CSE455/CSE552 – Machine Learning (Spring 2016)

Homework #2 Report

Murat ALTUNTAŞ (111044043)

Part 1:

Code:

```
##############################################################################
################################  Library  ################################
##############################################################################
library("data.tree")
library("entropy")
library("party")
library("rpart")
##############################################################################
################################# Functions ###############################
##############################################################################
PruningVal <- 0  # pruning değeri
decisionTree <- function(data, pruningValue) {
 allGains <- c()
 threshold <- c()
 # big entropy
 bigEnt <- entropy.empirical(freqs.empirical(table(data[[5]])))
# print(bigEnt)
# print(data)
 if(bigEnt <= pruningValue){
  return(Node$new(as.character(names(sort(table(data[[5]]),decreasing=TRUE)[1:1]))))
 }

  for (column in 1:4) {
    entropiesl <- c()
    entropiesr <- c()
    gains <- c()
    counter <- 1
```

```
    maxV <- max(data[[column]], na.rm = FALSE)
    minV <- min(data[[column]], na.rm = FALSE)
    #   print(maxV)
    #   print(minV)
    vector <- seq(minV+((maxV-minV)/300), maxV-((maxV-minV)/300), ((maxV-minV)/300))
    #   print(vector)
    for (i in vector) {
      iris_l <- data[data[[column]] < i,]
      #     print(data)
      #     print(data[[column]])
      iris_r <- data[data[[column]] >= i,]
      entropiesl[counter] <- entropy.empirical(freqs.empirical(table(iris_l[[5]])))
      entropiesr[counter] <- entropy.empirical(freqs.empirical(table(iris_r[[5]])))
      if(is.na(entropiesl[counter]))
      {
        entropiesl[counter] <- 0
      }
      if(is.na(entropiesr[counter]))
      {
        entropiesr[counter] <- 0
      }
      #     print(entropy.empirical(freqs.empirical(table(iris_l[[5]]))))
      gains[counter] <- bigEnt - ((entropiesl[counter] * (nrow(iris_l) / nrow(data))) +
(entropiesr[counter] * (nrow(iris_r) / nrow(data))))
      if(is.na(gains[counter]))
      {
        gains[counter] <- 0
      }
      #     print(gains[counter])
      counter <- counter + 1
    }

    #   print(entropiesl)

    allGains[column] <- max(gains, na.rm = FALSE)
```

```
      match(max(gains, na.rm = FALSE),gains)
      threshold[column] <- vector[match(max(gains, na.rm = FALSE),gains)]
   }
#     print(gains)
   rootColNum <- match(max(allGains, na.rm = FALSE),allGains)
#     print(allGains)
   result <- colnames(data[rootColNum])

   rootLabel <- Node$new(paste (result, threshold[rootColNum], sep = " ", collapse = NULL))
   child_l <- data[data[[rootColNum]] <  threshold[rootColNum],]
   rootLabel$AddChildNode(decisionTree(child_l, pruningValue))
   child_r <- data[data[[rootColNum]] >= threshold[rootColNum],]
   rootLabel$AddChildNode(decisionTree(child_r, pruningValue))

#  print(threshold[rootColNum])

   return(rootLabel)
}

myTreePredict <- function(myNode, testData) {

   rootNum <- strsplit(myNode$name, " ")
   index <- match(rootNum[[1]][1], colnames(testData)) # karşılaştırılacak kolonun indexi

   if(myNode$isLeaf){
      return(myNode$name)
   }

   if(testData[index] < rootNum[[1]][2]){
      myTreePredict(myNode$children[[1]] ,testData)
   }
   else
   {
      myTreePredict(myNode$children[[2]] ,testData)
   }
```

```
}


testPredict <- function(tree, testData){
   resultLabels <- c()
   for (i in 1:nrow(testData)) {
      resultLabels[i] <- myTreePredict(tree,testData[i,])
   }
   return(resultLabels)
}


### Karşılaştırma ###
### ctree ###
ctreeTest <- function(myData){
   oran <- 0
   myData<-myData[sample(nrow(myData)),]
   #Create 10 equally size folds
   folds <- cut(seq(1,nrow(myData)),breaks=10,labels=FALSE)
   gp <- runif(nrow(myData))   # random siralama
   myData <- myData[order(gp),]

   for(i in 1:10){
      #-- train ve test olarak ayırma --#
      #Segement your data by fold using the which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
      myData_test <- myData[testIndexes, ]
      myData_train <- myData[-testIndexes, ]
      myData_test_target <- myData[testIndexes, 5]


      root <- ctree(Species ~ . , data=myData_train)
      resultLbls <- (as.character(myData_test_target) == predict(root, newdata = myData_test, type
= "response"))
      if(length(table(resultLbls)) == 1)
      {
         oran <- table(resultLbls)[[1]] / length(resultLbls) + oran
      }
      else
```

```
        {
            oran <- table(resultLbls)[[2]] / length(resultLbls) + oran
        }
    }

    cat("ctree %", ((oran/10) * 100))
}

### rpart ###
rpartTest <- function(myData){
    oran <- 0
    myData<-myData[sample(nrow(myData)),]
    #Create 10 equally size folds
    folds <- cut(seq(1,nrow(myData)),breaks=10,labels=FALSE)
    gp <- runif(nrow(myData))   # random siralama
    myData <- myData[order(gp),]

    for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        myData_test <- myData[testIndexes, ]
        myData_train <- myData[-testIndexes, ]
        myData_test_target <- myData[testIndexes, 5]

        root <- rpart(Species ~ . , method="class", data=myData_train, parms = list(split = "information"))
        pfit<- prune(root, cp=PruningVal)
        resultLbls <- (as.character(myData_test_target) == predict(pfit, newdata = myData_test, type = "class"))
        if(length(table(resultLbls)) == 1)
        {
            oran <- table(resultLbls)[[1]] / length(resultLbls) + oran
        }
        else
        {
```

```
            oran <- table(resultLbls)[[2]] / length(resultLbls) + oran
        }
    }

    cat("rpart %", ((oran/10) * 100))
}

### My Decision Tree ###
myDecisionTreeTest <- function(myData){
    oran <- 0
    myData<-myData[sample(nrow(myData)),]
    #Create 10 equally size folds
    folds <- cut(seq(1,nrow(myData)),breaks=10,labels=FALSE)
    gp <- runif(nrow(myData))   # random siralama
    myData <- myData[order(gp),]

    for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        iris_test <- myData[testIndexes, ]
        iris_train <- myData[-testIndexes, ]
        iris_test_target <- myData[testIndexes, 5]

        root <- decisionTree(iris_train,PruningVal)
        resultLbls <- (as.character(iris_test_target) == testPredict(root,iris_test))
        if(length(table(resultLbls)) == 1)
        {
            oran <- table(resultLbls)[[1]] / length(resultLbls) + oran
        }
        else
        {
            oran <- table(resultLbls)[[2]] / length(resultLbls) + oran
        }
    }
```

```
   cat("My Decision Tree %", ((oran/10) * 100))
}

part3 <- function(train_data, testDatasi){
   partLabels <- c()
   roots <- list()
#   testDatasi <- train_data[-(1:85),]
   for (k in 1:nrow(testDatasi)) {
      resultLabels <- c()
      for (j in 1:5) {
         train_data <- train_data[sample(nrow(train_data)),]
         concatData <- train_data[1:85,]
         for (i in 1:50) {
            newindex <- sample(1:85, 1)
            concatData <- rbind(concatData, train_data[newindex,])
         }

         root <- decisionTree(concatData,PruningVal)
         #partLabels[j] <- testPredict(root, testDatasi)

         resultLabels[j] <- myTreePredict(root,testDatasi[k,])

      }
      partLabels[k] <- names(sort(table(resultLabels),decreasing=TRUE)[1:1])
   }
   return(partLabels)
}

baggingTest <- function(myData){
   oran <- 0
   myData<-myData[sample(nrow(myData)),]
   #Create 10 equally size folds
   folds <- cut(seq(1,nrow(myData)),breaks=10,labels=FALSE)
   gp <- runif(nrow(myData))   # random siralama
```

```
  myData <- myData[order(gp),]



  for(i in 1:10){
    #-- train ve test olarak ayırma --#
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==i,arr.ind=TRUE)
    iris_test <- myData[testIndexes, ]
    iris_train <- myData[-testIndexes, ]
    iris_test_target <- myData[testIndexes, 5]

    resultLbls <- (as.character(iris_test_target) == part3(iris_train,iris_test))
    if(length(table(resultLbls)) == 1)
    {
      oran <- table(resultLbls)[[1]] / length(resultLbls) + oran
    }
    else
    {
      oran <- table(resultLbls)[[2]] / length(resultLbls) + oran
    }
  }

  cat("Bagging Test %", ((oran/10) * 100))
}
##############################################################################
################################# Part 1 ################################
##############################################################################


ctreeTest(iris)
rpartTest(iris)
myDecisionTreeTest(iris)
```

Results:

```
> ctreeTest(iris)
ctree % 94.66667
> rpartTest(iris)
rpart % 93.33333
> myDecisionTreeTest(iris)
My Decision Tree % 94.66667
```

Comments:

decisionTree : Ağacı oluşturduğum fonksiyon

myTreePredict : Tek bir satır data için Prediction yaptığım recursive fonksiyon

testPredict : Test datasını test ettiğim Prediction fonksiyonu

ctreeTest : ctree'yi test ettiğim fonksiyon

rpartTest : rpart'ı test ettiğim fonksiyon

myDecisionTreeTest : Kendi implement ettiğim tree'nin test fonksiyonu

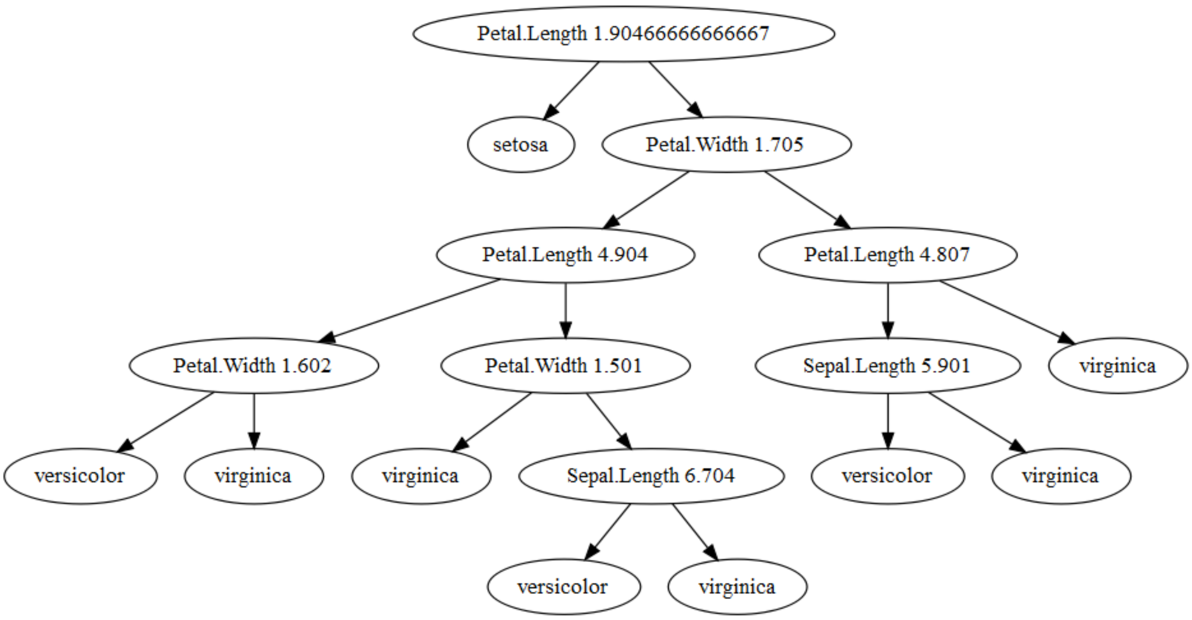
part3 : Bagging algoritmasını implement ettiğim fonksiyon

baggingTest : Bagging test fonksiyonu

Bu fonksiyonları part 2 ve part 3'te de kullanıyorum o yüzden tekrar yazmıyorum.

Yazdığım tree implementasyonunda çıkan sonuçlar ctree ve rpart testlerinden çıkan sonuçlar ile çok yakın. Ayrıca her denemede %90'ın üzerinde başarı elde ettim.

Oluşan Ağaç



Part 2:

Code:

```
###############################################################################
################################## Part 2 #################################
```

```
################################################################################
PruningVal <- 0.63
myDecisionTreeTest(iris)
```

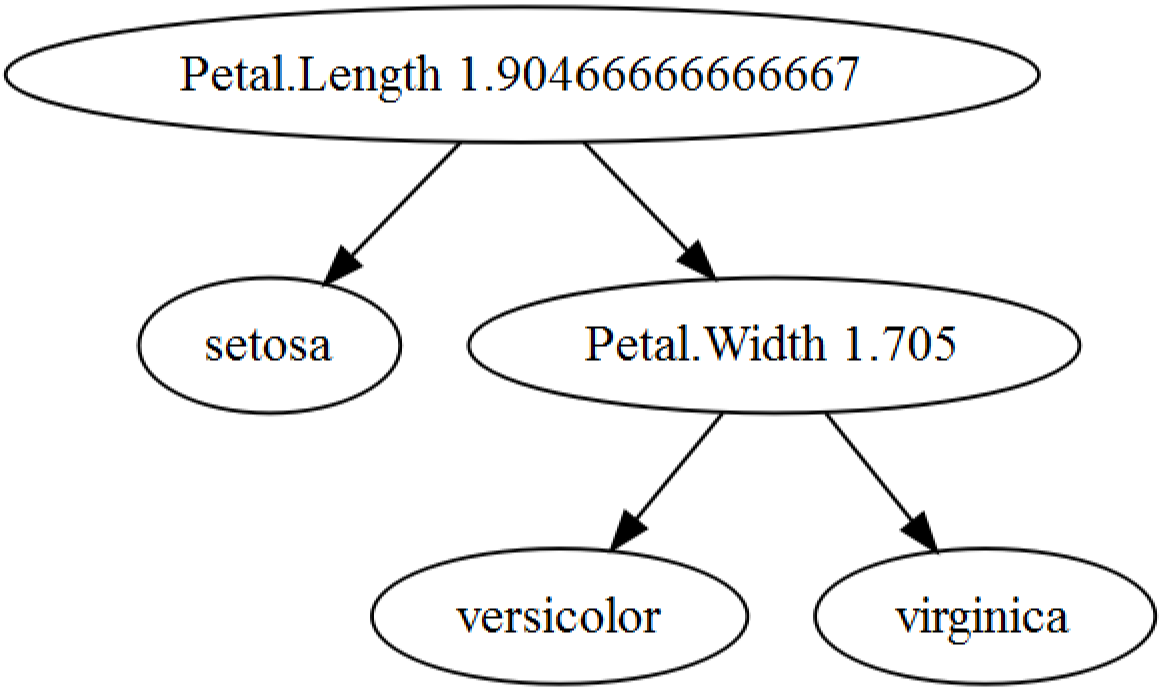
Results:

```
> PruningVal <- 0.63
> rpartTest(iris)
rpart % 21.33333
> myDecisionTreeTest(iris)
My Decision Tree % 95.33333
```

Comments:

Prepruning yaparak ağacı entropinin 0.63’den küçük eşit olduğu yerlerden kestim.

Pruning Sonucu Oluşan Ağaç



Part 3:

Code:

```
################################################################################
################################# Part 3 ################################
################################################################################
PruningVal <- 0
```

```
baggingTest(iris)
```

Results:

```
> baggingTest(iris)
Bagging Test % 92.66667
```

Comments:

Train datsının %63.2 sini sabit tutarak geri kalan kısmını attım. Daha sonar boş kısmı train datasından rastgele satırlar seçerek doldurdum (Duplicate). Bu şekilde iris datasını sürekli rastgele karıştırarak N tane data oluşturdum. (N i 5 seçtim) Daha sonra bu dataları kullanarak tree'ler oluşturdum. Oluşturduğum tree'leri predict ederek sonuçları hesapladım. Bu işlemleri k-cross validation kullanarak 10 kere tekrarladım ve ortalama bir performans değeri hesapladım.