ALOHA, the below code compare the performance AUC of the deep learning (implemened in H2O R package)

#and six other different algorithms implemented in CARET R package.

#The user is advised to download the data from the below paper:

[1] Budczies, J.; Brockmöller, S. F.; Müller, B. M.; Barupal, D. K.; Richter-Ehrenstein,

C.; Kleine-Tebbe, A.; Griffin, J. L.; Orešič, M.; Dietel, M.; Denkert, C.; et al. Comparative

metabolomics of estrogen receptor positive and estrogen receptor negative breast cancer:

Alterations in glutamine and beta-alanine metabolism. J. Proteomics 2013.

#[2] Budczies, J.; Denkert, C.; Muller, B. M.; Brockmoller, S. F.; Klauschen, F.; Gyorffy, B.; Dietel, M.;

#Richter-Ehrenstein, C.; Marten, U.; Salek, R. M.; Griffin, J. L.; Hilvo, M.; Oresic, M.; Wohlgemuth, G.;#

#Fiehn, O. Remodeling of central metabolism in invasive breast cancer compared to normal breast tissue -

#a GC-TOFMS based metabolomics study. BMC genomics 2012, 13, 334.

#if you have question please fell free to ask falakwaa@hawaii.edu

#This code will run for 10 times randome splitting and it takes one day.

#The below code was tested run on Intel(R) Xeon(R)

#CPU E5-2695 v3 @ 2.30GHz (56 cores), 14.04.1-Ubuntu , R version 3.4.0

################################## Load libraries

#library(mlbench)

library(caret) # six machine learning algorithms

library(h2o)   #Deep learning algorithm

library(pROC)   # AUC plot

library(preprocessCore)   #Quantile normalization

library(RFmarkerDetector) #Scalling

```
library(ggplot2)# Plot of the AUC
localH2O = h2o.init()
################################load the dataset
prostate_df <- read.csv(file="er_162.csv",check.names = T)
prostate_df <- prostate_df[,-1]# reomve sampel labels
prostate_df$subtype <- as.factor(ifelse(prostate_df$subtype==1,1,0))

###################
# Quantile normalization, each column corresponds to a sample and each row is a metabolites.
metadatanorm=normalize.quantiles(t(as.matrix(prostate_df[,-ncol(prostate_df)])))

#################
prostate_df[1:ncol(prostate_df)-1]=t(metadatanorm)



response <- 'subtype'
predictors <- setdiff(names(prostate_df), response)

performance_training=matrix( rep( 0, len=21), nrow = 3)  #AUC   SENS    SPECF
performance_testing=matrix( rep( 0, len=56), nrow = 8)  # ROC  SENS SPEC
performance=matrix(rep( 0, len=56), nrow = 8)  # ROC  SENS SPEC

performance_testing_list=list()
performance_training_list=list()

for (k in 1:10) {
```

```
###############Shuffle stat first
rand <- sample(nrow(prostate_df))
prostate_df=prostate_df[rand, ]

###############Randomly Split  the data in to training and testing
trainIndex <- createDataPartition(prostate_df$subtype, p = .8,list = FALSE,times = 1)
irisTrain <- prostate_df[ trainIndex,]
irisTest  <- prostate_df[-trainIndex,]
irisTrain$subtype=as.factor(paste0("X",irisTrain$subtype))
irisTest$subtype=as.factor(paste0("X",irisTest$subtype))
################################Training and tunning parameters
# prepare training scheme
control <- trainControl(method="cv", number=10,classProbs = TRUE,summaryFunction =
twoClassSummary)

#1- RPART ALGORITHM
set.seed(7)  #This ensures that the same resampling sets are used,which will come in handy when we
compare the resampling profiles between models.

#assign(paste0("fit.cart",k),train(subtype~., data=irisTrain, method="rpart",
trControl=control,metric="ROC"))
fit.cart <- train(subtype~., data=irisTrain, method = 'rpart', trControl=control,metric="ROC") #loclda
performance_training[1,1]=max(fit.cart$results$ROC)#AUC
performance_training[2,1]=fit.cart$results$Sens[which.max(fit.cart$results$ROC)]# sen
performance_training[3,1]=fit.cart$results$Spec[which.max(fit.cart$results$ROC)]# spec

#Model Testing
cartClasses <- predict( fit.cart, newdata = irisTest,type="prob")
cartClasses1 <- predict( fit.cart, newdata = irisTest)
cartConfusion=confusionMatrix(data = cartClasses1, irisTest$subtype)
```

```
 cart.ROC <-
roc(predictor=cartClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))


 performance_testing[1,1]=as.numeric(cart.ROC$auc)#AUC

 performance_testing[2,1]=cartConfusion$byClass[1]#SENS

 performance_testing[3,1]=cartConfusion$byClass[2]#SPEC

 performance_testing[4,1]=cartConfusion$overall[1]#accuracy

 performance_testing[5,1]=cartConfusion$byClass[5]#precision

 performance_testing[6,1]=cartConfusion$byClass[6]#recall = sens

 performance_testing[7,1]=cartConfusion$byClass[7]#F1

 performance_testing[8,1]=cartConfusion$byClass[11]#BALANCED ACCURACY



 # var.cart[[k]]= varImp(fit.cart, scale = FALSE,top=20)

 # var.lda[[k]]= varImp(fit.lda, scale = FALSE,top=20)

 # var.svm[[k]]=varImp(fit.svm, scale = FALSE,top=20)

 # var.rf[[k]]= varImp(fit.rf, scale = FALSE,top=20)

 # var.gbm[[k]]=varImp(fit.gbm, scale = FALSE,top=20)

 # var.pam[[k]]= varImp(fit.pam, scale = FALSE,top=20)


 #2-LDA ALGORITHM

 set.seed(7)

 #assign(paste0("fit.lda",k),train(subtype~., data=irisTrain, method="pls",
trControl=control,metric="ROC"))

 fit.lda <- train(subtype~., data=irisTrain, method = 'lda', trControl=control,metric="ROC") #loclda

 performance_training[1,2]=max(fit.lda$results$ROC)#AUC

 performance_training[2,2]=fit.lda$results$Sens[which.max(fit.lda$results$ROC)]# sen

 performance_training[3,2]=fit.lda$results$Spec[which.max(fit.lda$results$ROC)]# spec
```

```
#Model Testing
ldaClasses <- predict( fit.lda, newdata = irisTest,type="prob")
ldaClasses1 <- predict( fit.lda, newdata = irisTest)
ldaConfusion=confusionMatrix(data = ldaClasses1, irisTest$subtype)
lda.ROC <- roc(predictor=ldaClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))

performance_testing[1,2]=as.numeric(lda.ROC$auc)#AUC
performance_testing[2,2]=ldaConfusion$byClass[1]#SENS
performance_testing[3,2]=ldaConfusion$byClass[2]#SPEC
performance_testing[4,2]=ldaConfusion$overall[1]#accuracy
performance_testing[5,2]=ldaConfusion$byClass[5]#precision
performance_testing[6,2]=ldaConfusion$byClass[6]#recall = sens
performance_testing[7,2]=ldaConfusion$byClass[7]#F1
performance_testing[8,2]=ldaConfusion$byClass[11]#BALANCED ACCURACY

#3- SVM ALGORITHM
set.seed(7)

fit.svm <- train(subtype~., data=irisTrain, method="svmRadial", trControl=control,metric="ROC")
#assign(paste0("fit.svm",k),train(subtype~., data=irisTrain, method="svmRadical",
trControl=control,metric="ROC"))
performance_training[1,3]=max(fit.svm$results$ROC) #AUC
performance_training[2,3]=fit.svm$results$Sens[which.max(fit.svm$results$ROC)]# sen
performance_training[3,3]=fit.svm$results$Spec[which.max(fit.svm$results$ROC)]# spec

#Model Testing
svmClasses <- predict( fit.svm, newdata = irisTest,type="prob")
svmClasses1 <- predict( fit.svm, newdata = irisTest)
svmConfusion=confusionMatrix(data = svmClasses1, irisTest$subtype)
```

```
svm.ROC <-
roc(predictor=svmClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))

performance_testing[1,3]=as.numeric(svm.ROC$auc)#AUC
performance_testing[2,3]=svmConfusion$byClass[1]#SENS
performance_testing[3,3]=svmConfusion$byClass[2]#SPEC
performance_testing[4,3]=svmConfusion$overall[1]#accuracy
performance_testing[5,3]=svmConfusion$byClass[5]#precision
performance_testing[6,3]=svmConfusion$byClass[6]#recall = sens
performance_testing[7,3]=svmConfusion$byClass[7]#F1
performance_testing[8,3]=svmConfusion$byClass[11]#BALANCED ACCURACY

#4-RF ALGORITHM
set.seed(7)
fit.rf <- train(subtype~., data=irisTrain, method="rf", trControl=control,metric="ROC")
performance_training[1,4]=max(fit.rf$results$ROC) #AUC
performance_training[2,4]=fit.rf$results$Sens[which.max(fit.rf$results$ROC)]# sen
performance_training[3,4]=fit.rf$results$Spec[which.max(fit.rf$results$ROC)]# spec

#Model Testing
rfClasses <- predict( fit.rf, newdata = irisTest,type="prob")
rfClasses1 <- predict( fit.rf, newdata = irisTest)
rfConfusion=confusionMatrix(data = rfClasses1, irisTest$subtype)
rf.ROC <- roc(predictor=rfClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))

performance_testing[1,4]=as.numeric(rf.ROC$auc)#AUC
performance_testing[2,4]=rfConfusion$byClass[1]#SENS
performance_testing[3,4]=rfConfusion$byClass[2]#SPEC
performance_testing[4,4]=rfConfusion$overall[1]#accuracy
```

```
performance_testing[5,4]=rfConfusion$byClass[5]#precision
performance_testing[6,4]=rfConfusion$byClass[6]#recall = sens
performance_testing[7,4]=rfConfusion$byClass[7]#F1
performance_testing[8,4]=rfConfusion$byClass[11]#BALANCED ACCURACY

#5- GBM ALGORITHM
set.seed(7)
fit.gbm <- train(subtype~., data=irisTrain, method="gbm", trControl=control,metric="ROC")
performance_training[1,5]=max(fit.gbm$results$ROC) #AUC
performance_training[2,5]=fit.gbm$results$Sens[which.max(fit.gbm$results$ROC)]# sen
performance_training[3,5]=fit.gbm$results$Spec[which.max(fit.gbm$results$ROC)]# spec

#Model Testing
gbmClasses <- predict( fit.gbm, newdata = irisTest,type="prob")
gbmClasses1 <- predict( fit.gbm, newdata = irisTest)
gbmConfusion=confusionMatrix(data = gbmClasses1, irisTest$subtype)
gbm.ROC <-
roc(predictor=gbmClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))

performance_testing[1,5]=as.numeric(gbm.ROC$auc)#AUC
performance_testing[2,5]=gbmConfusion$byClass[1]#SENS
performance_testing[3,5]=gbmConfusion$byClass[2]#SPEC
performance_testing[4,5]=gbmConfusion$overall[1]#accuracy
performance_testing[5,5]=gbmConfusion$byClass[5]#precision
performance_testing[6,5]=gbmConfusion$byClass[6]#recall = sens
performance_testing[7,5]=gbmConfusion$byClass[7]#F1
performance_testing[8,5]=gbmConfusion$byClass[11]#BALANCED ACCURACY

#6- PAM ALGORITHM
```

```
set.seed(7)
fit.pam <- train(subtype~., data=irisTrain, method="pam", trControl=control,metric="ROC")#plr
performance_training[1,6]=max(fit.pam$results$ROC) #AUC
performance_training[2,6]=fit.pam$results$Sens[which.max(fit.pam$results$ROC)]# sen
performance_training[3,6]=fit.pam$results$Spec[which.max(fit.pam$results$ROC)]# spec

#Model Testing
pamClasses <- predict( fit.pam, newdata = irisTest,type="prob")
pamClasses1 <- predict( fit.pam, newdata = irisTest)
pamConfusion=confusionMatrix(data = pamClasses1, irisTest$subtype)
pam.ROC <-
roc(predictor=pamClasses$X1,response=irisTest$subtype,levels=rev(levels(irisTest$subtype)))

performance_testing[1,6]=as.numeric(pam.ROC$auc)#AUC
performance_testing[2,6]=pamConfusion$byClass[1]#SENS
performance_testing[3,6]=pamConfusion$byClass[2]#SPEC
performance_testing[4,6]=pamConfusion$overall[1]#accuracy
performance_testing[5,6]=pamConfusion$byClass[5]#precision
performance_testing[6,6]=pamConfusion$byClass[6]#recall = sens
performance_testing[7,6]=pamConfusion$byClass[7]#F1
performance_testing[8,6]=pamConfusion$byClass[11]#BALANCED ACCURACY


#7- DL ALGORITHM

prostate.hex<-as.h2o(irisTrain, destination_frame="train.hex")
#valid <- as.h2o(irisTest, destination_frame="test.hex")
#prostate.hex$subtype <- as.factor(prostate.hex$subtype) ##make categorical
```

```
  hyper_params <- list(
    activation=c("Rectifier","Tanh"),
    hidden=list(c(100),c(200),c(10,10),c(20,20),c(50,50),c(30,30,30),c(25,25,25,25)),
    input_dropout_ratio=c(0,0.05,0.1),
    #hidden_dropout_ratios=c(0.6,0.5,0.6,0.6),
    l1=seq(0,1e-4,1e-6),
    l2=seq(0,1e-4,1e-6),
    train_samples_per_iteration =c(0,-2),
    epochs = c(500),
    momentum_start=c(0,0.5),
    rho=c(0.5,0.99),
    quantile_alpha=c(0,1),
    huber_alpha=seq(0,1)     )
#path="/home/fadl2/mybest_deeplearning_covtype_model/dl_grid_random1_model_42"
#m_loaded <- h2o.loadModel(path)

  search_criteria = list(strategy = "RandomDiscrete",  max_models = 100, stopping_rounds=5,
stopping_tolerance=1e-2)
    dl_random_grid <- h2o.grid(
    algorithm="deeplearning",
    #grid_id = paste("dl_do3",k,sep = "_"),
    grid_id = "dl_grid_randome1",
    training_frame=prostate.hex,
    #validation_frame=valid,
    #hidden=c(25,25,25,25),
    x=predictors,
    y="subtype",
    #pretrained_autoencoder="dl_grid_random1_model_42",
    #autoencoder = TRUE,
```

```
  seed=7,
  #adaptive_rate=T,              ## manually tuned learning rate
  #momentum_start=0.5,           ## manually tuned momentum
  #momentum_stable=0.9,
  #momentum_ramp=1e7,
  variable_importances=TRUE,
  export_weights_and_biases=T,
  standardize=T,
  stopping_metric="misclassification",
  stopping_tolerance=1e-2,       ## stop when logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  #score_validation_samples=10000, ## downsample validation set for faster scoring
  score_duty_cycle=0.025,        ## don't score more than 2.5% of the wall time
  #max_w2=10,                    ## can help improve stability for Rectifier
  hyper_params = hyper_params,
  search_criteria = search_criteria,
  nfolds=10
 )


 grid <- h2o.getGrid("dl_grid_randome1",sort_by="mse",decreasing=FALSE)
 grid@summary_table[1,]
 best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest logloss


performance_training[1,7]=as.numeric(best_model@model$cross_validation_metrics_summary$mean)
[2] #AUC

performance_training[2,7]=as.numeric(best_model@model$cross_validation_metrics_summary$mean)
[17]# sen
```

```
performance_training[3,7]=as.numeric(best_model@model$cross_validation_metrics_summary$mean)[19]# spec




  perf=h2o.performance(best_model,as.h2o(irisTest, destination_frame="test.hex"))


performance_testing[1,7]=as.numeric(h2o.auc(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#AUC


performance_testing[2,7]=as.numeric(h2o.sensitivity(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#SENS


performance_testing[3,7]=as.numeric(h2o.specificity(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#SPEC


performance_testing[4,7]=as.numeric(h2o.accuracy(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#accuracy


performance_testing[5,7]=as.numeric(h2o.precision(perf,h2o.find_threshold_by_max_metric(perf,"f1")[[1]]))#precision


performance_testing[6,7]=as.numeric(h2o.sensitivity(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#recall = sens


performance_testing[7,7]=as.numeric(h2o.F1(perf,h2o.find_threshold_by_max_metric(perf,"f1"))[[1]])#F1

  performance_testing[8,7]=(performance_testing[2,7]+performance_testing[3,7])/2 #BALANCED ACCURACY




  performance_testing_list[[k]]=performance_testing

  performance_training_list[[k]]=performance_training
```

```
  performance_training=matrix( rep( 0, len=21), nrow = 3)  #AUC   SENS    SPECF
  performance_testing=matrix( rep( 0, len=56), nrow = 8)  # ROC  SENS SPEC

}

saveRDS(performance_testing_list,"performance_testing_list_October_2017.rds")
saveRDS(performance_training_list,"performance_training_list_October_2017.rds")

#}
######################AUC plot_testing
list_test=performance_testing_list
list_train=performance_training_list
ee=lapply(list_test, function(x) x[1,])
output <- do.call(rbind,lapply(ee,matrix,ncol=7,byrow=TRUE))
AUC_mean=apply(output,2,mean)
AUC_mean=data.frame(value=AUC_mean,Algorithm=c('RPART','LDA','SVM','RF','GBM','PAM','DL'))
colnames(AUC_mean)=c('RPART','LDA','SVM','RF','GBM','PAM','DL')

pdf("AUC_mean_testing.pdf")
p<-ggplot(data=AUC_mean, aes(x=Algorithm, y=value)) +  geom_bar(stat="identity")
plot(p)
dev.off()
```