Appendix 1

```
RunML <- function(method, Train_set, Train_label, mode = "Model", classVar){

 method = gsub(" ", "", method)
 method_name = gsub("(\\w+)\\[(.+)\\]", "\\1", method)
 method_param = gsub("(\\w+)\\[(.+)\\]", "\\2", method)

 method_param = switch(
  EXPR = method_name,
  "Enet" = list("alpha" = as.numeric(gsub("alpha=", "", method_param))),
  "Stepglm" = list("direction" = method_param),
  NULL
 )
 message("Run ", method_name, " algorithm for ", mode, "; ",
       method_param, ";",
       " using ", ncol(Train_set), " Variables")

 args = list("Train_set" = Train_set,
          "Train_label" = Train_label,
          "mode" = mode,
          "classVar" = classVar)
 args = c(args, method_param)

 obj <- do.call(what = paste0("Run", method_name),
             args = args)

 if(mode == "Variable"){
  message(length(obj), " Variables retained;\n")
```

```
  }else{message("\n")}
  return(obj)
}

RunEnet <- function(Train_set, Train_label, mode, classVar, alpha){
  cv.fit = cv.glmnet(x = Train_set,
                     y = Train_label[[classVar]],
                     family = "binomial", alpha = alpha, nfolds = 10)
  fit = glmnet(x = Train_set,
               y = Train_label[[classVar]],
               family = "binomial", alpha = alpha, lambda = cv.fit$lambda.min)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunLasso <- function(Train_set, Train_label, mode, classVar){
  RunEnet(Train_set, Train_label, mode, classVar, alpha = 1)
}

RunRidge <- function(Train_set, Train_label, mode, classVar){
  RunEnet(Train_set, Train_label, mode, classVar, alpha = 0)
}

RunStepglm <- function(Train_set, Train_label, mode, classVar, direction){
  fit <- step(glm(formula = Train_label[[classVar]] ~ .,
                  family = "binomial",
                  data = as.data.frame(Train_set)),
              direction = direction, trace = 0)
```

```
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunSVM <- function(Train_set, Train_label, mode, classVar){
  data <- as.data.frame(Train_set)
  data[[classVar]] <- as.factor(Train_label[[classVar]])
  fit = svm(formula = eval(parse(text = paste(classVar, "~."))),
            data= data, probability = T)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunLDA <- function(Train_set, Train_label, mode, classVar){
  data <- as.data.frame(Train_set)
  data[[classVar]] <- as.factor(Train_label[[classVar]])
  fit = train(eval(parse(text = paste(classVar, "~."))),
              data = data,
              method="lda",
              trControl = trainControl(method = "cv"))
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunglmBoost <- function(Train_set, Train_label, mode, classVar){
  data <- cbind(Train_set, Train_label[classVar])
```

```
  data[[classVar]] <- as.factor(data[[classVar]])
  fit <- glmboost(eval(parse(text = paste(classVar, "~."))),
                  data = data,
                  family = Binomial())

  cvm <- cvrisk(fit, papply = lapply,
                folds = mboost::cv(model.weights(fit), type = "kfold"))
  fit <- glmboost(eval(parse(text = paste(classVar, "~."))),
                  data = data,
                  family = Binomial(),
                  control = boost_control(mstop = max(mstop(cvm), 40)))

  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunplsRglm <- function(Train_set, Train_label, mode, classVar){
  cv.plsRglm.res = cv.plsRglm(formula = Train_label[[classVar]] ~ .,
                              data = as.data.frame(Train_set),
                              nt=10, verbose = FALSE)
  fit <- plsRglm(Train_label[[classVar]],
                 as.data.frame(Train_set),
                 modele = "pls-glm-logistic",
                 verbose = F, sparse = T)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}
```

```
RunRF <- function(Train_set, Train_label, mode, classVar){
  rf_nodesize = 5
  Train_label[[classVar]] <- as.factor(Train_label[[classVar]])
  fit <- rfsrc(formula = formula(paste0(classVar, "~.")),
               data = cbind(Train_set, Train_label[classVar]),
               ntree = 1000, nodesize = rf_nodesize,
               importance = T,
               proximity = T,
               forest = T)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunGBM <- function(Train_set, Train_label, mode, classVar){
  fit <- gbm(formula = Train_label[[classVar]] ~ .,
             data = as.data.frame(Train_set),
             distribution = 'bernoulli',
             n.trees = 10000,
             interaction.depth = 3,
             n.minobsinnode = 10,
             shrinkage = 0.001,
             cv.folds = 10,n.cores = 6)
  best <- which.min(fit$cv.error)
  fit <- gbm(formula = Train_label[[classVar]] ~ .,
             data = as.data.frame(Train_set),
             distribution = 'bernoulli',
             n.trees = best,
```

```
          interaction.depth = 3,
          n.minobsinnode = 10,
          shrinkage = 0.001, n.cores = 8)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunXGBoost <- function(Train_set, Train_label, mode, classVar){
  indexes = createFolds(Train_label[[classVar]], k = 5, list=T)
  CV <- unlist(lapply(indexes, function(pt){
    dtrain = xgb.DMatrix(data = Train_set[-pt, ],
                         label = Train_label[-pt, ])
    dtest = xgb.DMatrix(data = Train_set[pt, ],
                        label = Train_label[pt, ])
    watchlist <- list(train=dtrain, test=dtest)

    bst <- xgb.train(data=dtrain,
                     max.depth=2, eta=1, nthread = 2, nrounds=10,
                     watchlist=watchlist,
                     objective = "binary:logistic", verbose = F)
    which.min(bst$evaluation_log$test_logloss)
  }))

  nround <- as.numeric(names(which.max(table(CV))))
  fit <- xgboost(data = Train_set,
                 label = Train_label[[classVar]],
                 max.depth = 2, eta = 1, nthread = 2, nrounds = nround,
                 objective = "binary:logistic", verbose = F)
```

```
  fit$subFeature = colnames(Train_set)

  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}

RunNaiveBayes <- function(Train_set, Train_label, mode, classVar){
  data <- cbind(Train_set, Train_label[classVar])
  data[[classVar]] <- as.factor(data[[classVar]])
  fit <- naiveBayes(eval(parse(text = paste(classVar, "~."))),
                    data = data)
  fit$subFeature = colnames(Train_set)
  if (mode == "Model") return(fit)
  if (mode == "Variable") return(ExtractVar(fit))
}




quiet <- function(..., messages=FALSE, cat=FALSE){
  if(!cat){
    sink(tempfile())
    on.exit(sink())
  }
  out <- if(messages) eval(...) else suppressMessages(eval(...))
  out
}

standarize.fun <- function(indata, centerFlag, scaleFlag) {
  scale(indata, center=centerFlag, scale=scaleFlag)
```

```
}



scaleData <- function(data, cohort = NULL, centerFlags = NULL, scaleFlags = NULL){
  samplename = rownames(data)
  if (is.null(cohort)){
    data <- list(data); names(data) = "training"
  }else{
    data <- split(as.data.frame(data), cohort)
  }


  if (is.null(centerFlags)){
    centerFlags = F; message("No centerFlags found, set as FALSE")
  }
  if (length(centerFlags)==1){
    centerFlags = rep(centerFlags, length(data)); message("set centerFlags for all cohort as ",
unique(centerFlags))
  }
  if (is.null(names(centerFlags))){
    names(centerFlags) <- names(data); message("match centerFlags with cohort by order\n")
  }


  if (is.null(scaleFlags)){
    scaleFlags = F; message("No scaleFlags found, set as FALSE")
  }
  if (length(scaleFlags)==1){
    scaleFlags = rep(scaleFlags, length(data)); message("set scaleFlags for all cohort as ",
unique(scaleFlags))
  }
```

```
  if (is.null(names(scaleFlags))){
    names(scaleFlags) <- names(data); message("match scaleFlags with cohort by order\n")
  }

  centerFlags <- centerFlags[names(data)]; scaleFlags <- scaleFlags[names(data)]
  outdata <- mapply(standarize.fun, indata = data, centerFlag = centerFlags, scaleFlag = scaleFlags,
SIMPLIFY = F)

  outdata <- do.call(rbind, outdata)
  outdata <- outdata[samplename, ]
  return(outdata)
}

ExtractVar <- function(fit){
  Feature <- quiet(switch(
    EXPR = class(fit)[1],
    "lognet" = rownames(coef(fit))[which(coef(fit)[, 1]!=0)],
    "glm" = names(coef(fit)),
    "svm.formula" = fit$subFeature,
    "train" = fit$coefnames,
    "glmboost" = names(coef(fit)[abs(coef(fit))>0]),
    "plsRglmmodel" = rownames(fit$Coeffs)[fit$Coeffs!=0],
    "rfsrc" = var.select(fit, verbose = F)$topvars,
    "gbm" = rownames(summary.gbm(fit, plotit = F))[summary.gbm(fit, plotit = F)$rel.inf>0],
    "xgb.Booster" = fit$subFeature,
    "naiveBayes" = fit$subFeature

  ))
```

```
  Feature <- setdiff(Feature, c("(Intercept)", "Intercept"))
  return(Feature)
}



CalPredictScore <- function(fit, new_data, type = "lp"){
  new_data <- new_data[, fit$subFeature]
  RS <- quiet(switch(
    EXPR = class(fit)[1],
    "lognet"      = predict(fit, type = 'response', as.matrix(new_data)),
    "glm"         = predict(fit, type = 'response', as.data.frame(new_data)),
    "svm.formula" = predict(fit, as.data.frame(new_data), probability = T),
    "train"       = predict(fit, new_data, type = "prob")[[2]],
    "glmboost"    = predict(fit, type = "response", as.data.frame(new_data)),
    "plsRglmmodel" = predict(fit, type = "response", as.data.frame(new_data)),
    "rfsrc"       = predict(fit, as.data.frame(new_data))$predicted[, "1"],
    "gbm"          = predict(fit, type = 'response', as.data.frame(new_data)),
    "xgb.Booster" = predict(fit, as.matrix(new_data)),
    "naiveBayes" = predict(object = fit, type = "raw", newdata = new_data)[, "1"]

  ))
  RS = as.numeric(as.vector(RS))
  names(RS) = rownames(new_data)
  return(RS)
}

PredictClass <- function(fit, new_data){
  new_data <- new_data[, fit$subFeature]
  label <- quiet(switch(
```

```
    EXPR = class(fit)[1],
    "lognet"      = predict(fit, type = 'class', as.matrix(new_data)),
    "glm"         = ifelse(test = predict(fit, type = 'response', as.data.frame(new_data))>0.5,
                           yes = "1", no = "0"),
    "svm.formula" = predict(fit, as.data.frame(new_data), decision.values = T),
    "train"       = predict(fit, new_data, type = "raw"),
    "glmboost"    = predict(fit, type = "class", as.data.frame(new_data)),
    "plsRglmmodel" = ifelse(test = predict(fit, type = 'response', as.data.frame(new_data))>0.5,
                            yes = "1", no = "0"),
    "rfsrc"       = predict(fit, as.data.frame(new_data))$class,
    "gbm"         = ifelse(test = predict(fit, type = 'response', as.data.frame(new_data))>0.5,
                           yes = "1", no = "0"),
    "xgb.Booster" = ifelse(test = predict(fit, as.matrix(new_data))>0.5,
                           yes = "1", no = "0"),
    "naiveBayes" = predict(object = fit, type = "class", newdata = new_data)

  ))
  label = as.character(as.vector(label))
  names(label) = rownames(new_data)
  return(label)
}

RunEval <- function(fit,
                    Test_set = NULL,
                    Test_label = NULL,
                    Train_set = NULL,
                    Train_label = NULL,
                    Train_name = NULL,
                    cohortVar = "Cohort",
```

```
                classVar){

  if(!is.element(cohortVar, colnames(Test_label))) {
    stop(paste0("There is no [", cohortVar, "] indicator, please fill in one more column!"))
  }

  if((!is.null(Train_set)) & (!is.null(Train_label))) {
    new_data <- rbind.data.frame(Train_set[, fit$subFeature],
                                 Test_set[, fit$subFeature])

    if(!is.null(Train_name)) {
      Train_label$Cohort <- Train_name
    } else {
      Train_label$Cohort <- "Training"
    }
    colnames(Train_label)[ncol(Train_label)] <- cohortVar
    Test_label <- rbind.data.frame(Train_label[,c(cohortVar, classVar)],
                                   Test_label[,c(cohortVar, classVar)])
    Test_label[,1] <- factor(Test_label[,1],
                             levels = c(unique(Train_label[,cohortVar]),
setdiff(unique(Test_label[,cohortVar]),unique(Train_label[,cohortVar]))))
  } else {
    new_data <- Test_set[, fit$subFeature]
  }

  RS <- suppressWarnings(CalPredictScore(fit = fit, new_data = new_data))

  Predict.out <- Test_label
  Predict.out$RS <- as.vector(RS)
```

```
  Predict.out <- split(x = Predict.out, f = Predict.out[,cohortVar])
  unlist(lapply(Predict.out, function(data){
    as.numeric(auc(suppressMessages(roc(data[[classVar]], data$RS))))
  }))
}

SimpleHeatmap <- function(Cindex_mat, avg_Cindex,
                          CohortCol, barCol,
                          cellwidth = 1, cellheight = 0.5,
                          cluster_columns, cluster_rows){
  col_ha = columnAnnotation("Cohort" = colnames(Cindex_mat),
                            col = list("Cohort" = CohortCol),
                            show_annotation_name = F)

  row_ha = rowAnnotation(bar = anno_barplot(avg_Cindex, bar_width = 0.8, border = FALSE,
                                            gp = gpar(fill = barCol, col = NA),
                                            add_numbers = T, numbers_offset = unit(-10, "mm"),
                                            axis_param = list("labels_rot" = 0),
                                            numbers_gp = gpar(fontsize = 9, col = "white"),
                                            width = unit(3, "cm")),
                         show_annotation_name = F)

  Heatmap(as.matrix(Cindex_mat), name = "AUC",
          right_annotation = row_ha,
          top_annotation = col_ha,

          col = c("#4195C1", "#FFFFFF", "#CB5746"),
          rect_gp = gpar(col = "black", lwd = 1),
          cluster_columns = cluster_columns, cluster_rows = cluster_rows,
```

```
        width = unit(cellwidth * ncol(Cindex_mat) + 2, "cm"),
        height = unit(cellheight * nrow(Cindex_mat), "cm"),
        column_split = factor(colnames(Cindex_mat), levels = colnames(Cindex_mat)),
        column_title = NULL,
        cell_fun = function(j, i, x, y, w, h, col) {
          grid.text(label = format(Cindex_mat[i, j], digits = 3, nsmall = 3),
                    x, y, gp = gpar(fontsize = 10))
        }
  )
}
```