1. Hands On: Outlier Detection

Load the packages performanceEstimation, mlbench, Rcpp and consider the following example on how to implement a workflow:

```
library(performanceEstimation)
library(mlbench)
library(Rcpp)

example <- function (form, train, test, opt=1,...) {
tgt <- which(colnames(train)==as.character(form[[2]]))
# ... to be done
res <- list(trues=test[,tgt], preds=p) # p must be defined
res }

# Function to calculate AUC
AUC <- function(trues,preds,...) {
  library(AUC) #install.packages("AUC")
  c(auc=AUC::auc(roc(preds,trues)))}

exp1 <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset), # formula and dataset must be defined
        c(workflowVariants("example",opt=c(1,2,3))),
# example is the name of the function previously defined
EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10), evaluator="AUC"))

performanceEstimation::rankWorkflows(exp1, top = 3, maxs = TRUE)
```

1.1 Isolation Forest

(a) Load the dataset PimaIndiansDiabetes and define the formula knowing that the target attribute is diabetes

```
data("PimaIndiansDiabetes")
dataset <- PimaIndiansDiabetes
formula <- diabetes ~ .
```

(b) Using the package solitude, define the od.if function using the example function as template. That function has, as optional parameter, ntrees, that will be passed as the value of the parameter num_trees of the method isolationForest$new. Use the scaled anomaly_score of the prediction. Isolation forest is a supervised method.

```
od.if <- function(form,train,test,ntrees=100,...) {
  require(solitude)
  tgt <- which(colnames(train)==as.character(form[[2]]))
  iso = isolationForest$new(num_trees=ntrees)
  iso$fit(train,...)
  p <- iso$predict(test)
  p <- p$anomaly_score
  p <- scale(p)
  res <- list(trues=test[,tgt], preds=p)
  res }
```

(c) Test isolation forest for num_trees equal to 100, 200 and 500.

```
exp.od.if <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset),
        c(workflowVariants("od.if",ntrees=c(100,250,500,1000))),
  EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.od.if, top = 4, maxs = TRUE)
```

1.2 One-class SVM linear

(a) Create a second workflow for the one-class SVM method with the linear kernel. Use the e1071 package. The optional parameters should be the majority class and the value of nu. Only the instances of the majority class should be used to train the SVM classifier.

```
od.svm.linear <- function(form, train, test, classMaj, n=0.1) {
  require(e1071)
  tgt <- which(colnames(train)==as.character(form[[2]]))
  j <- which(train[,tgt]==classMaj)
  m.svm <- svm(form, train[j,], type='one-classification', kernel='linear', nu=n)
  p <- predict(m.svm,test)
  res <- list(trues=test[,tgt], preds=p)
  res }
```

(b) Test one-class SVM linear values of nu between 0.1 and 0.9 with lags of 0.1.

```
exp.od.svm.linear <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset),
        c(workflowVariants("od.svm.linear", classMaj='neg',
            n=c(seq(0.1,0.9,by=0.1)))),
  EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.od.svm.linear, top = 9, maxs = TRUE)
```

1.3 One-class SVM radial

(a) Create a third workflow for the one-class SVM method with the radial kernel using also the e1071 package. The optional parameters should be the majority class, the value of nu and the value of gamma, the parameter of the radial kernel. Only the instances of the majority class should be used to train the SVM classifier.

```
od.svm.radial <- function(form, train, test, classMaj, g, n=0.1) {
 require(e1071)
 tgt <- which(colnames(train)==as.character(form[[2]]))
 j <- which(train[,tgt]==classMaj)
 m.svm <- svm(form, train[j,], type='one-classification',
          kernel='radial', gamma=g, nu=n)
 p <- predict(m.svm,test)
 res <- list(trues=test[,tgt], preds=p)
 res }
```

(b) Test one-class SVM radial values of gamma between 2^0 and 2^5 and values of nu between 0.1 and 0.9 with lags of 0.1.

```
exp.od.svm.radial <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset),
        c(workflowVariants("od.svm.radial", classMaj='neg',
            g=c(1,2,4,8,16,32), n=c(seq(0.1,0.9,by=0.1)))),
 EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.od.svm.radial, top = 54, maxs = TRUE)
```

1.4 One-class SVM sigmoid

(a) Create a fourth workflow for the one-class SVM method with the Sigmoid kernel using also the e1071 package. The optional parameters should be the majority class, the value of nu, the value of gamma and the value of coef0. Only the instances of the majority class should be used to train the SVM classifier.

```
od.svm.sigmoid <- function(form, train, test, classMaj, g, c0, n=0.1) {
 require(e1071)
 tgt <- which(colnames(train)==as.character(form[[2]]))
 j <- which(train[,tgt]==classMaj)
 m.svm <- svm(form, train[j,], type='one-classification',
          kernel='sigmoid', gamma=g, coef0=c0, nu=n)
 p <- predict(m.svm,test)
 res <- list(trues=test[,tgt], preds=p)
 res }
```

(b) Test one-class SVM radial values of gamma between 2^0 and 2^5, values of nu between 0.1 and 0.9 with lags of 0.1, and values of coef0 between 2^0 and 2^5.

```
exp.od.svm.sigmoid <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset),
        c(workflowVariants("od.svm.sigmoid", classMaj='neg',g=c(1,2,4,8,16,32),
           c0= c(1,2,4,8,16,32),n=c(seq(0.1,0.9,by=0.1)))),
  EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.od.svm.sigmoid, top = 324, maxs =
TRUE)
```

1.5 Local Outlier Factor

(a) Create a fifth workflow for the local Outlier Factor (LOF) method (library Rlof). The optional parameter should be the threshold used by LOF. An instance is considered outlier when is local outlier factor is larger than the threshold. LOF is an unsupervised method. Consequently, the target attribute should not be used. How to use the training/test mechanism of the workflow in this case?

```
od.lof <- function(form,train,test,threshold=1,...) {
  require(Rlof)
  tgt <- which(colnames(train)==as.character(form[[2]]))
  ntrain <- nrow(train)
  ntest <- nrow(test)
  df <- rbind(train,test)
  df <- df[,-tgt]
  closeAllConnections()
  df.lof <- lof(df,c(1,2,4,8,16,32),cores=1)
  p <- c()
  for (i in (ntrain+1):(ntrain+ntest)) {
    if (max(df.lof[i,])>threshold) p <- c(p, 'pos')
    else p <- c(p, 'neg') }
  res <- list(trues=test[,tgt], preds=p)
  res }
```

(b) Test the threshold values of 0.1, 0.5, 1, 2 and 4 for LOF.

```
exp.od.lof <- performanceEstimation::performanceEstimation(
        PredTask(formula,dataset),
        c(workflowVariants("od.lof",threshold=c(0.1,0.5,1,2,4))),
  EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.od.lof, top = 5, maxs = TRUE)
```

1.6 Overall comparison

(a) Use all the workflows together using the values for the hyper-parameters at your choice and rank the first 500 results.

```
exp.overall <- performanceEstimation::performanceEstimation(
          PredTask(formula,dataset),
          c(workflowVariants("od.if",ntrees=c(100,250,500,1000)),
            workflowVariants("od.svm.linear", classMaj='neg',
                n=c(seq(0.2,0.3,by=0.01))),
            workflowVariants("od.svm.radial", classMaj='neg',
                g=c(1,2,4,8,16,32), n=c(seq(0.1,0.3,by=0.01))),
            workflowVariants("od.svm.sigmoid", classMaj='neg',
                g=c(1,2,4,8,16,32), c0=c(1,2,4,8,16,32),
                n=c(seq(0.1,0.3,by=0.02))),
           workflowVariants("od.lof",threshold=c(0.1,0.5,1,2,4))),
  EstimationTask(metrics="auc",method=CV(nReps = 1, nFolds=10),
evaluator="AUC"))
performanceEstimation::rankWorkflows(exp.overall, top = 500, maxs = TRUE)
```