Feature Selection and Predictive Performance in Incomplete, Higher-Dimensional Datasets

Method 4: MFRF - missForest with Random Forest

Scenario 38: General Overview

The third alternative method follows Method 3 - RF but instead of using the feature median in the imputation step, the imputed dataset using missForest from Method 2 - MFRL is utilized.

Initial Setup

Load relevant packages, set seed and other scenario parameters.

```
packs          = c("doParallel","dplyr","formattable","kableExtra","mltools","randomForest","ti
dyr")
lapply(packs, require, character.only = TRUE)
seedNum        = 103871
set.seed(seedNum)
scenID         = 38
scenName       = "MFRF"
```

Turn on Parallelization

```
cl             = makeCluster(20)
registerDoParallel(cl)
```

Load Train and Test Data

Load simulated datasets created in “1-SimulateDatasets.RMD”.

```r
# Load imputed datasets from method 3 MFRL and configure objects to method 4 RF
load(paste0("S",scenID,"MFRLimpTrainData.Rdata"))
load(paste0("S",scenID,"MFRLimpTestData.Rdata"))
load(paste0("S",scenID,"MFRLyTrain.Rdata"))
load(paste0("S",scenID,"MFRLyTest.Rdata"))
p              = dim(impTrainData[[1]]$ximp)[2]
trainDataSet = testDataSet = vector("list",100)
for (i in 1:100) {
 trainDataSet[[i]]          = cbind(impTrainData[[i]]$ximp,yTrain[[i]])
 colnames(trainDataSet[[i]]) = make.names(c(1:p, "y"))
 testDataSet[[i]] = cbind(impTestData[[i]]$ximp,yTest[[i]])
 colnames(testDataSet[[i]]) = make.names(c(1:p, "y"))
}
# remove imputed data sets
rm(list = c("impTrainData","impTestData"))
```

Random Forest

```r
outrfDataSet   = vector("list", 100)
yHatRF         = matrix(0, nrow=100, ncol=200)
mseRF          = rep(0,100)
varImpTypeOne  = varImpTypeTwo = matrix(0, nrow=100, ncol=p)

for (i in 1:100){
# Random Forest
  outrfDataSet[[i]] = randomForest(y~., data = trainDataSet[[i]],  ntree = 400, importance = TRU
E,  mtry = p/3)
# variable importance
  varImpTypeOne[i,] = importance(outrfDataSet[[i]], type = 1)
# prediction
  yHatRF[i,] = predict(outrfDataSet[[i]], newdata = na.roughfix(testDataSet[[i]]), type = "respo
nse")
  mseRF[i] = mean((yHatRF[,i]-yTest[[i]])^2)
}
```

Feature importance is used for feature selection and an initial indication of predictive performance is obtained below:

```r
cat(paste("RMSE for RF using all predictors:"),round(sqrt(mean(mseRF)),2))
```

```
## RMSE for RF using all predictors: 1.53
```

```r
topTenTypeOne = apply(as.data.frame(varImpTypeOne),1, function(x)names(sort(rank(x))))[p:(p-9),]
```

Select Top 10 Features

Run Random Forest across imputed datasets

```r
top10DataIndex    = matrix(as.numeric(sub('V','',topTenTypeOne)),nrow=10,ncol=100)
outrfTop10        = vector("list", 100)
yHatRFTop10       = matrix(0, nrow=100, ncol=200)
mseRFTop10        = rep(0,100)

for (i in 1:100) {
  xtrainT10       = trainDataSet[[i]][,c(top10DataIndex[,i],p+1)]
  xtestT10        = testDataSet[[i]][,top10DataIndex[,i]]
  ytest           = testDataSet[[i]][,p+1]
# Random Forest
  outrfTop10[[i]] = randomForest(y~., data = xtrainT10,  ntree = 500, importance = FALSE,  mtry
 = 10/3)
# prediction
  yHatRFTop10[i,] = predict(outrfTop10[[i]], newdata = na.roughfix(xtestT10) , type = "response"
)
  mseRFTop10[i]   = mean((yHatRFTop10[,i]-ytest)^2)
}
```

MCC, RMSE and Confusion Matrix averaged over all 100 datasets

Calculate and capture resulting aggregated performance metrics.

```r
#Calculate MCC - Evaluation of Variable Selection (Top 10 version)
vipTypeOne      = apply(as.data.frame(varImpTypeOne),1, function(x)names(sort(rank(x), decreasin
g = TRUE)))
vipTypeOneIndex = matrix(as.numeric(sub('V','',vipTypeOne)),nrow=p,ncol=100)
predic          = matrix(FALSE, nrow=p, ncol=100)  # set up matrix
for (i in 1:100) {
  predic[vipTypeOneIndex[1:10,i],i] = TRUE   # Top 10 VIP reset to TRUE
}
perfect         = c(rep(TRUE, times = 10), rep(FALSE, times = p-10))
mcc             = rep(0,100)
mccF2           = function(x) {mcc(preds = x, actuals = perfect)}
mcc             = apply(predic,2,mccF2)
avgMcc          = mean(mcc)
# Capture results data
truePosT10      = falsePosT10 = trueNegT10 = falseNegT10 = rep(0,p)
truePosT10      = apply(predic[1:10,],2,sum) # true positives
falsePosT10     = 10-truePosT10              # False Positives
falseNegT10     = apply(predic[11:p,],2,sum) # False Negatives
trueNegT10      = p-falseNegT10-falsePosT10-truePosT10

resultsMat      = data.frame(
  scenID   = scenID,
  scenName = scenName,
  method   = "Top10",
  rmse     = round((sqrt(mean(unlist(mseRFTop10)))),2),
  avgMcc   = avgMcc,
  tp       = mean(truePosT10),
  tn       = mean(trueNegT10),
  fp       = mean(falsePosT10),
  fn       = mean(falseNegT10),
  seRMSE   = sd(sqrt(unlist(mseRFTop10))) / sqrt(length(unlist(mseRFTop10))),
  seMCC    = sd(unlist(mcc)) / sqrt(length(unlist(mcc))),
  seTP     = sd(truePosT10) / sqrt(length(truePosT10)),
  seTN     = sd(trueNegT10) / sqrt(length(trueNegT10)),
  seFP     = sd(falsePosT10) / sqrt(length(falsePosT10)),
  seFN     = sd(falseNegT10) / sqrt(length(falseNegT10))
  )
```

Select Features with CV Threshold Method

Load threshold result from MIRL

```
# reset confusion matrix value temporary values
rmse = avgMCC = truePos = trueNeg = falsePos = falseNeg = 0
# load MIRL threshold results
threshRes          = readRDS("S38MIRLThreshPred.Rds")

outrfThresh        = threshDataIndex = vector("list", 100)
yHatRFThresh       = matrix(0, nrow=100, ncol=200)   # n = 200
mseRFThresh        = rep(0,100)
predicThresh       = matrix(FALSE, nrow=p, ncol=100)

for (i in 1:100) {
  threshDataIndex[[i]] = vipTypeOneIndex[1:length(threshRes[[i]]),i]   # predictor indices in ro
ws
  xtrainThresh         = trainDataSet[[i]][,c(threshDataIndex[[i]],p+1)]    # predictor indices
 in columns, y in last column
  xtestThresh          = testDataSet[[i]][,threshDataIndex[[i]]]
# calcs for MCC
  predicThresh[threshDataIndex[[i]],i] = TRUE   # VIP reset to TRUE for threshold selected featu
res
# Random Forest
  outrfThresh[[i]] = randomForest(y~., data = xtrainThresh,  ntree = 500, importance = FALSE,  m
try = max(1,length(threshRes[[i]])/3))
# prediction
  yHatRFThresh[i,] = predict(outrfThresh[[i]], newdata = na.roughfix(xtestThresh) , type = "resp
onse")
  mseRFThresh[i] = mean((yHatRFThresh[,i]-yTest[[i]])^2)
}
```

MCC, RMSE and Confusion Matrix averaged over all 100 datasets

Calculate and capture resulting aggregated performance metrics.

```
#Calculate MCC
mccThresh       = rep(0,10)
mcc             = apply(predicThresh,2,mccF2)
avgMcc          = round(mean(mcc),3)
# Capture results data
truePosThresh   = falsePosThresh = trueNegThresh = falseNegThresh = rep(0,p)
truePosThresh   = apply(predicThresh[1:10,],2,sum) # true positives
falsePosThresh  = apply(predicThresh[11:p,],2,sum) # False Positives
falseNegThresh  = 10-truePosThresh                 # False Negatives
trueNegThresh   = p-truePosThresh-falsePosThresh-falseNegThresh

resultsMat  = resultsMat %>% add_row(
  scenID   = scenID,
  scenName = scenName,
  method   = "Threshold",
  rmse     = round((sqrt(mean(unlist(mseRFThresh)))),2),
  avgMcc   = avgMcc,
  tp       = mean(truePosThresh),
  tn       = mean(trueNegThresh),
  fp       = mean(falsePosThresh),
  fn       = mean(falseNegThresh),
  seRMSE   = sd(sqrt(unlist(mseRFThresh))) / sqrt(length(unlist(mseRFThresh))),
  seMCC    = sd(unlist(mcc)) / sqrt(length(unlist(mcc))),
  seTP     = sd(truePosThresh) / sqrt(length(truePosThresh)),
  seTN     = sd(trueNegThresh) / sqrt(length(trueNegThresh)),
  seFP     = sd(falsePosThresh) / sqrt(length(falsePosThresh)),
  seFN     = sd(falseNegThresh) / sqrt(length(falseNegThresh))
  )
```

Results Summary

The results for both the Top 10 and CV Threshold scenarios using MFRF are shown below. For this scenario, predictive performance, as measured by RMSE, is similar between both scenarios. Feature selection performance is worse for the CV Threshold method, as one would expect given that the Top 10 scenario chooses exactly the right number of features. The CV Threshold scenario chooses an extra 6 features on average, across all 100 datasets.

```
kbl(resultsMat[,1:9], caption = "Table 1: Performance Metrics") %>% kable_styling(position="cent
er", font_size = 12)
```

Table 1: Performance Metrics

scenID	scenName	method	rmse	avgMcc	tp	tn	fp	fn
38	MFRF	Top10	1.57	0.6184	6.82	46.82	3.18	3.18
38	MFRF	Threshold	1.56	0.5170	7.32	41.44	8.56	2.68

Feature Selection Performance

Table 2 displays the number of times each truly associated feature (#1 through 10) is chosen over the 100 datasets. The features with the highest coefficients, #3-5 and 9-10 are chosen at least 88% of the time but there is a reduction in performance, when compared to the RL based methods, in selecting feature 8 as well as in selecting the features with the lower coefficient magnitudes. However, this method performs very well in selecting feature #6.

```
featureRes   = data.frame(
    Weight      = as.factor(rep(1:5,2)/10),
    Association = c(rep("Positive",5),rep("Negative",5)),
    Feature     = paste0('x', 1:10),
    Selected    = as.numeric(table(unlist(threshDataIndex))[1:10]))
featureResOrd   = featureRes[order(featureRes$Weight), ][,c(2,4)]

testF        = function(x) {
                  ifelse(x<=25,"#FF7276",
                    ifelse(x<=50,"#fed8b1",
                      ifelse(x<=75,"lightyellow","lightgreen")))}

tblCol       = testF(featureResOrd$Selected)

featureResOrd$Selected =  color_bar(tblCol)(featureResOrd$Selected)

kbl(featureResOrd, escape = F, caption ="Table 2: Feature Selection Performance Metrics") %>%
  kable_paper("hover", full_width = F) %>%
  column_spec(3, width = "6cm") %>%
  group_rows("Coefficient: 0.1",1,2) %>% group_rows("Coefficient: 0.2",3,4) %>% group_rows("Coef
ficient: 0.3",5,6) %>%       group_rows("Coefficient: 0.4",7,8) %>% group_rows("Coefficient: 0.
5",9,10)
```

Table 2: Feature Selection Performance Metrics

	Association	Selected
Coefficient: 0.1		
1	Positive	17
6	Negative	90
Coefficient: 0.2		
2	Positive	24
7	Negative	46
Coefficient: 0.3		
3	Positive	100
8	Negative	70
Coefficient: 0.4		
4	Positive	100
9	Negative	88
Coefficient: 0.5		
5	Positive	99
10	Negative	98

Turn off Parallelization

```
stopCluster(cl)
```

Save Workspace

```
save.image(file = paste0("S",scenID,scenName,".Rdata"))
saveRDS(resultsMat, file = paste0("S",scenID,scenName,"Results.Rds"))
saveRDS(threshDataIndex, file = paste0("S",scenID,scenName,"ThreshPred.Rds"))
saveRDS(featureRes, file = paste0("S",scenID,scenName,"FeatureRes.Rds"))
```