Automated Classification using Decision Trees

Krishna Nadiminti

March 19, 2016

Objective

To develop a program that, generates a Decision Tree via ID3 Algorithm with the given collection of Data.

Introduction

A decision tree is a rooted, directed tree akin to a flowchart. Each internal node corresponds to a partitioning decision, and each leaf node is mapped to a class label prediction. To classify a data item, we imagine the data item to be traversing the tree, beginning at the root. Each internal node is programmed with a splitting rule, which partitions the domain of one (or more) of the data's attributes. Based on the splitting rule, the data item is sent forward to one of the node's children.This testing and forwarding is repeated until the data item reaches a leaf node.

Background- ID3 Algorithm

ID3 is a nonincremental algorithm, meaning it derives its classes from a fixed set of training instances. An incremental algorithm revises the current concept definition, if necessary, with a new sample. The classes created by ID3 are inductive, that is, given a small set of training instances, the specific classes created by ID3 are expected to work for all future instances. The distribution of the unknowns must be the same as the test cases. Induction classes cannot be proven to work in every case since they may classify an infinite number of instances

Programming Approach

R-programming has been specifically used for generating the Decision trees using ID3 Algorithm.The training sets are read as dataframe into the global environment.

```r
dataTraining <- read.csv("C:/Krishna/ML-622/Project3/contact-lenses.data", sep="")
#dataTraining <- read.csv("C:/Krishna/ML-622/Project3/fishing.data", sep="")

dataTraining
```

```
##    A.            Age Prescription Astigmatism TearRate Oracle
## 1  D:          young        myope          no  reduced   none
## 2  D:          young        myope          no   normal   soft
## 3  D:          young        myope         yes  reduced   none
## 4  D:          young        myope         yes   normal   hard
## 5  D:          young hypermetrope          no  reduced   none
## 6  D:          young hypermetrope          no   normal   soft
## 7  D:          young hypermetrope         yes  reduced   none
## 8  D:          young hypermetrope         yes   normal   hard
## 9  D: pre-presbyopic        myope          no  reduced   none
## 10 D: pre-presbyopic        myope          no   normal   soft
## 11 D: pre-presbyopic        myope         yes  reduced   none
## 12 D: pre-presbyopic        myope         yes   normal   hard
## 13 D: pre-presbyopic hypermetrope          no  reduced   none
## 14 D: pre-presbyopic hypermetrope          no   normal   soft
## 15 D: pre-presbyopic hypermetrope         yes  reduced   none
## 16 D: pre-presbyopic hypermetrope         yes   normal   none
## 17 D:     presbyopic        myope          no  reduced   none
## 18 D:     presbyopic        myope          no   normal   none
## 19 D:     presbyopic        myope         yes  reduced   none
## 20 D:     presbyopic        myope         yes   normal   hard
## 21 D:     presbyopic hypermetrope          no  reduced   none
## 22 D:     presbyopic hypermetrope          no   normal   soft
## 23 D:     presbyopic hypermetrope         yes  reduced   none
## 24 D:     presbyopic hypermetrope         yes   normal   none
```

Taining data been read into DataFrame

```
uniqueOracle<- unique(dataTraining$Oracle)
```

Unique Oracle data has been factored

```
Entropy<-function()
{

distCountVec<-vector(mode = "numeric")
probdist<-vector(mode = "numeric")

for(j in 1:length(uniqueOracle))
{
  sumDistinct=0
  for(i in 1:nrow(dataTraining))
  {

    if(dataTraining$Oracle[i]==uniqueOracle[j])

      sumDistinct= sumDistinct+1
  }
  distCount=sumDistinct


distCountVec<-append(distCountVec,distCount)
probdist<- append(probdist,(distCount/nrow(dataTraining)))

}



  probVec<- sapply(probdist, function(x) (x)*log2(x))
  sum <- sum(probVec)

    return(-sum)
}
```

Global Entropy has been Calculated.

```
getDataSet<- function(i,j,dataTraining){

  dataTraining[dataTraining[j] == toString(unique(dataTraining[[j]])[i]) ,]

}
```

Method to get the dataset on every factorization.

```
atomicEntropy<- function(atomicData,entropySum){

  dataSetProb<-apply(atomicData,2,function(x) x/sum(x))

  dataSetlog<-apply(dataSetProb,2,function(x) (x)*log2(x))
  dataSetlog<- replace(dataSetlog,is.nan(dataSetlog),0)
  dataSetEntropy<- apply(dataSetlog,2,function(x) -sum(x))

  return(dataSetEntropy)
}
```

Method to get the atomic entropy at ech level to get the Information Gain at each level to find the node.

```
EntropyS<- Entropy()

infoGainVector<- vector(mode = "numeric")

for(i in 2:(ncol(dataTraining)-1)){

  nodedataSetTemp<-table(dataTraining$Oracle,dataTraining[[i]])
  nodemargintemp<- margin.table(nodedataSetTemp, 2)
  entropySum<- vector(mode = "numeric")
  nodedataSetEntropy<-atomicEntropy(nodedataSetTemp)
  nodedataSetEntropy<-replace(nodedataSetEntropy,is.nan(nodedataSetEntropy),0)

  sum=0
  for(i in 1:length(nodedataSetEntropy)){
    sum= sum+ nodedataSetEntropy[i]*(nodemargintemp[i]/sum(nodemargintemp))
  }

 nodeGainInfo<- EntropyS-sum
 #print(GainInfo)
 infoGainVector<- append(infoGainVector,nodeGainInfo)


}

NodeColumn<- which.max(infoGainVector)+1

nodeTabletemp<- table(dataTraining$Oracle,dataTraining[[NodeColumn]])
nodeSetEntropy<-atomicEntropy(nodeTabletemp)
nodeSetEntropy<-replace(nodeSetEntropy,is.nan(nodeSetEntropy),0)

dataTraining[[1]]<-NULL # First Column has been deleted
NodeColumn<- NodeColumn-1
```

Root Node has been calculated

```
AlgorithmDecision<- function(node,newdataTraining,nodeSetEntropy){

  ##restricts the data when no attributes are left
  if(ncol(newdataTraining)<=1){
    print(unique(newdataTraining$Oracle))
  }
  else
    {
  ##Looping to all the attributes of the rootNode

  for(i in 1:length(unique(newdataTraining[[node]]))){

    nodeEntropy<-EntropyS
    truncData<-getDataSet(i,node,newdataTraining)
    # Factored Data has been generated

    print(names(truncData)[node])
    print(unique(truncData[[node]]))
    truncData[[node]]<-NULL
    newDatatrain<-truncData

    if(length(unique(newDatatrain$Oracle))<=1){
     print(unique(newDatatrain$Oracle))
      return
    }
   else{
     print("Next Node........")
    GainVector<- vector(mode = "numeric")
    #Looping to all the columns of the factored dataset
    for(i in 1:(ncol(newDatatrain)-1)){

      dataSetTemp<-table(newDatatrain$Oracle,newDatatrain[[i]])
      margintemp<- margin.table(dataSetTemp, 2)
      entropySum<- vector(mode = "numeric")
      dataSetEntropy<-atomicEntropy(dataSetTemp)
      dataSetEntropy<-replace(dataSetEntropy,is.nan(dataSetEntropy),0)
      sum=0
      for(i in 1:length(dataSetEntropy)){
        sum= sum+ dataSetEntropy[i]*(margintemp[i]/sum(margintemp))
      }

      GainInfo<- nodeEntropy-sum
      GainVector<- append(GainVector,GainInfo)

    }
        if((length(GainVector)==0)||(length(GainVector)==1)){
          print(unique(newDatatrain[[1]]))
           print(unique(newDatatrain$Oracle))
        return
    }
    else{
    nodeElement<- which.max(GainVector)
    nodeTabletemp<- table(newDatatrain$Oracle,newDatatrain[[i]])
    nodeSetEntropy<-atomicEntropy(nodeTabletemp)
    nodeSetEntropy<-replace(nodeSetEntropy,is.nan(nodeSetEntropy),0)
    AlgorithmDecision(nodeElement,newDatatrain,nodeSetEntropy)
    }
   }
  }
}
}
```

The Method AlgorithmDecision has slightly modified version of ID3 Algorithm to get the decision trees. The Information gain has been slightly modified, where Entropy(S) has been the global entropy. The global entropy has been chosen to make computation easier, because teh information gain is calculated to find the maximum amon the calculated nodes to find the next node.

Results

```
AlgorithmDecision(NodeColumn,dataTraining,nodeSetEntropy)
```

```
## [1] "TearRate"
## [1] reduced
## Levels: normal reduced
## [1] none
## Levels: hard none soft
## [1] "TearRate"
## [1] normal
## Levels: normal reduced
## [1] "Next Node........"
## [1] "Astigmatism"
## [1] no
## Levels: no yes
## [1] "Next Node........"
## [1] "Age"
## [1] young
## Levels: pre-presbyopic presbyopic young
## [1] soft
## Levels: hard none soft
## [1] "Age"
## [1] pre-presbyopic
## Levels: pre-presbyopic presbyopic young
## [1] soft
## Levels: hard none soft
## [1] "Age"
## [1] presbyopic
## Levels: pre-presbyopic presbyopic young
## [1] "Next Node........"
## [1] myope        hypermetrope
## Levels: hypermetrope myope
## [1] none soft
## Levels: hard none soft
## [1] "Astigmatism"
## [1] yes
## Levels: no yes
## [1] "Next Node........"
## [1] "Prescription"
## [1] myope
## Levels: hypermetrope myope
## [1] hard
## Levels: hard none soft
## [1] "Prescription"
## [1] hypermetrope
## Levels: hypermetrope myope
## [1] "Next Node........"
## [1] young          pre-presbyopic presbyopic
## Levels: pre-presbyopic presbyopic young
## [1] hard none
## Levels: hard none soft
```

Output for “Fishing Data”

[1] “Forecast” [1] Sunny Levels: Cloudy Rainy Sunny [1] “Next Node is……..” [1] “Wind” [1] Strong Levels: Strong Weak [1] Yes Levels: No Yes [1] “Wind” [1] Weak Levels: Strong Weak [1] “Next Node is……..” [1] “Water” [1] Warm Levels: Cold Moderate Warm [1] No Levels: No Yes [1] “Water” [1] Cold Levels: Cold Moderate Warm [1] No Levels: No Yes [1] “Water” [1] Moderate Levels: Cold Moderate Warm [1] Yes Levels: No Yes [1] “Forecast” [1] Cloudy Levels: Cloudy Rainy Sunny [1] Yes Levels: No Yes [1] “Forecast” [1] Rainy Levels: Cloudy Rainy Sunny [1] “Next Node is……..” [1] “Air” [1] Warm Levels: Cool Warm [1] “Next Node is……..” [1] “Wind” [1] Strong Levels: Strong Weak [1] Yes Levels: No Yes [1] “Wind” [1] Weak Levels: Strong Weak [1] No Levels: No Yes [1] “Air” [1] Cool Levels: Cool Warm [1] No Levels: No Yes

Discussion

The program would have been much effective if the display content would have been produced instead of displaying nodes. However, the analysis of the nodes have been made and produced as decision tree down

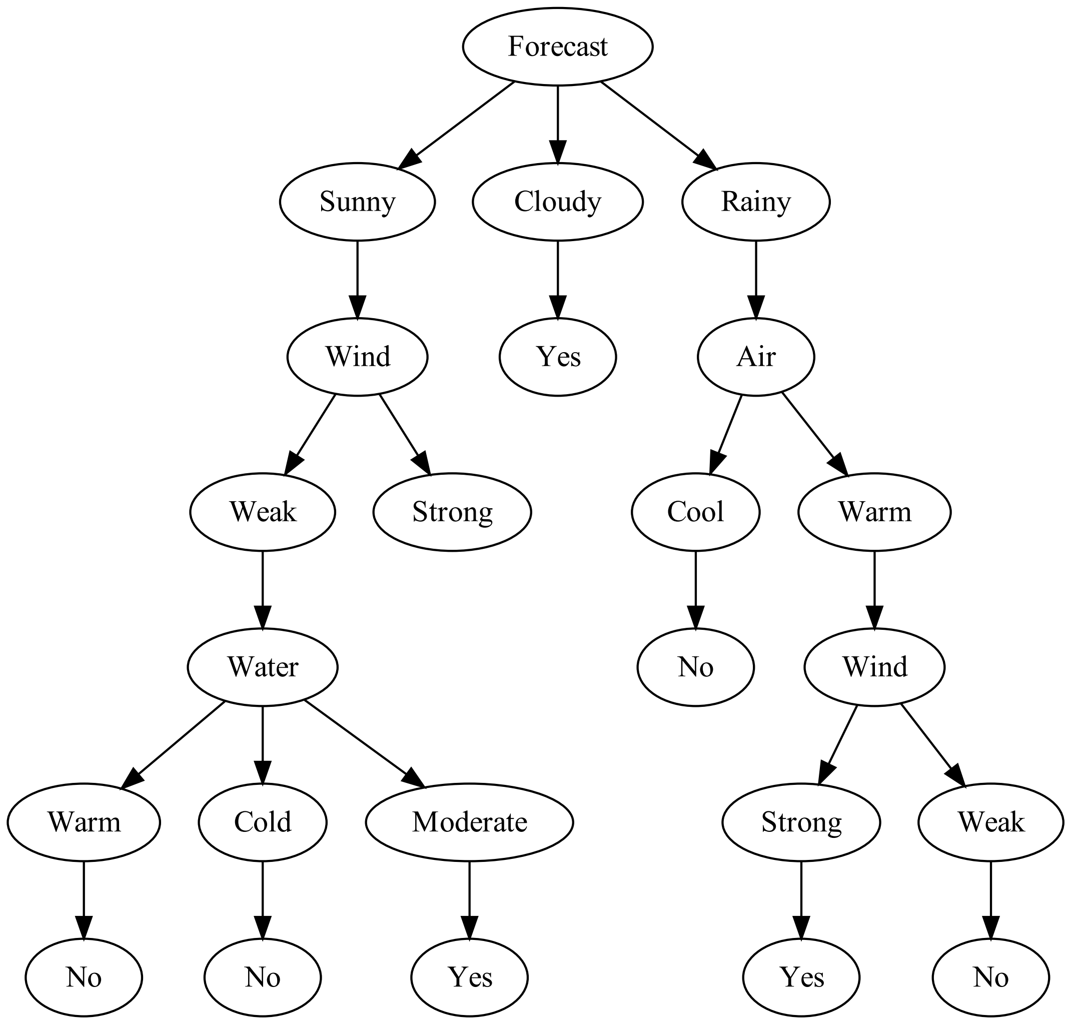
```
library(data.tree)
```

```
## Warning: package 'data.tree' was built under R version 3.2.3
```

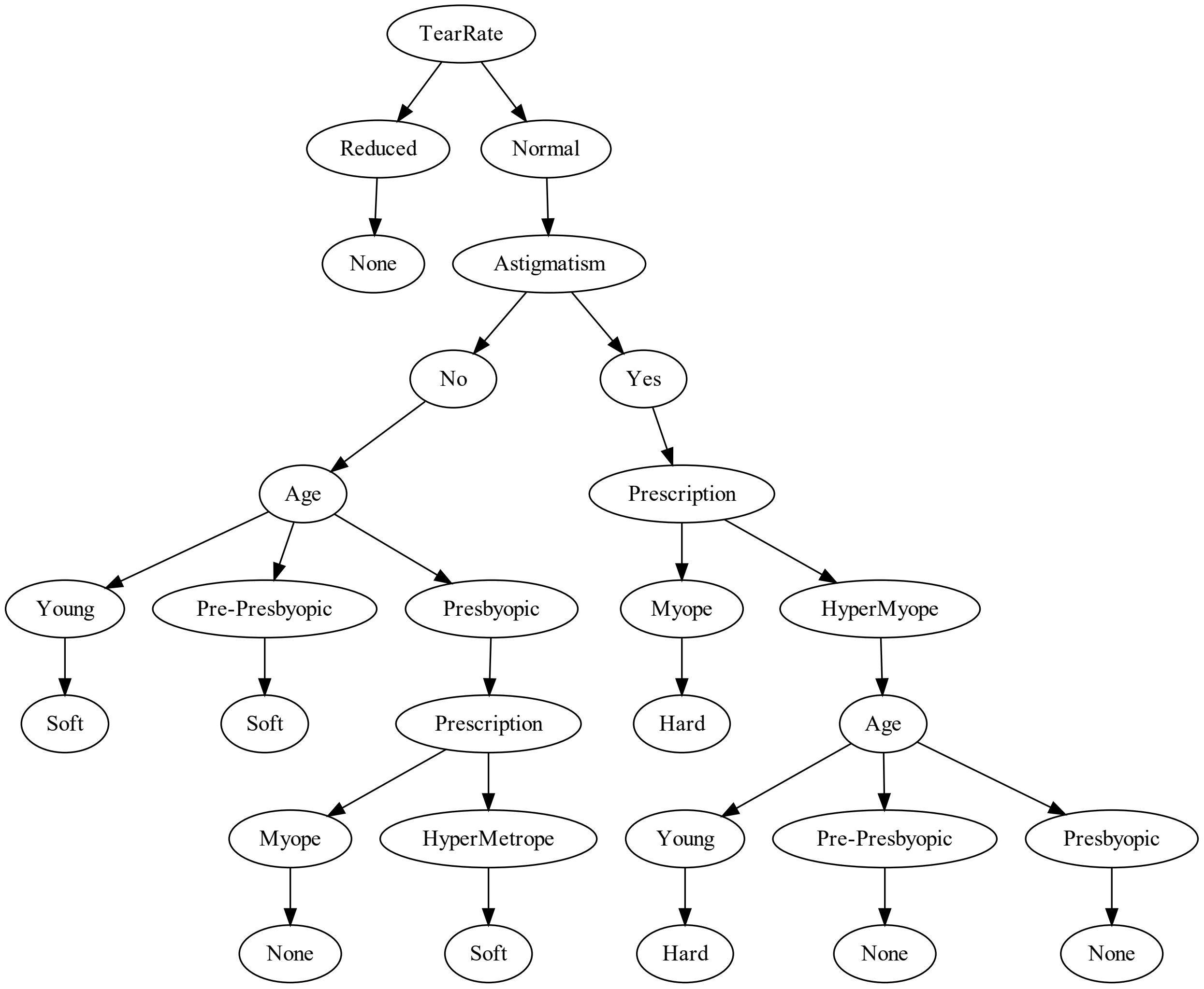
```
ForeCast<- Node$new("Forecast")
Sunny<-ForeCast$AddChild("Sunny")
Cloudy<- ForeCast$AddChild("Cloudy")
Rainy<-ForeCast$AddChild("Rainy")
SunnyWind<-Sunny$AddChild("Wind")
SunnyWeakWind<- SunnyWind$AddChild("Weak")
SunnystrongWind<- SunnyWind$AddChild("Strong")
SunnyWeakWater<- SunnyWeakWind$AddChild("Water")
SunnyWeakWater$AddChild("Warm")$AddChild("No")
SunnyWeakWater$AddChild("Cold")$AddChild("No")
SunnyWeakWater$AddChild("Moderate")$AddChild("Yes")
Cloudy$AddChild("Yes")
RainyAir<- Rainy$AddChild("Air")
RainyAirCool<- RainyAir$AddChild("Cool")
RainyAirCool$AddChild("No")
RainyAirWarm<- RainyAir$AddChild("Warm")
RainyAirWind<- RainyAirWarm$AddChild("Wind")
RainyAirWind$AddChild("Strong")$AddChild("Yes")
RainyAirWind$AddChild("Weak")$AddChild("No")
print(ForeCast)
```

```
##                        levelName
## 1  Forecast
## 2   ¦--Sunny
## 3   ¦   °--Wind
## 4   ¦       ¦--Weak
## 5   ¦       ¦   °--Water
## 6   ¦       ¦       ¦--Warm
## 7   ¦       ¦       ¦   °--No
## 8   ¦       ¦       ¦--Cold
## 9   ¦       ¦       ¦   °--No
## 10  ¦       ¦       °--Moderate
## 11  ¦       ¦           °--Yes
## 12  ¦       °--Strong
## 13  ¦--Cloudy
## 14  ¦   °--Yes
## 15  °--Rainy
## 16      °--Air
## 17          ¦--Cool
## 18          ¦   °--No
## 19          °--Warm
## 20              °--Wind
## 21                  ¦--Strong
## 22                  ¦   °--Yes
## 23                  °--Weak
## 24                      °--No
```

```
plot(ForeCast)
```



```
TearRate<- Node$new("TearRate")
Reduced<- TearRate$AddChild("Reduced")
Normal<- TearRate$AddChild("Normal")
Reduced$AddChild("None")
NormAst<-Normal$AddChild("Astigmatism")
NormAstno<- NormAst$AddChild("No")
NormAstyes<- NormAst$AddChild("Yes")
NormAstnoAge<- NormAstno$AddChild("Age")
noAgeYng<- NormAstnoAge$AddChild("Young")$AddChild("Soft")
noAgepre<- NormAstnoAge$AddChild("Pre-Presbyopic")$AddChild("Soft")
noAgepres<- NormAstnoAge$AddChild("Presbyopic")
noAgepresSight<- noAgepres$AddChild("Prescription")
noAgepresSight$AddChild("Myope")$AddChild("None")
noAgepresSight$AddChild("HyperMetrope")$AddChild("Soft")
YesPres<- NormAstyes$AddChild("Prescription")
YesPresMyope<- YesPres$AddChild("Myope")$AddChild("Hard")
YespresHyper<- YesPres$AddChild("HyperMyope")
YesAge<- YespresHyper$AddChild("Age")
YesAge$AddChild("Young")$AddChild("Hard")
YesAge$AddChild("Pre-Presbyopic")$AddChild("None")
YesAge$AddChild("Presbyopic")$AddChild("None")
plot(TearRate)
```



Conclusion

The correctness fishing data has been verified as a known example. However, Unknown example has been Contact-lenses data, Tear Rate being the root node has been making some sense, as the contact lenses initially required for such conditions.The down to leaf decision tree looks pretty much interesting and should make sense in predicting the data.