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

Method 2: MFRL - Multiple Imputation (MissForest) & Random LASSO

Scenario 38: General Overview

The first alternative method is MFRL - MissForest with Random LASSO. Here, the MICE algorithm in step 1 of MIRL is replaced with a tree based multiple imputation method, MissForest (Stekhoven and Buhlmann, 2012). The remaining steps 2 through 4 remain the same as in MIRL. MissForest was found to outperform both K nearest neighbors and MICE when applied to various medical datasets. However, those datasets had smaller amounts of missing data and different pairwise correlations than used in this study.

Initial Setup

Load relevant packages, set seed and other scenario parameters.

```
packs           = c("caret","doParallel","dplyr","formattable","gdata","glmnet","kableExtra","mi
ce","missForest","mltools","rlang","tidyr")
lapply(packs, require, character.only = TRUE)
seedNum         = 103871
set.seed(seedNum)
scenID          = 38
scenName        = "MFRL"
```

Impute Missing Data with MissForest (Random Forest imputation method)

Load Train and Test Data
Load simulated datasets created in “1-SimulateDatasets.RMD”.

```
load(paste0("trainDataSet",scenID,".Rdata"))
load(paste0("testDataSet",scenID,".Rdata"))
p              = dim(trainDataSet[[1]])[2]-1
# separate supervisor from explanatory variables
yTrain         = lapply(trainDataSet, function(x) x[,(p+1)])
yTest          = lapply(testDataSet, function(x) x[,(p+1)])
# remove supervisor from train and test data
trainDataSet   = lapply(trainDataSet, function(x) x[,-(p+1)])
testDataSet    = lapply(testDataSet, function(x) x[,-(p+1)])
```

Turn on Parallelization

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

Impute Missing Values

For the multiple imputation step, MFRL utilizes the missForest algorithm which is a Random Forest based approach. Similar to MICE, missing values are initially set to the feature mean or another starting value. The missing values for a single feature are then predicted using a random forest using all other features. Each feature with missing values is imputed in the same manner, starting with the feature with the fewest missing values and moving to the feature with the most missing values, which completes one full cycle. This process is repeated for multiple cycles until a stopping criterion is met.

```
impTrainData   = impTestData = list()
mfFnct         = function(x) {missForest(as.data.frame(x), maxiter = 10, ntree = 100, mtry=floor
(p/3),                                parallelize = "variables")}
impTrainData   = lapply(trainDataSet, mfFnct)
impTestData    = lapply(testDataSet, mfFnct)
```

Load Random LASSO function

The mirlFunction is created in “2-MIRL.RMD” and is reloaded here.

```
source("mirlFunction.R")
```

Select Top 10 Predictors

Run Random LASSO across imputed datasets

Programming note: missForest results in one imputed, complete dataset rather than 5 as when using MICE, but we still want the same number of bootstrap samples as used in method 1 - MIRL. Furthermore, the mirl function is expecting a mids object type as an input. Inserting new code for E (see #Note 1, below) provides 5 copies of the missForest imputed, complete dataset as a mids object without actually running mice.

```
mirlList   = list()
for(i in 1:100){
  E                = mice(impTrainData[[i]]$ximp,m=5,cluster.seed=seedNum,printFlag=FALSE, maxit
=10) # Note 1
  y                = mirl(trainDataSet[[i]], yTrain[[i]], p/2, im=5, E=E)
  name             = paste('dataset:',i,sep='')
  results          = list(y)
  mirlList[[name]] = results
}
```

Matthew’s Correlation Coefficient to Evaulate Predictor Selection

```
orderedList = tfList = mccFinal = list()
truePos     = trueNeg = falsePos = falseNeg = rep(0,100)

for(i in 1:100) {{{
  ordered             = as.list(order(mirlList[[i]][[1]]$Probability[-1], decreasing=TRUE))
  name                = paste('dataset:',i,sep='')
  resultsList         = list(ordered)
  orderedList[[name]] = resultsList
}
predic      = ifelse(as.list(orderedList[[i]][[1]])>10, FALSE, TRUE)  # predictors 1-10 generate
d y
# Confusion Matrix Data
truePos[i]  = sum(predic[1:10])
falsePos[i] = sum(predic[11:p])
trueNeg[i]  = (p-10-falsePos[i])
falseNeg[i] = 10-sum(predic[1:10])
predicList  = list(predic)
tfList[i]   = predicList
}
perfect     = c(rep(TRUE, times = 10), rep(FALSE, times = (p-10) ))   # perfect selection benchm
ark
mcc         = mcc(preds = tfList[[i]], actuals = perfect)
mccList     = list(mcc)
mccFinal[i] = mccList
}
avgMcc = mean(unlist(mccFinal))
```

Generate Test Dataset Supervisor Predictions

Calculate supervisor prediction for each of 5 imputations, take average.

```
yHatList = yHat = vector("list", 100)
for (i in 1:100){
   top10Pred         = unlist(orderedList[[i]])[1:10]    # vector of predictors chosen
   testDataCoef      = mirlList[[i]][[1]]$coef[-1]       # beta hat initial from step 3
   testDataInt       = mirlList[[i]][[1]]$coef[1]        # beta hat intercept
   top10Coef         = testDataCoef[top10Pred]           # beta hat corresp to top 10 predictors
   testDataTop10     = impTestData[[i]]$ximp[,top10Pred] # imputed data set with only important
 predictors
   yHat[[i]]         = as.matrix(testDataTop10)%*%top10Coef+testDataInt
}
```

RMSE and Confusion Matrix averaged over all 100 datasets

Calculate and capture resulting aggregated performance metrics.

```
completeRMSE = list()
for(i in 1:100){
   rmse            = rmse(unlist(yHat[[i]]),yTest[[i]])
   rmseList        = list(rmse)
   completeRMSE[i] = rmseList
}

# Capture results data
resultsMat  = data.frame(
  scenID   = scenID,
  scenName = scenName,
  method   = "Top10",
  rmse     = round(mean(unlist(completeRMSE)),3),
  avgMcc   = avgMcc,
  tp       = mean(truePos),
  tn       = mean(trueNeg),
  fp       = mean(falsePos),
  fn       = mean(falseNeg),
  seRMSE   = sd(unlist(completeRMSE)) / sqrt(length(unlist(completeRMSE))),
  seMCC    = sd(unlist(mccFinal)) / sqrt(length(unlist(mccFinal))),
  seTP     = sd(truePos) / sqrt(length(unlist(truePos))),
  seTN     = sd(trueNeg) / sqrt(length(unlist(trueNeg))),
  seFP     = sd(falsePos) / sqrt(length(unlist(falsePos))),
  seFN     = sd(falseNeg) / sqrt(length(unlist(falseNeg)))
  )

# clear objects for threshold run
keep(cl, impTestData, impTrainData, mfFnct, mirl, mirlList, orderedList,p, packs, perfect, resul
tsMat, scenID, scenName,seedNum, testDataSet, trainDataSet, yTest, yTrain, sure = TRUE)
```

Select Features with CV Threshold Method

Choose probability threshold by cross validation, when we don’t know how many features are truly associated with the supervisor. Here, the method will determine the number of features to select via cross validation.

Setup Threshold Function

```
threshold<-function(x,y,q2,im=5,m=4,thr=c(0.5,0.6,0.7,0.8,0.9)){
  rand = sample(n)%%m+1
  MSE  = matrix(0,length(thr),m)
  n    = dim(x)[1]               # n is number of obs. in the original data
  p    = dim(x)[2]
  for (i in 1:m){                # m fold cross validation
    X  = x[rand!=i,]
    Y  = y[rand!=i]
    E  = if (!is.mids(X)) {
         MF = mfFnct(X)
         E  = mice(MF$ximp,m=5,cluster.seed=seedNum,printFlag=FALSE, maxit=10)}
    R  = mirl(X,Y,q2,im=5,E=E)   # im is number of imputation
    PP = R[[1]]
   for (j in 1:length(thr)){
      if (max(PP[1:p+1])>thr[j]){
        W        = (PP>thr[j])[2:(p+1)]
        test     = na.omit(cbind(x[rand==i,W],y[rand==i]))
        yt       = test[,ncol(test)]
        xt       = cbind(rep(1,nrow(test)),test[,-ncol(test)])
        cotest   = R[[2]][PP>thr[j]]
        MSE[j,i] = apply((yt-xt%*%cotest)^2,2,mean)
      }
      else{
        yt       = y[rand==i]
        MSE[j,i] = var(yt)*(n-1)/n
      } } }
  mimi = apply(MSE,1,mean,na.rm=1)
  best = which.min(mimi)
  s    = best
  best = thr[s]
}
```

Run Threshold across all datasets

```
n=200                             # fixed for all scenarios
threshList =  R = list()
for(i in 1:100){
  thr                = threshold(trainDataSet[[i]],yTrain[[i]],q2 = p/2, im=5)
  name               = paste('dataset:',i,sep='')
  results            = list(thr)
  threshList[[name]] = results
}
```

MCC, RMSE and Confusion Matrix averaged over all 100 datasets

Calculate and capture resulting aggregated performance metrics for the CV Threshold scenario.

```
tfList = mccFinalT = list()
truePos = trueNeg = falsePos = falseNeg = rep(0,100)

for(i in 1:100) {{
  predic      = ifelse(mirlList[[i]][[1]]$Probability[-1] > threshList[[i]], TRUE, FALSE)
  predicList  = list(predic)
  tfList[i]   = predicList
  truePos[i]  = sum(predic[1:10])
  falseNeg[i] = 10-sum(predic[1:10])
  falsePos[i] = sum(predic[11:p])
  trueNeg[i]  = (p-10-falsePos[i])
}
mcc          = mcc(preds = tfList[[i]], actuals = perfect)
mccList      = list(mcc)
mccFinalT[i] = mccList
}
avgMcc       = round(mean(unlist(mccFinalT)),3)

# Calculate yHat for each imputed dataset (im = 5) and take average
yHat = threshPred = vector("list", 100)
for (i in 1:100){
   countThreshPred = sum(tfList[[i]])
   threshPred[[i]] = unlist(orderedList[[i]])[1:countThreshPred]   # vector of predictors chosen
   testDataCoef    = mirlList[[i]][[1]]$coef[-1]                   # beta hat initial from step
 3
   testDataInt     = mirlList[[i]][[1]]$coef[1]                    # beta hat intercept
   threshCoef      = testDataCoef[threshPred[[i]]]                 # beta hat corresp to top 10
 predictors
   testDataThresh  = impTestData[[i]]$ximp[,threshPred[[i]]]       # imputed data set with only
 important predictors
   yHat[[i]]       = as.matrix(testDataThresh)%*%threshCoef+testDataInt
}

#Calculates RMSE and Average RMSE over all Datasets
threshRMSE = list()
for(i in 1:100){
rmse          = rmse(unlist(yHat[[i]]),yTest[[i]])  # correction here for y actual
rmseList      = list(rmse)
threshRMSE[i] = rmseList
}
# Capture results data
resultsMat  = resultsMat %>% add_row(
  scenID   = scenID,
  scenName = scenName,
  method   = "Threshold",
  rmse     = round(mean(unlist(threshRMSE)),3),
  avgMcc   = avgMcc,
  tp       = mean(truePos),
  tn       = mean(trueNeg),
  fp       = mean(falsePos),
  fn       = mean(falseNeg),
  seRMSE   = sd(unlist(threshRMSE)) / sqrt(length(unlist(threshRMSE))),
  seMCC    = sd(unlist(mccFinalT)) / sqrt(length(unlist(mccFinalT))),
  seTP     = sd(truePos) / sqrt(length(truePos)),
  seTN     = sd(trueNeg) / sqrt(length(trueNeg)),
  seFP     = sd(falsePos) / sqrt(length(falsePos)),
  seFN     = sd(falseNeg) / sqrt(length(falseNeg))
  )
```

Results Summary

The results for both the Top 10 and CV Threshold scenarios using MFRL are shown below. For this scenario, predictive performance, as measured by RMSE, is fairly similar between both scenarios. Feature selection performance is again 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	MFRL	Top10	1.116	0.6772	7.31	47.31	2.69	2.69
38	MFRL	Threshold	1.120	0.5660	7.78	41.83	8.17	2.22

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 8-10 are chosen at least 89% of the time. It is important to recall that features 5 and 10 contained the most missing data but were still reliably selected. The method performs poorly in selecting feature #6 which while it doesn't contain any missing data, is highly correlated with features 3 through 5, is negatively associated with the supervisor and has a lower coefficient magnitude.

```r
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(threshPred))[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)
```

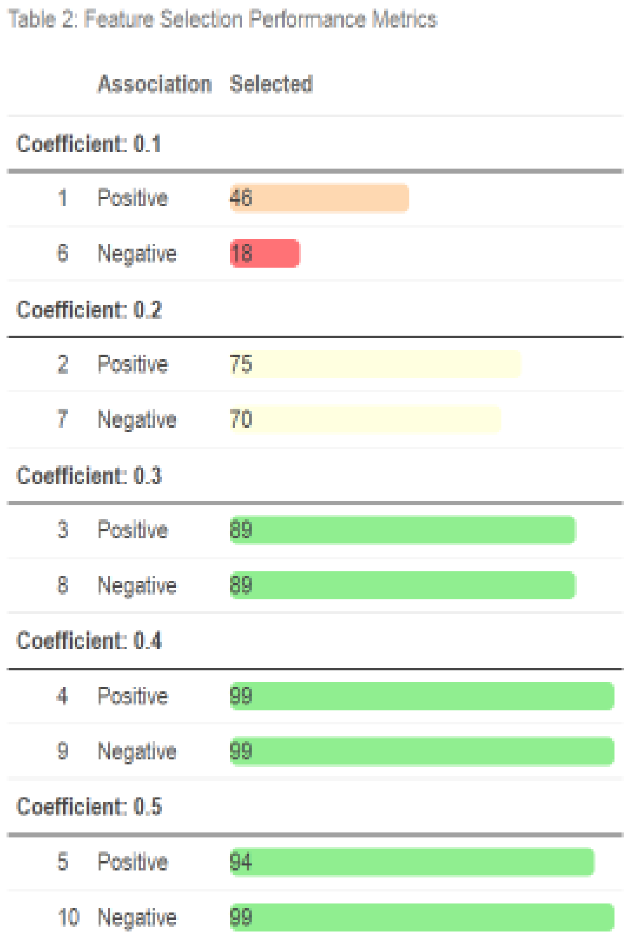

Table 2: Feature Selection Performance Metrics

	Association	Selected
Coefficient: 0.1		
1	Positive	46
6	Negative	18
Coefficient: 0.2		
2	Positive	75
7	Negative	70
Coefficient: 0.3		
3	Positive	89
8	Negative	89
Coefficient: 0.4		
4	Positive	99
9	Negative	99
Coefficient: 0.5		
5	Positive	94
10	Negative	99

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(threshPred, file = paste0("S",scenID,scenName,"ThreshPred.Rds"))
saveRDS(featureRes, file = paste0("S",scenID,scenName,"FeatureRes.Rds"))
save(impTrainData, file = paste0("S",scenID,scenName,"impTrainData.Rdata"))
save(impTestData, file = paste0("S",scenID,scenName,"impTestData.Rdata"))
save(yTrain, file = paste0("S",scenID,scenName,"yTrain.Rdata"))
save(yTest, file = paste0("S",scenID,scenName,"yTest.Rdata"))
```