# Documented program code

## Data mining project, academic year 2019/2020

#tree.grow

#x: data matrix containing attribute values of the training set

#y: vector of class labels

#nmin: number of observations that a node must contain at least, for it to be allowed to be split

#minleaf: minimun number of observations required for a leaf node

#nfeat: number of features to consider for each split

#returns: a dataframe where each row represents a node of the tree

tree.grow <- function(x,y,nmin, minleaf,nfeat){

#creation of the dataframe with the root node

#the semantics of the attributes is explained where they are used

tree <- data.frame (1, -1, length(y[y==0]), sum(y[y==1]),list(1),-1,-1,-1,-1,1)

names(tree) <- c("node.id", "parent.id", "class.0", "class.1", "observations", "left.id", "right.id", "split.attribute", "split.value", "is.splittable")

tree$observations <- list(c(1:length(y)))

#add observations identifiers

x<- cbind(x, c(1:nrow(x)))

#is.splittable == 1 <==> the node could be split

#is.splittable == 0 <==> the node has already been splitted or can't be splitted ( it's a leaf node)

while (sum(tree$is.splittable) > 0){#while there are splittable nodes

splittable.rows <- which(tree$is.splittable == 1)

for(row in splittable.rows){

i <- tree[row,]

#the attribute "observations" contains a list of the identifiers of the observations pertaining to the current node

if (length(i$observations[[1]]) < nmin){#the node can't be splitten

tree[tree$node.id == i$node.id,]$is.splittable <- 0

}

else{

#computation of all the best splitpoints for each attribute

#we may want to use not all the attributes for random forests

#we select the right observations

x.filter = x[as.vector(unlist(i$observations)),]

y.filter = y[as.vector(unlist(i$observations))]

attributes.numbers <- sample(1:(ncol(x.filter)-1), nfeat, replace=FALSE)

splitpoints <- sapply(attributes.numbers, function(i){bestsplit(x.filter[,i],y.filter, minleaf)})

splitpoints <- t(splitpoints)

#introduce a new column to keep track of the attribute corresponding to each splitpoint

splitpoints <- cbind(splitpoints,attributes.numbers)

#select only valid splitpoints, filtering out Na

rows = complete.cases(splitpoints)

splitpoints <- splitpoints[rows, ]

if (length(splitpoints) == 0){#there is no splitpoint which respects the minleaf condition

tree[tree$node.id == i$node.id,]$is.splittable <- 0

}

else{

if(is.vector(splitpoints)){#there is only one candidate splitpoint

#select the attribute and the value on which splitting

split.value <- splitpoints[ 1]

split.attribute <- splitpoints[ 3]

}

else{# there are more than one candidate splitpoints

#order the splitpoints by their quality

splitpoints <- splitpoints[order(splitpoints[,2], decreasing = TRUE), ]

#select the attribute and the value on which splitting

split.value <- splitpoints[1 , 1]

split.attribute <- splitpoints[1, 3]

}

#determine the observations per each child

left.obs <- x.filter[x.filter[,split.attribute]< split.value, ncol(x.filter)]

right.obs <- x.filter[x.filter[,split.attribute] >= split.value, ncol(x.filter)]

#determine the number of elements per each class per each child node

left.obs.1 <- sum(y[left.obs])

left.obs.0 <- length(left.obs) - left.obs.1

right.obs.1 <- sum(y[right.obs])

right.obs.0 <- length(right.obs) - right.obs.1

#determine the ids of the two new nodes

left.id <- max(tree$node.id) +1

right.id <- left.id +1

#create the new nodes

left.child <- data.frame (left.id, i$node.id,left.obs.0,left.obs.1,-1,-1,-1,-1,-1,1)

names(left.child) <- c("node.id", "parent.id", "class.0", "class.1", "observations", "left.id", "right.id", "split.attribute", "split.value", "is.splittable")

left.child$observations <- list(left.obs)

right.child <- data.frame (right.id, i$node.id,right.obs.0,right.obs.1,-1,-1,-1,-1,-1,1)

names(right.child) <- c("node.id", "parent.id", "class.0", "class.1", "observations", "left.id", "right.id", "split.attribute", "split.value", "is.splittable")

right.child$observations <- list(right.obs)

#update the parent node and the tree

tree[tree$node.id == i$node.id,]$left.id <- left.id

tree[tree$node.id == i$node.id,]$right.id <- right.id

tree[tree$node.id == i$node.id,]$is.splittable <- 0

tree[tree$node.id == i$node.id,]$split.attribute <- split.attribute

tree[tree$node.id == i$node.id,]$split.value <- split.value

tree <- rbind(tree, left.child)

tree <- rbind(tree, right.child)

}

}

}

}

return (tree)

}

#tree.classify

#x: data matrix containing the attribute values of the cases for which predictions are required

#tr: tree object used to classify the observations

#returns: a vector of predicted class labels for the cases in x

tree.classify <- function(x, tr){

result <- apply(x,1, function(i){

current.node <- tr[1, ]

#the split attribute represents the number of the column of the attribute values on which splitting

#if it's equal to -1 it means that we have reached a leaf node

while(current.node$split.attribute != -1){# we go through the dataframe until we reach a leaf node

if (i[current.node$split.attribute] < current.node$split.value){

current.node <- tr[current.node$left.id, ]

}

else{

current.node <- tr[current.node$right.id, ]

}

}

#we return a value depending on the majority of the observations in that leaf node

if (current.node$class.0 >= current.node$class.1){# if there is a tie we return 0

return (0)

}

else return (1)

})

}

#tree.grow.bag

#x: data matrix containing attribute values of the training set

#y: vector of class labels

#nmin: number of observations that a node must contain at least, for it to be allowed to be split

#minleaf: minimun number of observations required for a leaf node

#m: number of bootstrap samples to be drawn

#returns: a list of dataframes where each dataframe represents a tree, and each row a node of the tree

tree.grow.bag <- function(x,y,nmin, minleaf,nfeat,m){

list.trees <- list()

#grow trees and put in a list

list.trees <- lapply(c(1:m), function(i){

bootstrap.samples <- sample(nrow(x), size=nrow(x), replace = TRUE)

x.bootstrap = x[bootstrap.samples,]

y.bootstrap = y[bootstrap.samples]

tree <- tree.grow(x.bootstrap,y.bootstrap,nmin,minleaf,nfeat)

return (tree)

})

return (list.trees)

}

#tree.classify.bag

#x: data matrix containing the attribute values of the cases for which predictions are required

#list.trees: list of trees object used to classify the observations

#returns: a vector of predicted class labels for the cases in x

tree.classify.bag <- function(x, list.trees){

#apply the classify function for each tree of the list

#returns a matrix where each row is a vector of class labels for an observation

classification\_matrix <- sapply(list.trees, function(tree){

labels <- tree.classify(x, tree)

return (labels)

})

#decide on the majority of results

classification\_matrix <- apply(classification\_matrix, 1, function(row){

if (length(row[row == 0]) >= length(row[row == 1])){#in case of tie we return 0

return (0)

}

else return (1)

})

return (classification\_matrix)

}

#impurity

#y: vector of binary elements

#returns: the impurity of the vector

impurity <- function(y){

if(length(y)== 0){

return (0)

}

x <- sum(y)/ length(y)

result <- x\*(1-x)

}

# x: vector of values of the observations for an attribute

# y: vector of class labels

# minleaf: minimum minimun number of observations required for a leaf node

#returns: a data matrix composed of two columns:

#splitpoints (the first column) is the value of the best splitpoint

#quality(the second column) is the quality of the split

bestsplit <- function(x,y,minleaf){

#create a unique matrix and order it

my\_matrix <- cbind(x,y)[order(x,y),]

#select the splitpoints

#if a splitpoint is not on a segment border, return NA

splitpoints <- sapply(c(2:nrow(my\_matrix)-1), function(i){

if(my\_matrix[i,2] != my\_matrix[i+1, 2]){

return ((my\_matrix[i,1]+ my\_matrix[i+1,1])/2)

}

else return (NA)

})

splitpoints <- splitpoints[!is.na(splitpoints)]

#discard splitpoints which do not satisfy the minleaf condition

splitpoints <- sapply(splitpoints, function(i){

if (length(x[x<i]) < minleaf || length(x[x >= i])< minleaf){

return (NA)

}

else {

return (i)

}

})

splitpoints <- splitpoints[!is.na(splitpoints)]

#if no splitpoint has been found, we return an empty matrix

if (length(splitpoints)==0){

empty.matrix <- matrix(nrow=1, ncol=2, byrow = FALSE, dimnames = list(c(),c("splitpoints", "quality")))

return (empty.matrix)

}

#compute the impurities of the children nodes generated by each split

impurities <- sapply(splitpoints, function(i) {

less\_mat <- my\_matrix[which(my\_matrix[,1] <i), 2]

great\_mat <- my\_matrix[which(my\_matrix[,1] >= i), 2]

length(less\_mat)\*impurity(less\_mat)/length(y) + length(great\_mat)\*impurity(great\_mat)/length(y)

})

#compute the quality of splits

quality <- impurity(y) - impurities

quality\_matrix <- cbind(splitpoints, quality)

#select the best split and return it

quality\_matrix <- quality\_matrix[which.max(quality\_matrix[,2]), ]

quality\_matrix

}

#utilities.precision

#actual values: vector of actual class labels of the test set

#predictions: vector of the predictions

#returns: the precision of the classification model

utilities.precision <- function (actual.values, predictions){

matrix <- cbind(actual.values, predictions)

return ((true.positives(matrix)/(true.positives(matrix) + false.positives(matrix))))

}

#utilities.recall

#actual values: vector of actual class labels of the test set

#predictions: vector of the predictions

#returns: the recall of the classification model

utilities.recall <- function (actual.values, predictions){

matrix <- cbind(actual.values, predictions)

return ((true.positives(matrix)/(true.positives(matrix) + false.negatives(matrix))))

}

#utilities.accuracy

#actual values: vector of actual class labels of the test set

#predictions: vector of the predictions

#returns: the accuracy of the classification model

utilities.accuracy <- function(actual.values, predictions){

matrix <- cbind(actual.values, predictions)

return (((true.negatives(matrix) + true.positives(matrix))/nrow(matrix)))

}

#true.positives

#matrix: a matrix containing the actual values as first column and the predictions as second column

#returns: the number of true positives

true.positives <- function(matrix){

true.positive <- apply(matrix, 1, function(row){

if(row[1] ==1 && row[2] ==1){

return (1)

}

else return (0)

})

return (sum(true.positive))

}

#false.positives

#matrix: a matrix containing the actual values as first column and the predictions as second column

#returns: the number of false positives

false.positives <- function(matrix){

false.positive <- apply(matrix, 1, function(row){

if(row[1] ==0 && row[2] ==1){

return (1)

}

else return (0)

})

return (sum(false.positive))

}

#true.negatives

#matrix: a matrix containing the actual values as first column and the predictions as second column

#returns: the number of true negatives

true.negatives <- function(matrix){

true.negative <- apply(matrix, 1, function(row){

if(row[1] ==0 && row[2] ==0){

return (1)

}

else return (0)

})

return (sum(true.negative))

}

#false.negatives

#matrix: a matrix containing the actual values as first column and the predictions as second column

#returns: the number of false negatives

false.negatives <- function(matrix){

false.negative <- apply(matrix, 1, function(row){

if(row[1] ==1 && row[2] ==0){

return (1)

}

else return (0)

})

return (sum(false.negative))

}