)
library(glmnet)
library(data.table)
library(nlme)
library(lme4)
```

Load data.

```r
metadata = curatedMetagenomicData(paste0(c("KarlssonFH_2013.",
                          "Heitz-BuschartA_2016.",
                          "QinJ_2012."), "marker_abundance.stool"), dryrun = FALSE)
meta.merged = mergeData(metadata)
meta.exp = data.frame(t(exprs(meta.merged)))
rm(list=c("metadata"))
```

Add clinical variables to marker abundance data and restrict to female patients.

```r
meta.exp = meta.exp[, which(colSums(meta.exp)!=0)]
# remove features that are very sparse in the training data
min.samples = floor(length(which(meta.merged$studyID=="KarlssonFH_2013.marker_abundance.stool" &
                                   meta.merged$gender=="female"))/10)
meta.exp = meta.exp[vapply(meta.exp, function(x)
  length(unique(x[which(meta.merged$studyID=="KarlssonFH_2013.marker_abundance.stool" &
                          meta.merged$gender=="female")]))>min.samples &
    length(unique(x[which(meta.merged$studyID=="QinJ_2012.marker_abundance.stool" &
                            meta.merged$gender=="female")]))>min.samples, logical(1L))]
meta.exp$cholesterol = meta.merged$cholesterol
meta.exp$age = meta.merged$age
meta.exp$group = as.numeric(as.factor(meta.merged$studyID))
meta.exp = meta.exp[which(meta.merged$gender=="female"), ]
meta.exp = meta.exp[complete.cases(meta.exp),]
```

Divide dataset into training and test sets.

```r
meta.train = meta.exp[which(meta.exp$group %in% 2:3),]
meta.test = meta.exp[which(meta.exp$group %in% 1),]
```

Use the top 20 marker abundances most highly correlated with the outcome in the training set as the predictors.

```
feature.list = lapply(split(as.list(as.data.frame(meta.train[, which(!names(meta.train) %in%
                                                                    c("cholesterol", "age",
                                                                      "group"))])),
                            cut(1:ncol(meta.train[, which(!names(meta.train) %in%
                                                         c("cholesterol", "age", "group"))]), 20)),
                      as.data.frame)
weight.list = lapply(feature.list, function(x) linear.correlation(meta.train$cholesterol ~., x))
weight.df = rbind.fill(weight.list)
weight.df$feature = unlist(lapply(weight.list, row.names))
weight.df = weight.df[order(weight.df$attr_importance, decreasing = T), ]
max.features = 20
meta.train = meta.train[, which(names(meta.train) %in% c(weight.df$feature[1:max.features],
                                                       c("cholesterol", "age", "group")))]
meta.test = meta.test[, which(names(meta.test) %in% c(weight.df$feature[1:max.features],
                                                    c("cholesterol", "age", "group")))]
```

Set up design matrices.

```
ndat = 2
meta.all = rbind(meta.train, meta.test)
parts = split(meta.all, meta.all$group)
edat_train = parts[2:3]
edat_test = parts[1]
train = rbindlist(edat_train)
test = rbindlist(edat_test)
```

Estimate random effect variances and variance of residuals using REML via a linear mixed effects model.

```
features = names(train)[which(!names(train) %in% c("cholesterol", "group"))]
lm.formula = as.formula(paste("cholesterol~", paste0(features, collapse="+")))
feature.cols = which(names(train) %in% c(features))

lmer.formula = as.formula(paste("cholesterol~ (1|group) +",
                               paste0(unique(names(train)[feature.cols]), collapse="+"), "+",
                               paste0("(0+", names(train)[feature.cols], "|group)", collapse="+")))
tol = 1e-10
fit.lmer = lmer(lmer.formula, data=train)
ind.re = which(as.data.frame(VarCorr(fit.lmer))[2:(length(feature.cols)+1), 4]>tol)
sigma.eps = summary(fit.lmer)$sigma
as.data.frame(VarCorr(fit.lmer))[1:(length(feature.cols)+1), 4]

##  [1] 2.970851e-03 7.385479e+03 0.000000e+00 0.000000e+00 7.558953e-04
##  [6] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
## [11] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 7.009743e-02
## [16] 0.000000e+00 8.634381e-04 0.000000e+00 0.000000e+00 4.019086e-04
## [21] 0.000000e+00 0.000000e+00

sigma2.bar = mean(as.data.frame(VarCorr(fit.lmer))[1:(length(feature.cols)+1), 4])
sigma2.bar

## [1] 335.707
```

Estimate optimal LS weights.

```r
source("transition_point_fns.R")
parts2 = split(cbind(rep(1, nrow(meta.all)), meta.all[, feature.cols]), meta.all$group)
edat_train2 = parts2[2:3]
edat_test2 = parts2[1]

# estimate optimal LS weights
vec.re = sqrt(as.data.frame(VarCorr(fit.lmer))[1:(length(feature.cols)+1), 4])
wk.ols = optimal_weights(edat_train2, edat_test2, sigma.eps, vec.re)
wk.ols

## [1] 0.4999645 0.5000355
```

Tune ridge regression regularization parameters.

```r
# choose regularization parameter
set.seed(1)
cv.ridge.merged = cv.glmnet(data.matrix(train)[, feature.cols], train$cholesterol,
                            alpha = 0,
                            intercept=T, lambda=2^seq(-8, 8, length=100), standardize=F)
sd.y = sqrt(var(train$cholesterol)*(length(train$cholesterol)-1)/length(train$cholesterol))
# compute regularization parameter for formulation of ridge regression objective function assumed by
lam = cv.ridge.merged$lambda.min*nrow(train)/sd.y

lamk = rep(NA, ndat)

for (i in 1:ndat) {
  dataset = edat_train[[i]]

  set.seed(1)
  cv.ridge = cv.glmnet(data.matrix(dataset)[, feature.cols], dataset$cholesterol,
                       alpha = 0,
                       intercept=T, lambda=2^seq(-8, 8, length=100),
                       standardize=F)
  sd.y = sqrt(var(dataset$cholesterol)*(length(dataset$cholesterol)-1)/length(dataset$cholesterol))
  lamk[i] = cv.ridge$lambda.min*nrow(dataset)/sd.y
}
```

Train and validate merged LS model.

```r
fit.ols.merged = lm(lm.formula, data=train)
pred.ols.merged = predict(fit.ols.merged, newdata=test)
err.ols.merged = mean((pred.ols.merged-test$cholesterol)^2)
sqrt(err.ols.merged)

## [1] 38.65778
```

Train and validate merged ridge regression model.

```r
fit.ridge.merged = glmnet(data.matrix(train)[, feature.cols], train$cholesterol,
                          alpha = 0,
                          lambda = cv.ridge.merged$lambda.min,
                          intercept=T, standardize=F)
pred.ridge.merged = predict(fit.ridge.merged, newx=data.matrix(test)[, feature.cols])
err.ridge.merged = mean((pred.ridge.merged - test$cholesterol)^2)
sqrt(err.ridge.merged)

## [1] 38.54562
```

Calculate transition intervals using optimal LS weights for the CSLs.

```
if (as.data.frame(VarCorr(fit.lmer))[1, 4] > tol) {
  ind.re = c(0, ind.re)
}

clist = as.list(ind.re+1)

ls.bounds = tau_ls_range(edat_train2, edat_test2, wk.ols, sigma.eps, cols_re_list=clist)
ls.bounds

## [1]   0.01786922 232.45457901

ridge.bounds = tau_r_range(edat_train2, edat_test2, wk.ols, sigma.eps, lambda=lam, lambdak=lamk,
                           beta=fit.ols.merged$coefficients, cols_re_list=clist)
ridge.bounds

## [1]  0.001405342 19.564695058

sigma2.bar > ls.bounds[2]

## [1] TRUE

sigma2.bar > ridge.bounds[2]

## [1] TRUE
```

Train and validate CSLs.

```
beta.ols.mat = matrix(data=NA, nrow=ndat, ncol=length(fit.ols.merged$coefficients))
beta.ridge.mat = beta.ols.mat
ols.mat = matrix(data=NA, nrow=ndat, ncol=nrow(test))
ridge.mat = ols.mat

# fit models to each study
for (i in 1:ndat) {

  dataset = edat_train[[i]]

  # OLS
  fit.ols = lm(lm.formula, data=dataset)
  beta.ols.mat[i, ] = fit.ols$coefficients
  ols.mat[i, ] = predict(fit.ols, newdata=test)

  # ridge
  sd.y = sqrt(var(dataset$cholesterol)*(length(dataset$cholesterol)-1)/length(dataset$cholesterol))
  fit.ridge = glmnet(data.matrix(dataset)[, feature.cols], dataset$cholesterol,
                     alpha = 0,
                     lambda = lamk[i]*sd.y/nrow(dataset),
                     intercept=T, standardize=F)
  beta.ridge.mat[i, ] = c(fit.ridge$a0, as.vector(fit.ridge$beta))
  ridge.mat[i, ] = predict(fit.ridge, newx=data.matrix(test)[, feature.cols])
}

pred.ols.csl = wk.ols %*% ols.mat
err.ols.csl = mean((pred.ols.csl-test$cholesterol)^2)
sqrt(err.ols.csl)

## [1] 34.35825

pred.ridge.csl = wk.ols %*% ridge.mat
err.ridge.csl = mean((pred.ridge.csl-test$cholesterol)^2)
sqrt(err.ridge.csl)

## [1] 35.21478
```

Get bootstrap confidence intervals for prediction error.

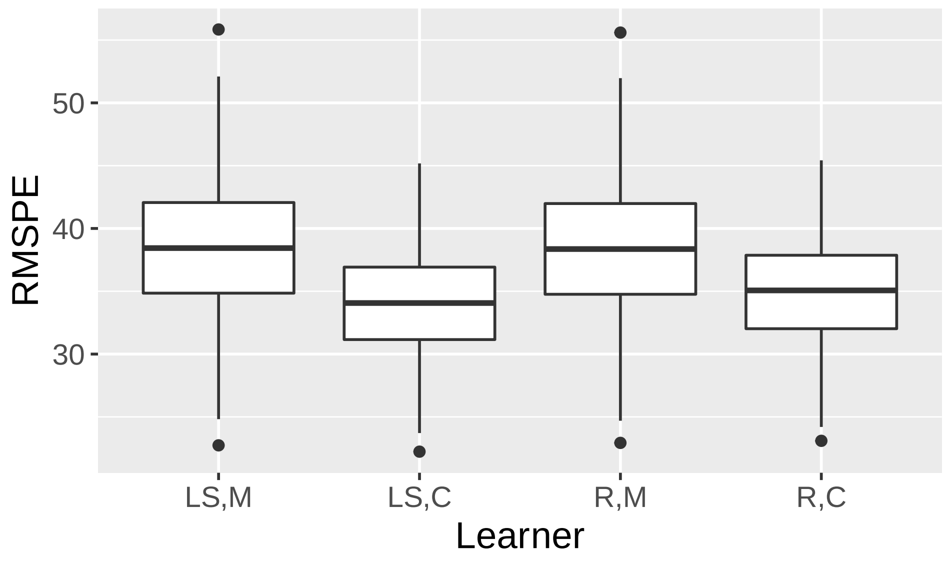
```
nboot = 500

set.seed(1)
err.ridge = data.frame(merged=rep(NA, nboot), csl=NA)
err.ols = data.frame(merged=rep(NA, nboot), csl=NA)
for (i in 1:nboot) {
  ind.boot = sample(1:length(pred.ridge.merged), length(pred.ridge.merged), replace=T)
  err.ridge$merged[i] = mean((pred.ridge.merged[ind.boot] - test$cholesterol[ind.boot])^2)
  err.ridge$csl[i] = mean((pred.ridge.csl[ind.boot] - test$cholesterol[ind.boot])^2)
  err.ols$merged[i] = mean((pred.ols.merged[ind.boot] - test$cholesterol[ind.boot])^2)
  err.ols$csl[i] = mean((pred.ols.csl[ind.boot] - test$cholesterol[ind.boot])^2)
}
save(err.ridge, err.ols, file="cholesterol_multi.RData")
```

Make boxplots of prediction error.

```
library(ggplot2)
library(reshape2)

err.ridge = sqrt(err.ridge)
err.ols = sqrt(err.ols)
names(err.ridge) = c("R,M", "R,C")
names(err.ols) = c("LS,M", "LS,C")
err2 = melt(cbind(err.ols, err.ridge))
names(err2) = c("Learner", "RMSPE")
ggplot(err2, aes(Learner, RMSPE)) + geom_boxplot() + theme(text = element_text(size = 14))
```



```
# save figure
library(gridExtra)
png('rmse_scenario2.png', width=800, height=500, res=100)
ggplot(err2, aes(Learner, RMSPE)) + geom_boxplot() + theme(text = element_text(size = 14))
dev.off()

## pdf
##   2
```