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

Method 1: MIRL - Multiple Imputation (MICE) & Random LASSO

Scenario 38: General Overview

The base scenario (Method 1: MIRL) is the Multiple Imputation Random LASSO method proposed by Liu, et. al. (2016). MIRL utilizes bootstrap samples of multiply imputed datasets to produce coefficient estimates and perform feature selection. For missing value imputation, the MICE algorithm is used. Random LASSO is used for feature selection, as it performs well when some features are highly correlated. Feature selection is also augmented by stability selection, and coefficient estimates are aggregated.

The method has 4 steps. First, the incomplete dataset is imputed multiple times via MICE, creating 5 iterations of complete datasets. 10 bootstrap samples of each of these complete datasets are created and lasso-OLS coefficient estimates are produced resulting in an importance measure for each feature. Next, Random LASSO is applied, utilizing half of the features, which are selected with a probability proportional to the importance measure created in step 2. Final coefficient estimates are then obtained via aggregation, utilizing stability selection. The number of selected features is either set to 10 (Top 10) to match the true number of features used in creating the supervisor, or determined via cross-validation (CV Threshold). The CV Threshold method employs 4-fold cross validation using a one-standard error rule to determine a feature selection probability threshold.

Initial Setup

Load relevant packages, set seed and other scenario parameters.

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

Impute Missing Data with MICE

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, MIRL utilizes the multivariate imputation by chained equations (MICE) algorithm, although the method is amenable to other methods of imputation, which serves as the basis for Method 2: MFRL. MICE is a regression method where missing values are initially set to the feature mean. The missing values for a single feature are then predicted using regression with all other features. This process is repeated for each individual feature with missing values until a complete dataset is rendered. This process is repeated 10 times (similar to burn in) before taking the resulting complete dataset. This entire process is then repeated 5 times to result in 5 complete datasets.

```
impTrainData    = impTestData = list()
miceFnct        = function(x) {mice(x,m=5,cluster.seed=seedNum,printFlag=FALSE, maxit=10)}
impTrainData    = lapply(trainDataSet, miceFnct)
impTestData     = lapply(testDataSet, miceFnct)
```

Setup Random LASSO method

The multiple imputation portion of step 1 is contained in the above code. The remaining MIRL method is defined below:

```
mirl = function(x=NULL,y,q2,im=5,E,lam=exp(seq(from=log(0.55),to=log(0.001),length.out=70))){
# Step 1 - Multiple Imputation, Center and Scale Data
p    = dim(x)[2]
n    = dim(x)[1]
M    = matrix(0,10*im,p+1)
y    = as.matrix(y)
for (s in 1:im)                             # for sth imputation, im is number of imputations (5)
{ MM           = complete(E,s)              # E is mice generated data
  x            = as.matrix(MM)
  nn           = length(y)
  one          = rep(1, nn)
  meanx        = drop(one %*% x)/nn
  xc           = scale(x, meanx, FALSE)   # first subtracts meanrep
  normx        = sqrt(drop(one %*% (xc^2)))
  names(normx) = NULL
  xs           = scale(xc, FALSE, normx)

# Step 2 - Calculate Variable Importance
for (j in 1:10)   # 10 Bootstrap samples of each imputed dataset
{ w    = sample.int(length(y),replace =TRUE)
  x1   = xs[w,]
  y1   = y[w]
  fit1 = cv.glmnet(x1,y1)
  cod  = as.matrix(predict(fit1,s=fit1$lambda.1se,type="coeff"))
   if((sum(cod!=0)==1)|(sum(cod!=0)>n)){ M[(s-1)*10+j,1] = cod[1]}
   if(sum(cod!=0)!=1){M[(s-1)*10+j,which(cod!=0)] = coef(lm(y1~x1[,which(cod!=0)[-1]-1]))}
} }
# M take records of coef for im imputation and 10 bootstrap samples to compute importance measur
e
m    = dim(M)[2]
Impms= as.matrix(colSums(abs(M[,2:m]))/(10*im))
# Step 3 - MIRL estimates pre threshold
N    = array(0,dim=c(50*im,p+1,length(lam)))   #N records of coef for stability selection
NN   = matrix(0,50*im,p+1)                     #NN for computing coefficient
for (s in 1:im) {
  MM = complete(E,s)
  x  = as.matrix(MM[,1:p])
 for (j in 1:50)
{   w    = sample.int(as.integer(length(y)/2),replace =TRUE)
    v    = sample.int(p,size=q2,prob=Impms)
    x1   = x[w,v]
    y1   = y[w]
    fit1 = cv.glmnet(x1,y1,lambda=lam)
    N[(s-1)*50+j,c(1,v+1),1:dim(predict(fit1,s=fit1$lambda,type="coeff"))[2]] = as.matrix(predic
t(fit1,s=fit1$lambda,type="coeff"))
# Step 3b, 4a Average coefficients and Selection Probability
   #N is used to compute stability selection probability
          bob = as.matrix(predict(fit1,s=fit1$lambda.1se,type="coeff"))
   if((sum(bob!=0)==1)|(sum(bob!=0)>n)){ NN[(s-1)*50+j,1] = bob[1]}
   else{NN[(s-1)*50+j,c(1,v+1)][which(bob!=0)] = coef(lm(y1~x1[,which(bob!=0)[-1]-1]))
} } }
P            = apply(abs(sign(N)),c(2,3),sum)/(50*im)
Probability = apply(P,1,max)
coef        = colSums(NN)/(50*im)
r           = list(Probability=Probability,coef=coef)
}
```

Select Top 10 Features

Run Random LASSO across imputed datasets

In the Top 10 scenario, the model is explicitly directed to only choose 10 features. This serves as a baseline for model performance, but it is unlikely that the true number of associated features will be known in real world applications.

```
mirlList   = list()
for(i in 1:100){
  y                = mirl(trainDataSet[[i]], yTrain[[i]], p/2, im=5, E=impTrainData[[i]])
  name             = paste('dataset:',i,sep='')
  results          = list(y)
  mirlList[[name]] = results
}
```

Matthew's Correlation Coefficient to Evaulate Predictor Selection

Matthew's Correlation Coefficient ("MCC") is used to measure feature selection performance. It is a single measure of the confusion matrix and is defined as follows:

$$MCC = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}$$

TP indicates the number of true positives which is the number of features truly associated with the supervisor that are chosen by the method. TN, or true negatives, are the number of noise features correctly identified by the method. FP, or false positives, are the number of noise features incorrectly selected by the model. FN, or false negatives, are the features truly associated with the supervisor that the model fails to select.

```
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){
   yHatList[[i]]     = matrix(0, nrow=200, ncol=5)
   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
  for (j in 1:5) {                                      # j is number of imp. iterations
   testDataYhat      = complete(impTestData[[i]],j)     # imputed data set
   testDataTop10     = testDataYhat[,top10Pred]         # imputed data set with only important p
redictors
   yHatList[[i]][,j] = as.matrix(testDataTop10)%*%top10Coef+testDataInt  # loop through # of im
p.
  }
   yHat[[i]]         = apply(yHatList[[i]],1,mean)
}
```

RMSE and Confusion Matrix averaged over all 100 datasets

Calculate and capture resulting aggregated performance metrics.

```r
completeRMSE = list()
for(i in 1:100){
   rmse            = rmse(unlist(yHat[[i]]),yTest[[i]])  # correction here for y actual
   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, miceFnct, mirl, mirlList, orderedList,p, packs, perfect, res
ultsMat, 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)) {E=mice(X,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.

```r
tfList = mccFinalT = list()
truePos = trueNeg = falsePos = falseNeg = rep(0,10)

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
yHatList  = yHat  = threshPred = vector("list", 100)
for (i in 1:100){
   yHatList[[i]]   = matrix(0, nrow=200, ncol=5)
   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
  for (j in 1:5) {                                       # j = number of imputation iterations
   testDataYhat      = complete(impTestData[[i]],j)        # imputed test data set
   testDataThresh    = testDataYhat[,threshPred[[i]]]      # imputed data set with only importan
t predictors
   yHatList[[i]][,j] = as.matrix(testDataThresh)%*%threshCoef+testDataInt  # Loop through # of i
mputations
  }
   yHat[[i]]       = apply(yHatList[[i]],1,mean)
}

#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(unlist(truePos))),
  seTN      = sd(trueNeg) / sqrt(length(unlist(trueNeg))),
  seFP      = sd(falsePos) / sqrt(length(unlist(falsePos))),
  seFN      = sd(falseNeg) / sqrt(length(unlist(falseNeg)))
  )
```

Results Summary

The results for both the Top 10 and CV Threshold scenarios using MIRL are shown below. For this scenario, predictive performance, as measured by RMSE, is fairly 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="center", font_size = 12)
```

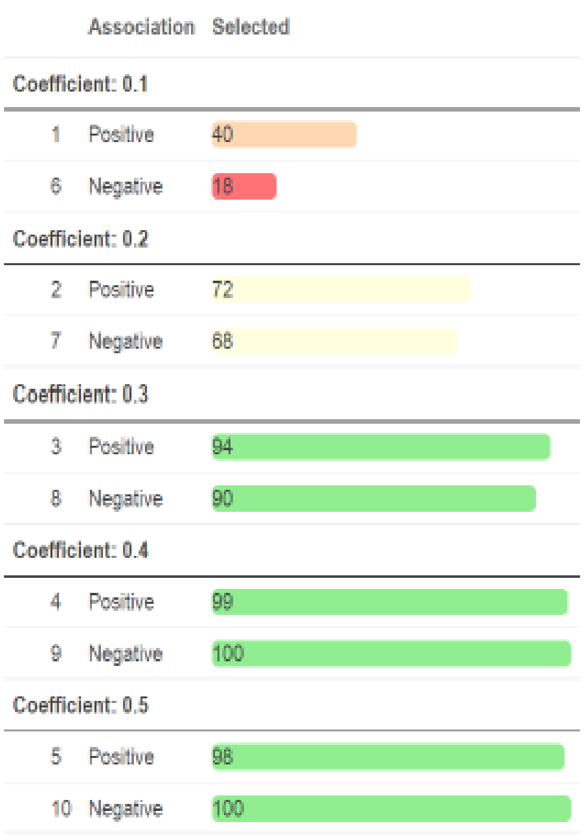
Table 1: Performance Metrics

scenID	scenName	method	rmse	avgMcc	tp	tn	fp	fn
38	MIRL	Top10	1.106	0.6832	7.36	47.36	2.64	2.64
38	MIRL	Threshold	1.103	0.5660	7.79	41.91	8.09	2.21

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 90% 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: 0.5",9,10) %>% group_rows("Coefficient: 0.4",7,8) %>%
    group_rows("Coefficient: 0.3",5,6) %>% group_rows("Coefficient: 0.2",3,4) %>%
     group_rows("Coefficient: 0.1",1,2)
```

Table 2: Feature Selection Performance Metrics



	Association	Selected
Coefficient: 0.1		
1	Positive	40
6	Negative	18
Coefficient: 0.2		
2	Positive	72
7	Negative	68
Coefficient: 0.3		
3	Positive	94
8	Negative	90
Coefficient: 0.4		
4	Positive	99
9	Negative	100
Coefficient: 0.5		
5	Positive	98
10	Negative	100

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"))
```