STA567 homework6

Lina Lee

10/31/2019

Remove missing values

```
gaming<-gaming[!(gaming$Age=="?"),]
gaming<-gaming[!(gaming$HoursPerWeek=="?"),]
gaming<-gaming[!(gaming$TotalHours=="?"),]
```

mutate factor variables into numeric variables

```
gaming <- gaming %>%
  mutate(Age=as.numeric(Age),HoursPerWeek=as.numeric(HoursPerWeek),TotalHours=as.nume
ric(TotalHours))
```

Use training/test validation set specification (with 75%/25% split)

Break into training and test data

```
set.seed(11012019)
test_index <- sample(1:nrow(gaming), floor(nrow(gaming)*.25))
game_test <- gaming[test_index,]
game_train <- gaming[-test_index,]
```

1) Regression tree

```
tree_mod <- tree(APM~., data=game_train)
# tree_mod
# summary(tree_mod)
# plot(tree_mod)
# text(tree_mod, pretty=0)

tree_pred <- predict(tree_mod, game_test)
mean((game_test$APM - tree_pred)^2)

## [1] 405.5644
```

2) Bagging

```
set.seed(04052018)
bag_mod <- randomForest(APM ~ . , data=game_train, mtry=13)

bag_pred <- predict(bag_mod, game_test)
mean((bag_pred - game_test$APM)^2)

## [1] 61.95926

# 61.95926
```

3) Random Forest

```
# Tuning number of variable to consider at each split
mses_rf <- rep(NA,13)
set.seed(04052018)
for(m in 1:13){
 rf_mod <- randomForest(APM~., data=game_train, mtry=m)
# validation set testMSE
rf_pred <- predict(rf_mod, game_test)
mses_rf[m]<-mean((rf_pred - game_test$APM)^2)
}
# testMSE
mses_rf

##  [1] 328.29104 177.21751 120.16794  95.21490  80.71583  67.82131  62.80123
##  [8]  61.09345  60.51396  59.61457  59.75723  60.44327  62.08735
```

4) Boosting

```
# create a tuning set of all considered n.tree and interaction.depth combinations
boost_tune <- expand.grid(nt=seq(1000,20000,by=2000),id=1:5)

boost_tune$mse <- NA
set.seed(04052018)
for(i in 1:nrow(boost_tune)){
  boost_mod <- gbm(APM~., data=game_train,
                   n.trees=boost_tune$nt[i], distribution = "gaussian",
                   interaction.depth = boost_tune$id[i])

  boost_pred <- predict(boost_mod, game_test, n.trees=boost_tune$nt[i])
  boost_tune$mse[i] <- mean((game_test$APM - boost_pred)^2)
}
boost_tune[which.min(boost_tune$mse),]

##       nt id     mse
## 20 15000  3 20.3435


boost_tune <- expand.grid(nt=seq(10000,15000,by=500),id=1:5)
boost_tune$mse <- NA
set.seed(04052018)
for(i in 1:nrow(boost_tune)){
  boost_mod <- gbm(APM~., data=game_train,
                    n.trees=boost_tune$nt[i], distribution = "gaussian",
                    interaction.depth = boost_tune$id[i])

  boost_pred <- predict(boost_mod, game_test, n.trees=boost_tune$nt[i])
  boost_tune$mse[i] <- mean((game_test$APM - boost_pred)^2)
}
boost_tune[which.min(boost_tune$mse),]

##       nt id      mse
## 20 14000  2 20.05398
```

For all the tree methods, I included all the numeric variables as the predictors, and fitted models based on the training dataset. I put mtry=13 to fit bagging model. For Random forest, I tried mtry from 1 to 13. The Random forest model with mtry=10 has the lowest Test-MSE. For boosting, I tried the sequence from 1000 to 20000 by 2000 for the number of trees and the sequence from 1 to 5 for id. It gaves me the result that the model with the number of tree = 15000 and id=3 has the lowest test-MSE as 20.3435. I investigated further into the parameters by applying sequence from 1000 to 15000 by 500 for the number of trees. Among all the tree models, the boosting model with the number of trees = 14000 and id=2 performed best with Test-MSE 20.05398 based on the given validation approach.

Table 1 Model Parameters and Test-MSE for Tree Models

Model	Model Parameters	Test-MSE
Tree		405.5644
Bagging	mtry=13	61.9593
Random Forest	mtry=10	59.6146
Boosting	No. of Tree=14000, id=2	20.05398

Models from the take-home exam

1) MLR

```
mlr_mod <- train(APM ~ Age + SelectByHotkeys + MinimapAttacks + MinimapRightClicks +
NumberOfPACs + ActionLatency + ActionsInPAC + TotalMapExplored + WorkersMade,
              data=game_train,
              method="lm",
              trControl=trainControl(method="none"),
              preProcess = c("center", "scale"))


mlr_pred <- predict(mlr_mod, game_test)
#Test-MSE
mean((mlr_pred - game_test$APM)^2)

## [1] 58.78113
```

2) KNN regression

```
knn_mod <- train(APM~. ,
                     data=game_train,
                     method="knn",
                     preProcess=c("center","scale"),
                     tuneGrid=expand.grid(k=8),
                     trControl = trainControl(method="none"))



knn_pred <- predict(knn_mod, game_test)
#Test-MSE
mean((knn_pred - game_test$APM)^2)

## [1] 236.8122
```

3) lasso regression

```
lasso_mod<-train(APM~. ,
                    data=game_train,
                    method="glmnet",
                    preProcess = c("center","scale"),
                    trControl=trainControl(method="none"),
                    tuneGrid=expand.grid(alpha=0,lambda=3.1)
                    )

lasso_pred <- predict(lasso_mod, game_test)
#Test-MSE
mean((lasso_pred - game_test$APM)^2)

## [1] 75.193
```

4) ridge regression

```
ridge_mod<-train(APM~. ,
                    data=game_train,
                    method="glmnet",
                    trControl=trainControl(method="none"),
                    preProcess = c("center","scale"),
                    tuneGrid=expand.grid(alpha=1,lambda=0.1 )
)


ridge_pred <- predict(ridge_mod, game_test)
#Test-MSE
mean((ridge_pred - game_test$APM)^2)

## [1] 59.28582
```

5) Enet

```
### elastic net
enet_mod <- train(APM~ .,
                   data=game_train,
                   method="glmnet",
                   trControl=trainControl(method="none"),
                   preProcess = c("center", "scale"),
                   tuneGrid = data.frame(alpha=1, lambda=0.29))

enet_pred <- predict(enet_mod, game_test)
#Test-MSE
mean((enet_pred - game_test$APM)^2)

## [1] 59.92787
```

6) pcr

```
pcr_mod <- train(APM~ .,
                   data=game_train,
                   method="pcr",
                   preProcess=c("center","scale"),
                   trControl=trainControl(method="none"),
                   tuneGrid = data.frame(ncomp=13))

pcr_pred <- predict(pcr_mod, game_test)
#Test-MSE
mean((pcr_pred - game_test$APM)^2)

## [1] 59.32758
```

7) plsr

```
gaming_plsr <- train(APM~ .,
                   data=game_train,
                   method="pls",
                   preProcess=c("center","scale"),
                   trControl=trainControl(method="none"),
                   tuneGrid = data.frame(ncomp=10))

plsr_pred <- predict(gaming_plsr, game_test)
#Test-MSE
mean((plsr_pred - game_test$APM)^2)

## [1] 59.32127
```

Table 2 Models from the take-home exam

Model	Tuning Parameter	Test-MSE
MLR		58.7811
KNN	k=8	236.8122
lasso	lambda=3.1	75.193
ridge	lambda=0.1	59.2858
enet	alpha=1, lambda=0.29	59.9278
PCR	ncomp=13	59.3276
PLSR	ncomp=10	59.3213

Comparison with tree methods:

The MLR model (APM ~ Age + SelectByHotkeys + MinimapAttacks + MinimapRightClicks + NumberOfPACs + ActionLatency + ActionsInPAC + TotalMapExplored + WorkersMade)has the smallest Test-MSE as 58.7811 among the models in the take-home Exam, the boosting model with the Number of Tree=14000 and id=2 has the lowest test-MSE as 20.3435 among all the tree methods. The boosting model has the lower Test-MSE than the MLR model, therefore the boosting model best predicts the APM using all other numeric variables in the gaming dataset based on the given validation approach.

Problem2: Classification Modeling

Using pima diabetes data: trying to predict test results by measurements

```r
# Use training/test validation set specification (with 75%/25% split)
# Break into training and test data, DO NOT CHANGE THE SEED!
set.seed(11012019)
test_index <- sample(1:768, 192)
pima_test <- pima[test_index,]
pima_train <- pima[-test_index,]
```

Random forest classifier

```r
p<-11012019
m<-1
set.seed(p)
mod_final1 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass1 <- sum(predict(mod_final1, pima_test, type="response")!= pima_test$Class)
```

```
/nrow(pima_test)
  misclass1

## [1] 0.2135417

m<-2
  set.seed(p)
mod_final2 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass2 <- sum(predict(mod_final2, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass2

## [1] 0.1979167

m<-3
  set.seed(p)
mod_final3 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass3 <- sum(predict(mod_final3, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass3

## [1] 0.234375

m<-4
  set.seed(p)
mod_final4 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass4 <- sum(predict(mod_final4, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass4

## [1] 0.203125

m<-5
  set.seed(p)
mod_final5 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass5 <- sum(predict(mod_final5, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass5

## [1] 0.2291667

m<-6
 set.seed(p)
mod_final6 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass6 <- sum(predict(mod_final6, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass6

## [1] 0.2291667

m<-7
 set.seed(p)
mod_final7 <- randomForest(Class~. , data=pima_train, mtry=m)
  misclass7 <- sum(predict(mod_final7, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass7
```

```
## [1] 0.2395833

m<-8
 set.seed(p)
mod_final8<- randomForest(Class~. , data=pima_train, mtry=m)
  misclass8 <- sum(predict(mod_final8, pima_test, type="response")!= pima_test$Class)
/nrow(pima_test)
  misclass8

## [1] 0.234375

mod_final2

## 
## Call:
##  randomForest(formula = Class ~ ., data = pima_train, mtry = m) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
## 
##         OOB estimate of  error rate: 24.65%
## Confusion matrix:
##     neg pos class.error
## neg 331  48   0.1266491
## pos  94 103   0.4771574
```

Answer the following:

2.a) What is the test-misclassification rate that you achieve with your selected models? `0.1979167`

b) How many variables does your selected random forest consider at each split? Number of variables tried at each split: 2

(Actually, as I run and run again with different seed, I get different results.)

2.c) How many trees are in your selected random forest? Number of trees: 500