C. code

```r
## Non-linear SVM -------------------------------
library(e1071)

### math regression -----------------
svm.math.reg = svm.cv.mine(math.train, regression.formula, 10,
    "grade.con")
svm.math.reg
# $kernel [1] 'radial' $gamma [1] 0.01 $cost [1] 0.5

svm.math.reg.fit = do.call(svm, c(list(formula = regression.formula,
    data = math.train), svm.math.reg))
predict.test = predict(svm.math.reg.fit, math.test, decision.values = F)

svm.math.reg.error.test = mean((predict.test - math.test$grade.con)^2)
svm.math.reg.error.train = mean((predict(svm.math.reg.fit) -
    math.train$grade.con)^2)
svm.math.reg.error.test

svm.math.reg.var.imp = numeric(length(predictors))
for (i in 1:length(predictors)) {
    svm.math.reg.fit.inf = do.call(svm, c(list(formula = as.formula(paste("grade.con~",
        paste(predictors[-i], collapse = "+"), sep = "")), data = math.train),
        svm.math.reg))

    svm.math.reg.var.imp[i] = mean((predict(svm.math.reg.fit.inf) -
        math.train$grade.con)^2) - svm.math.reg.error.train
}

svm.math.reg.var.imp = data.frame(predictors = predictors, importance = svm.math.reg.var.imp)
svm.math.reg.var.imp = svm.math.reg.var.imp[order(svm.math.reg.var.imp$importance,
    decreasing = T), ]
xtable(svm.math.reg.var.imp)


### por regression -------------------------
svm.por.reg = svm.cv.mine(por.train, regression.formula, 10,
    "grade.con")
svm.por.reg
# $kernel [1] 'radial' $gamma [1] 0.005959184 $cost [1] 0.8

svm.por.reg.fit = do.call(svm, c(list(formula = regression.formula,
    data = por.train), svm.por.reg))
predict.test = predict(svm.por.reg.fit, por.test, decision.values = F)

svm.por.reg.error.test = mean((predict.test - por.test$grade.con)^2)
svm.por.reg.error.test
```

```r
svm.por.reg.error.train = mean((predict(svm.por.reg.fit) - por.train$grade.con)^2)

svm.por.reg.var.imp = numeric(length(predictors))
for (i in 1:length(predictors)) {
    svm.por.reg.fit.inf = do.call(svm, c(list(formula = as.formula(paste("grade.con~",
        paste(predictors[-i], collapse = "+"), sep = "")), data = por.train),
        svm.por.reg))

    svm.por.reg.var.imp[i] = mean((predict(svm.por.reg.fit.inf) -
        por.train$grade.con)^2) - svm.por.reg.error.train
}

svm.por.reg.var.imp = data.frame(predictors = predictors, importance = svm.por.reg.var.imp)
svm.por.reg.var.imp = svm.por.reg.var.imp[order(svm.por.reg.var.imp$importance,
    decreasing = T), ]
xtable(svm.por.reg.var.imp[1:10, ])

### por class ------------------------------
svm.por.cat = svm.cv.mine.cat(por.train, classification.formula,
    10, "grade.cat")
svm.por.cat
# $kernel [1] 'radial' $gamma [1] 0.002120408 $cost [1] 3.4

svm.por.cat.fit = do.call(svm, c(list(formula = classification.formula,
    data = por.train), svm.por.cat))
predict.test = predict(svm.por.cat.fit, por.test, decision.values = F)

svm.por.cat.error.test = mean((predict.test != por.test$grade.cat))
svm.por.cat.error.train = mean(predict(svm.por.cat.fit) != por.train$grade.cat)
svm.por.cat.error.test

### math class ----------------
svm.math.cat = svm.cv.mine.cat(math.train, classification.formula,
    10, "grade.cat")
svm.math.cat
# $kernel [1] 'radial' $gamma [1] 0.00252449 $cost [1] 5

svm.math.cat.fit = do.call(svm, c(list(formula = classification.formula,
    data = math.train), svm.math.cat))
predict.test = predict(svm.math.cat.fit, math.test, decision.values = F)

svm.math.cat.error.test = mean((predict.test != math.test$grade.cat))
svm.math.cat.error.train = mean(predict(svm.math.cat.fit) !=
    math.train$grade.cat)
svm.math.cat.error.test

svm.math.cat.var.imp = numeric(length(predictors))
for (i in 1:length(predictors)) {
    svm.math.cat.fit.inf = do.call(svm, c(list(formula = as.formula(paste("grade.cat~",
        paste(predictors[-i], collapse = "+"), sep = "")), data = math.train),
        svm.math.cat))

    svm.math.cat.var.imp[i] = mean((predict(svm.math.cat.fit.inf) !=
```

```r
        math.train$grade.cat)) - svm.math.cat.error.train
}

svm.math.cat.var.imp = data.frame(predictors = predictors, importance = svm.math.cat.var.imp)
svm.math.cat.var.imp = svm.math.cat.var.imp[order(svm.math.cat.var.imp$importance,
    decreasing = T), ]
xtable(svm.math.cat.var.imp[1:10, ])

svm.por.cat.var.imp = numeric(length(predictors))
for (i in 1:length(predictors)) {
    svm.por.cat.fit.inf = do.call(svm, c(list(formula = as.formula(paste("grade.cat~",
        paste(predictors[-i], collapse = "+"), sep = "")), data = por.train),
        svm.por.cat))

    svm.por.cat.var.imp[i] = mean((predict(svm.por.cat.fit.inf) !=
        por.train$grade.cat)) - svm.por.cat.error.train
}

svm.por.cat.var.imp = data.frame(predictors = predictors, importance = svm.por.cat.var.imp)
svm.por.cat.var.imp = svm.por.cat.var.imp[order(svm.por.cat.var.imp$importance,
    decreasing = T), ]
xtable(svm.por.cat.var.imp[1:10, ])

intersect(svm.por.cat.var.imp[1:10, 1], svm.math.cat.var.imp[1:10,
    1])


## KNN ----------------------------

## Neural network -------------------
library(keras)
library(dplyr)

nn.model.math.reg = keras_model_sequential()


nunits = c(10, 5)
act_fun = c("relu", "elu")

nn.model.math.reg %>%
    layer_dense(units = nunits[1], activation = act_fun[1], input_shape = c(39)) %>%
    layer_dense(units = 1)

nn.model.math.reg %>%
    compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

nn.model.math.reg.fit <- nn.model.math.reg %>%
    fit(as.matrix(math.train[, predictors]), math.train$grade.con,
        epochs = 20, batch_size = 8, validation_split = 0.2)
mean((nn.model.math.reg %>%
    predict(as.matrix(math.test[, predictors])) - math.test$grade.con)^2)

nn.model.math.reg %>%
```

```r
    evaluate(as.matrix(math.test[, predictors]), math.test$grade.con)


paras_list = expand.grid(c(5, 10, 20), c(2, 4, 5, 10), c("relu",
    "elu", "selu", "sigmoid"), c("relu", "elu", "selu", "sigmoid"),
    c(8, 16, 32), c(10, 20, 40, 60, 80))
# set up a validation set to select parameters
val_d.idx = sample(nrow(math.train), 100, replace = F)
val_d = math.train[val_d.idx, ]
train_d = math.train[-val_d.idx, ]

val_error = numeric(nrow(paras_list))

for (i in 1:nrow(paras_list)) {
    nn.model.math.reg = keras_model_sequential()
    nn.model.math.reg %>%
        layer_dense(units = paras_list[i, 1], activation = paras_list[i,
            3], input_shape = c(39)) %>%
        layer_dense(units = paras_list[i, 2], activation = paras_list[i,
            4]) %>%
        layer_dense(units = 1)

    nn.model.math.reg %>%
        compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

    nn.model.math.reg.fit <- nn.model.math.reg %>%
        fit(as.matrix(train_d[, predictors]), train_d$grade.con,
            epochs = paras_list[i, 6], batch_size = paras_list[i,
                5], )

    val_error[i] = mean((nn.model.math.reg %>%
        predict(as.matrix(val_d[, predictors])) - val_d$grade.con)^2)
}


paras_list[which.min(val_error), ]
# Var1 Var2 Var3 Var4 Var5 Var6 580 5 4 relu relu 8 20

idx = 179
nn.model.math.reg = keras_model_sequential()
nn.model.math.reg %>%
    layer_dense(units = paras_list[idx, 1], activation = paras_list[idx,
        3], input_shape = c(39)) %>%
    layer_dense(units = paras_list[idx, 2], activation = paras_list[idx,
        4]) %>%
    layer_dense(units = 1)

nn.model.math.reg %>%
    compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

nn.model.math.reg.fit <- nn.model.math.reg %>%
    fit(as.matrix(math.train[, predictors]), math.train$grade.con,
        epochs = paras_list[idx, 6], batch_size = paras_list[idx,
```

```
           5], validation_split = 0.2)

mean((nn.model.math.reg %>%
    predict(as.matrix(math.test[, predictors])) - math.test$grade.con)^2)

nn.model.math.reg = keras_model_sequential()
nn.model.math.reg %>%
    layer_dense(units = 5, activation = "relu", input_shape = c(39)) %>%
    layer_dense(units = 4, activation = "relu") %>%
    layer_dense(units = 1)

nn.model.math.reg %>%
    compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

nn.model.math.reg.fit <- nn.model.math.reg %>%
    fit(as.matrix(math.train[, predictors]), math.train$grade.con,
        epochs = 20, batch_size = 8, validation_split = 0.2)

mean((nn.model.math.reg %>%
    predict(as.matrix(math.test[, predictors])) - math.test$grade.con)^2)
mean((nn.model.math.reg %>%
    predict(as.matrix(math.train[, predictors])) - math.train$grade.con)^2)

library(keras)
library(dplyr)




nn.model.por.reg = keras_model_sequential()


nunits = c(10, 5)
act_fun = c("relu", "elu")

nn.model.por.reg %>%
    layer_dense(units = nunits[1], activation = act_fun[1], input_shape = c(39)) %>%
    layer_dense(units = 1)

nn.model.por.reg %>%
    compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

nn.model.por.reg.fit <- nn.model.por.reg %>%
    fit(as.matrix(por.train[, predictors]), por.train$grade.con,
        epochs = 20, batch_size = 8, validation_split = 0.2)
mean((nn.model.por.reg %>%
    predict(as.matrix(por.test[, predictors])) - por.test$grade.con)^2)

nn.model.por.reg %>%
    evaluate(as.matrix(por.test[, predictors]), por.test$grade.con)


paras_list = expand.grid(c(5, 10, 20), c(2, 4, 5, 10), c("relu",
```

```r
    "elu", "selu", "sigmoid"), c("relu", "elu", "selu", "sigmoid"),
    c(8, 16, 32), c(10, 20, 40, 60, 80))
# set up a validation set to select parameters
set.seed(1)
val_d.idx = sample(nrow(por.train), 100, replace = F)
val_d = por.train[val_d.idx, ]
train_d = por.train[-val_d.idx, ]

val_error = numeric(nrow(paras_list))


options(keras.view_metrics = FALSE)

set.seed(1)
for (i in 1:nrow(paras_list)) {
    nn.model.math.reg = keras_model_sequential()
    nn.model.math.reg %>%
        layer_dense(units = paras_list[i, 1], activation = paras_list[i,
            3], input_shape = c(39)) %>%
        layer_dense(units = paras_list[i, 2], activation = paras_list[i,
            4]) %>%
        layer_dense(units = 1)

    nn.model.math.reg %>%
        compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

    nn.model.math.reg.fit <- nn.model.math.reg %>%
        fit(as.matrix(train_d[, predictors]), train_d$grade.con,
            epochs = paras_list[i, 6], batch_size = paras_list[i,
                5], )

    val_error[i] = mean((nn.model.math.reg %>%
        predict(as.matrix(val_d[, predictors])) - val_d$grade.con)^2)
}


paras_list[which.min(val_error), ]


paras_list[which.min(val_error), ]
# Var1 Var2 Var3 Var4 Var5 Var6 2100 20 10 selu sigmoid 16
# 60

idx = 2100
nn.model.por.reg = keras_model_sequential()
nn.model.por.reg %>%
    layer_dense(units = paras_list[idx, 1], activation = paras_list[idx,
        3], input_shape = c(39)) %>%
    layer_dense(units = paras_list[idx, 2], activation = paras_list[idx,
        4]) %>%
    layer_dense(units = 1)

nn.model.por.reg %>%
```

```
    compile(loss = "mse", optimizer = "rmsprop", metrics = "mse")

nn.model.por.reg.fit <- nn.model.por.reg %>%
    fit(as.matrix(por.train[, predictors]), por.train$grade.con,
        epochs = paras_list[idx, 6], batch_size = paras_list[idx,
            5], validation_split = 0.2)

mean((nn.model.por.reg %>%
    predict(as.matrix(por.test[, predictors])) - por.test$grade.con)^2)
mean((nn.model.por.reg %>%
    predict(as.matrix(por.train[, predictors])) - por.train$grade.con)^2)

## Tree based ---------------------------
```

Tree-based Regression————

```
library(rpart.plot)
library(rattle)
library(rpart)
library(caret)
library(randomForest)
library(gbm)
## Decision Tree for math--------- Unpruned Tree----------
tree.math <- rpart(regression.formula, data = math.train, control = rpart.control(cp = 0))
# predicted result for tree
train.math <- predict(tree.math, newdata = math.train)
tree.pred.math <- predict(tree.math, newdata = math.test)
tree.pred.math
# MSE for unpruned
mean((train.math - math.train$grade.con)^2)
mean((tree.pred.math - math.test$grade.con)^2)
### Pruned Tree-------
set.seed(114514)
tree.tune.math <- train(regression.formula, data = math.train,
    method = "rpart", preProcess = c("center", "scale"), tuneGrid = data.frame(.cp = seq(0,
        0.2, by = 0.004)), trControl = trainControl(method = "repeatedcv",
        repeats = 5, number = 10))
plot(tree.tune.math)
# best cost parameter
best <- tree.tune.math$bestTune
best
tree.math.pruned <- rpart(regression.formula, data = math.train,
    control = rpart.control(cp = best))
fancyRpartPlot(tree.math.pruned, caption = NULL)
# predicted result for pruned tree
train.math <- predict(tree.math.pruned, newdata = math.train)
tree.pruned.math.pred <- predict(tree.math.pruned, newdata = math.test)
tree.pruned.math.pred
# MSE for pruned
mean((train.math - math.train$grade.con)^2)
mean((tree.pruned.math.pred - math.test$grade.con)^2)
```

```r
## Decision Tree for Por---------
tree.por <- rpart(regression.formula, data = por.train, control = rpart.control(cp = 0))
# predicted result for tree
train.por <- predict(tree.por, newdata = por.train)
tree.pred.por <- predict(tree.por, newdata = por.test)
tree.pred.por

# MSE for unpruned
mean((train.por - por.train$grade.con)^2)
mean((tree.pred.por - por.test$grade.con)^2)
### Pruned Tree-------
set.seed(114514)
tree.tune.por <- train(regression.formula, data = por.train,
    method = "rpart", preProcess = c("center", "scale"), tuneGrid = data.frame(.cp = seq(0,
        0.2, by = 0.004)), trControl = trainControl(method = "repeatedcv",
        repeats = 5, number = 10))
plot(tree.tune.por)
# best cost parameter
best <- tree.tune.por$bestTune
best
tree.por.pruned <- rpart(regression.formula, data = por.train,
    control = rpart.control(cp = best))
fancyRpartPlot(tree.por.pruned, caption = NULL)
# predicted result for pruned tree
train.por <- predict(tree.por.pruned, newdata = por.train)
tree.pruned.por.pred <- predict(tree.por.pruned, newdata = ,
    por.test)
tree.pruned.por.pred
# MSE
mean((train.por - por.train$grade.con)^2)
mean((tree.pruned.por.pred - por.test$grade.con)^2)
```

Random Forrest regression

```r
library(randomForest)
library(caret)
### math-------- choice of m
37/3
# cv on number of trees
test.error <- rep(1, 10)
set.seed(114514)
folds <- createFolds(1:nrow(math.train), k = 10, returnTrain = FALSE)
tree.vec <- seq(50, 1000, by = 50)
tree.error <- rep(1, length(tree.vec))
for (k in 1:length(tree.vec)) {
    for (i in 1:10) {
        set.seed(114514)
        train.rand.for.math <- randomForest(regression.formula,
            data = math.train[-folds[[i]], ], mtry = 12, ntree = k)
        train.pred.rand.for.math <- predict(train.rand.for.math,
            newdata = math.train[folds[[i]], ])
```

```r
        test.error[i] <- mean((train.pred.rand.for.math - math.train[folds[[i]],
            ]$grade.con)^2)
    }
    tree.error[k] <- mean(test.error)
}
# the choice of number of trees
tree.vec[which.min(tree.error)]
best.rd.math <- tree.vec[which.min(tree.error)]
set.seed(114514)
train.rand.for.math <- randomForest(regression.formula, data = math.train,
    mtry = 14, ntree = best.rd.math)
train.rand.for.math
test.pred.rand.for.math <- predict(train.rand.for.math, newdata = math.test)
train.pred.rand.for.math <- predict(train.rand.for.math, newdata = math.train)
# test result
test.pred.rand.for.math
# misclassifictaion rate
mean((train.pred.rand.for.math - math.train$grade.con)^2)
mean((test.pred.rand.for.math - math.test$grade.con)^2)
library(vip)
vip::vip(train.rand.for.math)


### por--------- choice of m
37/3
# cv on number of trees
test.error <- rep(1, 10)
set.seed(114514)
folds <- createFolds(1:nrow(por.train), k = 10, returnTrain = FALSE)
tree.vec <- seq(50, 1000, by = 50)
tree.error <- rep(1, length(tree.vec))
for (k in 1:length(tree.vec)) {
    for (i in 1:10) {
        set.seed(114514)
        train.rand.for.por <- randomForest(regression.formula,
            data = por.train[-folds[[i]], ], mtry = 12, ntree = k)
        train.pred.rand.for.por <- predict(train.rand.for.por,
            newdata = por.train[folds[[i]], ])
        test.error[i] <- mean((train.pred.rand.for.por - por.train[folds[[i]],
            ]$grade.con)^2)
    }
    tree.error[k] <- mean(test.error)
}
# the choice of number of trees
tree.vec[which.min(tree.error)]
best.rd.por <- tree.vec[which.min(tree.error)]
set.seed(114514)
train.rand.for.por <- randomForest(regression.formula, data = por.train,
    mtry = 14, ntree = best.rd.por)
train.rand.for.por
test.pred.rand.for.por <- predict(train.rand.for.por, newdata = por.test)
train.pred.rand.for.por <- predict(train.rand.for.por, newdata = por.train)
# test result
```

```r
test.pred.rand.for.por
# misclassifictaion rate
mean((train.pred.rand.for.por - por.train$grade.con)^2)
mean((test.pred.rand.for.por - por.test$grade.con)^2)
library(vip)
vip::vip(train.rand.for.por)
```

TreeBased Classification

```r
library(rpart.plot)
library(rattle)
library(rpart)
library(caret)
library(gbm)
## Decision Tree for math--------- Unpruned Tree----------
tree.math <- rpart(classification.formula, data = math.train,
    control = rpart.control(cp = 0))
# predicted result for tree
tree.pred.math <- predict(tree.math, newdata = math.test, type = "class")
tree.pred.math.train <- predict(tree.math, newdata = math.train,
    type = "class")
tree.pred.math
# misclassification for unpruned
1 - mean(tree.pred.math.train == math.train$grade.cat)
1 - mean(tree.pred.math == math.test$grade.cat)
### Pruned Tree-------
set.seed(114514)
tree.tune.math <- train(classification.formula, data = math.train,
    method = "rpart", preProcess = c("center", "scale"), tuneGrid = data.frame(cp = seq(0,
        0.05, by = 0.001)), trControl = trainControl(method = "repeatedcv",
        repeats = 5, number = 10))
plot(tree.tune.math)
# best cost parameter
best <- tree.tune.math$bestTune
best
tree.math.pruned <- rpart(classification.formula, data = math.train,
    control = rpart.control(cp = best))
## We can't plot this because it's only a root--------
## predicted result for pruned tree
tree.pruned.math.pred <- predict(tree.math.pruned, newdata = math.test,
    type = "class")
tree.pruned.math.train <- predict(tree.math.pruned, newdata = math.train,
    type = "class")
tree.pruned.math.pred
# misclassification for pruned
1 - mean(tree.pruned.math.train == math.train$grade.cat)
1 - mean(tree.pruned.math.pred == math.test$grade.cat)



## Decision Tree for Por---------
tree.por <- rpart(classification.formula, data = por.train, control = rpart.control(cp = 0))
# predicted result for tree
```

```r
tree.pred.por <- predict(tree.por, newdata = por.test, type = "class")
tree.pred.por.train <- predict(tree.por, newdata = por.train,
    type = "class")
tree.pred.por
# misclassification rate for unpruned
1 - mean(tree.pred.por.train == por.train$grade.cat)
1 - mean(tree.pred.por == por.test$grade.cat)
### Pruned Tree-------
set.seed(114514)
tree.tune.por <- train(classification.formula, data = por.train,
    method = "rpart", preProcess = c("center", "scale"), tuneGrid = data.frame(cp = seq(0,
        0.2, by = 0.004)), trControl = trainControl(method = "repeatedcv",
        repeats = 5, number = 10))
plot(tree.tune.por)
# best cost parameter
best <- tree.tune.por$bestTune
best
tree.por.pruned <- rpart(classification.formula, data = por.train,
    control = rpart.control(cp = best))
fancyRpartPlot(tree.por.pruned, caption = NULL)
# predicted result for pruned tree
tree.pruned.por.pred <- predict(tree.por.pruned, newdata = por.test,
    type = "class")
tree.pruned.por.train <- predict(tree.por.pruned, newdata = por.train,
    type = "class")
tree.pruned.por.pred
# misclassification rate for pruned
1 - mean(tree.pruned.por.train == por.train$grade.cat)
1 - mean(tree.pruned.por.pred == por.test$grade.cat)
```

Random Forrest Classification

```r
library(randomForest)
### math-------- choice of m
sqrt(37)
# cv on number of trees
test.error <- rep(1, 10)
set.seed(114514)
folds <- createFolds(1:nrow(math.train), k = 10, returnTrain = FALSE)
tree.vec <- seq(50, 1000, by = 50)
tree.error <- rep(1, length(tree.vec))
for (k in 1:length(tree.vec)) {
    for (i in 1:10) {
        set.seed(114514)
        train.rand.for.math <- randomForest(classification.formula,
            data = math.train[-folds[[i]], ], mtry = 6, ntree = k)
        train.pred.rand.for.math <- predict(train.rand.for.math,
            newdata = math.train[folds[[i]], ], type = "class")
        test.error[i] <- 1 - mean(train.pred.rand.for.math ==
            math.train[folds[[i]], ]$grade.cat)
    }
    tree.error[k] <- mean(test.error)
}
```

```
# the choice of number of trees
tree.vec[which.min(tree.error)]
best.rd.math <- tree.vec[which.min(tree.error)]
set.seed(114514)
train.rand.for.math <- randomForest(classification.formula, data = math.train,
    mtry = 6, ntree = best.rd.math)
train.rand.for.math
test.pred.rand.for.math <- predict(train.rand.for.math, math.test,
    type = "class")
train.pred.rand.for.math <- predict(train.rand.for.math, math.train,
    type = "class")
# test result
test.pred.rand.for.math
# misclassifictaion rate
1 - mean(train.pred.rand.for.math == math.train$grade.cat)
1 - mean(test.pred.rand.for.math == math.test$grade.cat)
library(vip)
vip::vip(train.rand.for.math)
### por--------- por-------- choice of m
sqrt(37)
# cv on number of trees
test.error <- rep(1, 10)
set.seed(114514)
folds <- createFolds(1:nrow(por.train), k = 10, returnTrain = FALSE)
tree.vec <- seq(50, 1000, by = 50)
tree.error <- rep(1, length(tree.vec))
for (k in 1:length(tree.vec)) {
    for (i in 1:10) {
        set.seed(114514)
        train.rand.for.por <- randomForest(classification.formula,
            data = por.train[-folds[[i]], ], mtry = 6, ntree = k)
        train.pred.rand.for.por <- predict(train.rand.for.por,
            newdata = por.train[folds[[i]], ], type = "class")
        test.error[i] <- 1 - mean(train.pred.rand.for.por ==
            por.train[folds[[i]], ]$grade.cat)
    }
    tree.error[k] <- mean(test.error)
}
# the choice of number of trees
tree.vec[which.min(tree.error)]
best.rd.por <- tree.vec[which.min(tree.error)]
set.seed(114514)
train.rand.for.por <- randomForest(classification.formula, data = por.train,
    mtry = 6, ntree = best.rd.por)
train.rand.for.por
test.pred.rand.for.por <- predict(train.rand.for.por, por.test,
    type = "class")
train.pred.rand.for.por <- predict(train.rand.for.por, por.train,
    type = "class")
# test result
test.pred.rand.for.por
# misclassifictaion rate
1 - mean(train.pred.rand.for.por == por.train$grade.cat)
```

```
1 - mean(test.pred.rand.for.por == por.test$grade.cat)
library(vip)
vip::vip(train.rand.for.por)
```

KNN Regression

```
library(plotrix)
library(FNN)
library(e1071)
## Math-----------------

ctrl <- trainControl(method = "cv", number = 10)

knn.model.math <- train(regression.formula, data = math.train,
    preProcess = c("center", "scale"), method = "knn", trControl = ctrl,
    tuneLength = 20)
plot(knn.model.math)

bestK.math <- knn.model.math$bestTune$k
bestK.math
Knn.for.math <- knn.reg(train = math.train[, -c(14, 15, 16, 43,
    44)], test = math.test[, -c(14, 15, 16, 43, 44)], y = math.train$grade.con,
    k = bestK.math)

# MSE
mean((math.test$grade.con - Knn.for.math$pred)^2)

Knn.for.math <- knn.reg(train = math.train[, -c(14, 15, 16, 43,
    44)], test = math.train[, -c(14, 15, 16, 43, 44)], y = math.train$grade.con,
    k = bestK.math)

# MSE
mean((math.train$grade.con - Knn.for.math$pred)^2)
## Por--------------- For Portuguese

knn.model.por <- train(regression.formula, data = por.train,
    preProcess = c("center", "scale"), method = "knn", trControl = ctrl,
    tuneLength = 20)
plot(knn.model.por)
bestK.por <- knn.model.por$bestTune$k
bestK.por

Knn.for.por <- knn.reg(train = por.train[, -c(14, 15, 16, 43,
    44)], test = por.test[, -c(14, 15, 16, 43, 44)], y = por.train$grade.con,
    k = bestK.por)

# MSE
mean((por.test$grade.con - Knn.for.por$pred)^2)

Knn.for.por <- knn.reg(train = por.train[, -c(14, 15, 16, 43,
    44)], test = por.train[, -c(14, 15, 16, 43, 44)], y = por.train$grade.con,
```

```r
    k = bestK.por)

# MSE
mean((por.train$grade.con - Knn.for.por$pred)^2)
```

KNN Classification

```r
## math----------
library(class)
set.seed(114514)
ctrl <- trainControl(method = "cv", number = 10)

knn.model.math <- train(classification.formula, data = math.train,
    preProcess = c("center", "scale"), method = "knn", trControl = ctrl,
    tuneLength = 20)
plot(knn.model.math)
bestK.math <- knn.model.math$bestTune$k
bestK.math
math.train$grade.cat <- as.numeric(math.train$grade.cat)
math.test$grade.cat <- as.numeric(math.test$grade.cat)
Knn.for.math <- knn(train = math.train[, -c(14, 15, 16, 43, 44)],
    test = math.test[, -c(14, 15, 16, 43, 44)], cl = math.train$grade.cat,
    k = bestK.math)
# Misclassification Error
1 - mean(math.test$grade.cat == Knn.for.math)
Knn.for.math <- knn(train = math.train[, -c(14, 15, 16, 43, 44)],
    test = math.train[, -c(14, 15, 16, 43, 44)], cl = math.train$grade.cat,
    k = bestK.math)
# Misclassification Error
1 - mean(math.train$grade.cat == Knn.for.math)
## por----------
set.seed(114514)
ctrl <- trainControl(method = "cv", number = 10)

knn.model.por <- train(classification.formula, data = por.train,
    preProcess = c("center", "scale"), method = "knn", trControl = ctrl,
    tuneLength = 20)
plot(knn.model.por)
bestK.por <- knn.model.por$bestTune$k
bestK.por
por.train$grade.cat <- as.numeric(por.train$grade.cat)
por.test$grade.cat <- as.numeric(por.test$grade.cat)
Knn.for.por <- knn(train = por.train[, -c(14, 15, 16, 43, 44)],
    test = por.test[, -c(14, 15, 16, 43, 44)], cl = por.train$grade.cat,
    k = bestK.por)
# Misclassification Error
1 - mean(por.test$grade.cat == Knn.for.por)
Knn.for.por <- knn(train = por.train[, -c(14, 15, 16, 43, 44)],
    test = por.train[, -c(14, 15, 16, 43, 44)], cl = por.train$grade.cat,
    k = bestK.por)
# Misclassification Error
1 - mean(por.train$grade.cat == Knn.for.por)
```

Model Diagnostics

Check Multicollinearity

```
library(faraway)
## math--------------------
math.ols <- lm(regression.formula, data = math)
# conduct multicollinearity detection
vif(math.ols)
math.mult <- math[, -c(27, 28)]
math.ols.mult <- lm(grade.con ~ . - G1 - G2 - G3 - grade.cat,
    data = math.mult)
summary(math.ols.mult)
# R^2=0.2595 por-----------------
por.ols <- lm(regression.formula, data = por)
# conduct multicollinearity detection
vif(por.ols)
por.mult <- por
por.ols.mult <- lm(grade.con ~ . - G1 - G2 - G3 - grade.cat,
    data = por)
summary(por.ols.mult)
# R^2=0.3247
```

Identifying Outlier

```
library(faraway)
## math--------------------
cook.math <- cooks.distance(math.ols.mult)
halfnorm(cook.math, 5, ylab = "Cook's Distance")
math.mult.out <- math.mult[-c(1, 158, 193, 199, 266), ]
math.ols.mult.out <- lm(grade.con ~ . - G1 - G2 - G3 - grade.cat,
    data = math.mult.out)
summary(math.ols.mult.out)
# R^2=0.2858 por-------------------
cook.por <- cooks.distance(por.ols.mult)
halfnorm(cook.por, 5, ylab = "Cook's Distance")
por.mult.out <- por.mult[-c(1, 604, 500, 524, 550), ]
por.ols.mult.out <- lm(grade.con ~ . - G1 - G2 - G3 - grade.cat,
    data = por.mult.out)
summary(por.ols.mult.out)
# R^2=0.358
```

Check Error Assumption

```
library(MASS)
## math-------------
rn <- rnorm(300, 0, 1)
par(mfrow = c(1, 2))
qqnorm(rn, pch = 1, ylab = "normally generated data points",
    frame = FALSE)
```

```r
qqnorm(math.ols.mult.out$residuals, pch = 1, ylab = "residual of OLS",
    frame = FALSE)
math.mult.out.error <- math.mult.out
math.mult.out.error$grade.con <- math.mult.out.error$grade.con +
    4
math.ols.mult.out.error <- lm(grade.con ~ . - G1 - G2 - G3 -
    grade.cat, data = math.mult.out.error)
bc <- boxcox(math.ols.mult.out.error, plotit = T)
lambda <- bc$x[which.max(bc$y)]
lambda
math.mult.out.error.sq <- math.mult.out.error
math.mult.out.error.sq$grade.con <- (math.mult.out.error.sq$grade.con)^0.5
math.ols.trans <- lm(grade.con ~ . - grade.cat - G1 - G2 - G3,
    data = math.mult.out.error.sq)
summary(math.ols.trans)
# R^2=0.2954 por-------------
rn <- rnorm(300, 0, 1)
par(mfrow = c(1, 2))
qqnorm(rn, pch = 1, ylab = "normally generated data points",
    frame = FALSE)
qqnorm(por.ols.mult.out$residuals, pch = 1, ylab = "residual of OLS",
    frame = FALSE)
por.mult.out.error <- por.mult.out
por.mult.out.error$grade.con <- por.mult.out.error$grade.con +
    4
por.ols.mult.out.error <- lm(grade.con ~ . - G1 - G2 - G3 - grade.cat,
    data = por.mult.out.error)
bc <- boxcox(por.ols.mult.out.error, plotit = T)
lambda <- bc$x[which.max(bc$y)]
lambda
por.mult.out.error.sq <- por.mult.out.error
por.mult.out.error.sq$grade.con <- (por.mult.out.error.sq$grade.con)^0.5
por.ols.trans <- lm(grade.con ~ . - grade.cat - G1 - G2 - G3,
    data = por.mult.out.error.sq)
summary(por.ols.trans)
# R^2=0.3708
```

Nonlinearity Detection

```r
## math--------
library(car)
crPlots(math.ols.trans)
## por--------
library(car)
crPlots(por.ols.trans)
```

Inference About the Equivalence between two regressions

```r
beta1 <- as.vector(math.ols.trans$coefficients)
beta1.cov <- cov(beta1, beta1)
beta2 <- as.vector(por.ols.trans$coefficients)
beta2.cov <- cov(beta2, beta2)
Z <- (beta2.cov - beta1.cov)/sqrt(beta1.cov + beta2.cov)
```

```r
Z
1 - pnorm(Z)
# Two models are more or less equivalent
```

Variable Selection

```r
## math--------------
library(leaps)
math.reg <- regsubsets(grade.con ~ . - grade.cat - G1 - G2 -
    G3, data = math.mult.out.error.sq)
summary(math.reg)
# variables:
# failures+schoolsup_yes+Mjob_other+Fjob_teacher+famsup_yes+studytime+goout+freetime
math.ols.select <- lm(grade.con ~ failures + schoolsup_yes +
    Mjob_other + Fjob_teacher + famsup_yes + studytime + goout +
    freetime, data = math.mult.out.error.sq)
summary(math.ols.select)
## por--------------
por.reg <- regsubsets(grade.con ~ . - grade.cat - G1 - G2 - G3,
    data = por.mult.out.error.sq)
summary(por.reg)
# variables:
# failures+school_MS+higher_yes+studytime+Fjob_teacher+schoolsup_yes+absences+Dalc
por.ols.select <- lm(grade.con ~ failures + school_MS + higher_yes +
    studytime + Fjob_teacher + schoolsup_yes + absences + Dalc,
    data = por.mult.out.error.sq)
summary(por.ols.select)
```