Homework 2 - Pattern Recognition

Dzvezdana Arsovska

April 8, 2017

Decision trees are produced by algorithms that identify various ways of splitting a data set into branch-like segments. They are a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). A decision node has two or more branches (for example: Sunny, Overcast and Rainy). Leaf node (for example: Play(Yes or No)) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor is called root node.

One of the algorithms for building decision trees is ID3, which uses Entropy and Information Gain to construct a decision tree. Entropy is used to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one. The information gain is based on the decrease in entropy after a dataset is split on an attribute. When we are constructing a decision tree we are finding attribute that returns the highest information gain (i.e., the most homogeneous branches). ID3 is typically used in the machine learning and natural language processing.

The disadvantages of the ID3 algorithm are:

- Data may be over-fitted or over-classified, if a small sample is tested.
- Only one attribute at a time is tested for making a decision.
- Does not handle missing values.

C4.5 is an improvement of the ID3 algorithm. Some of the advantages of C4.5 over ID3 are:

- it can handle both continuous and discrete attributes (In order to handle continuous attributes, C4.5 creates a threshold and then splits the list into those whose attribute value is above the threshold and those that are less than or equal to it.).
- it can handle data with missing attribute values.
- it can handle attributes with differing costs.
- it prunes the tree after creation.

1 Task 1

Build a ID3 for the given data. The data consists of four attributes (features) 'Outlook, Temperature, Humidity and Wind' and the decision made, 'play' or 'not play' ball. Use 'train.csv' to generate the ID3. Use 'test.csv' to predict the outcome.

Answer:

Because I have previously implemented some parts of this algorithm using R I decided to use this language for this task. For some reason I had issues to load the .csv tables (which usually is done using the read.csv command in R). I tried to fix the issue using OpenRefine and change the formats but it did not help. Since we have a small data set at the end I decided to manually insert the test and train data.

The ID3 steps are:

1. If the data set is pure, then construct a leaf having the name of the class.

   else

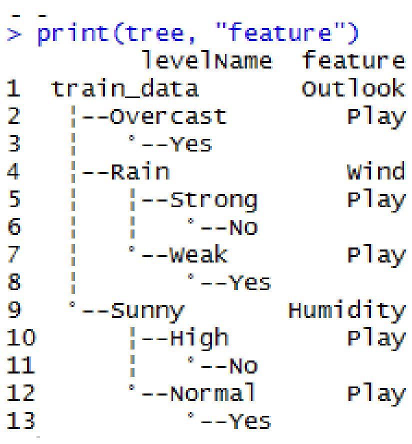
2. Choose the feature with the highest information gain (In the beginning the gain is Gain(S,Outlook) = 0.246; Gain(S, Temperature) = 0.029; Gain(S, Humidity) = 0.151; Gain(S, Wind) = 0.048); so the root node is Outlook). Since Temperature has very low information gain we can expect that it will not be included in the final tree.

3. For each value of that feature:

   - take the subset of the data-set having that feature value
   - construct a child node having the name of that feature value
   - call the algorithm recursively on the child node and the subset

For creating data frames and node like objects used for visualizing the tree the data.table and data.tree libraries are used.

The results are shown in the following images:



```
> print(tree, "feature")
         levelName  feature
1  train_data       Outlook
2   ¦--Overcast        Play
3   ¦   °--Yes
4   ¦--Rain            Wind
5   ¦   ¦--Strong      Play
6   ¦   ¦   °--No
7   ¦   °--Weak        Play
8   ¦       °--Yes
9   °--Sunny       Humidity
10      ¦--High        Play
11      ¦   °--No
12      °--Normal      Play
13          °--Yes
```

Figure 1: Output in R

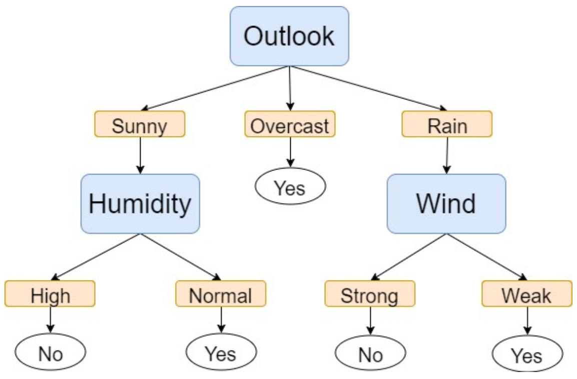
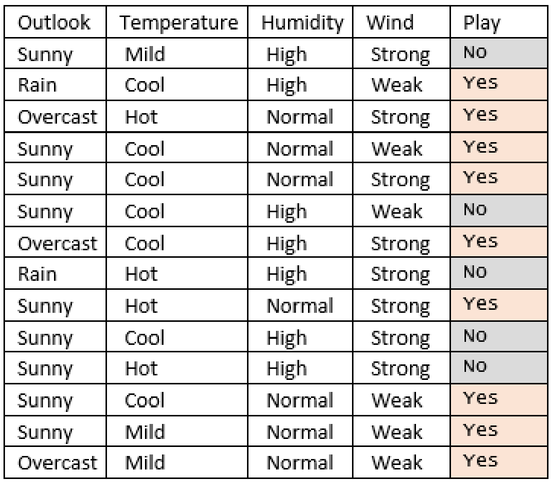


Figure 2: Diagram of the tree

The set of rules generated by the tree are:

- **Rule1**: IF (Outlook = Sunny) AND (Humidity = High) THEN (Play = No)
- **Rule2**: IF (Outlook = Sunny) AND (Humidity = Normal) THEN (Play = Yes)
- **Rule3**: IF (Outlook = Overcast) THEN (Play = Yes)
- **Rule4**: IF (Outlook = Rain) AND (Wind = Strong) THEN (Play = No)
- **Rule5**: IF (Outlook = Rain) AND (Wind = Weak) THEN (Play = Yes)



Outlook	Temperature	Humidity	Wind	Play
Sunny	Mild	High	Strong	No
Rain	Cool	High	Weak	Yes
Overcast	Hot	Normal	Strong	Yes
Sunny	Cool	Normal	Weak	Yes
Sunny	Cool	Normal	Strong	Yes
Sunny	Cool	High	Weak	No
Overcast	Cool	High	Strong	Yes
Rain	Hot	High	Strong	No
Sunny	Hot	Normal	Strong	Yes
Sunny	Cool	High	Strong	No
Sunny	Hot	High	Strong	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Weak	Yes
Overcast	Mild	Normal	Weak	Yes

Figure 3: Predictions

```r
rm(list=ls())
library(data.tree)

# Insert the features

Outlook <- c('Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', '
    Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast',
              'Overcast', 'Rain')

Temperature <- c('Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild'
    ,'Mild','Mild','Hot','Mild')

Humidity <- c('High','High','High','High','Normal','Normal','Normal','High','Normal
    ','Normal','Normal','High','Normal','High')

Wind <- c('Weak','Strong','Weak','Weak','Weak','Strong','Strong','Weak','Weak','
    Weak','Strong','Strong','Weak','Strong')

Play <- c('No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes'
    ,'No')

# Bind all features of the test data
train_data <- data.frame(Outlook,Temperature,Humidity,Wind,Play)


Outlook <- c('Sunny','Rain','Overcast','Sunny','Sunny','Sunny','Overcast','Rain','
    Sunny','Sunny','Sunny','Sunny','Sunny','Overcast')

Temperature <- c('Mild','Cool','Hot','Cool','Cool','Cool','Cool','Hot','Hot','Cool'
    ,'Hot','Cool','Mild','Mild')

Humidity <- c('High','High','Normal','Normal','Normal','High','High','High','Normal
    ','High','High','Normal','Normal','Normal')
```

```
Wind <- c('Strong','Weak','Strong','Weak','Strong','Weak','Strong','Strong','Strong
    ','Strong','Strong','Weak','Weak','Weak')

# Bind all features of the train data
test <- data.frame(Outlook,Temperature,Humidity,Wind)

# Check if one of the features is completely pure (contains only one class, in our
    case only Yes or No).
PureNode <- function(data) {
  length(unique(data[,5])) == 1}

# Calculate the entropy
Entropy <- function(S) {
  res <- S/sum(S) * log2(S/sum(S))
  res[S == 0] <- 0
  -sum(res)}

#Calculate the Information gain
Information_Gain <- function(feat) {
  entropyBefore <- Entropy(colSums(feat))
  s <- rowSums(feat)
  entropyAfter <- sum (s/sum(s) * apply(feat, MARGIN = 1, FUN = Entropy ))
  Information_Gain <- entropyBefore - entropyAfter
  return (Information_Gain)
}

# Choose attribute with the largest information gain as the decision node, divide
    the dataset by its branches and repeat the same process on every branch.

TrainID3 <- function(node, data) {

  node$obsCount <- nrow(data)
  # if the dataset is pure then create a leaf with the name of the pure feature
  # if the dataset is pure STOP
  if (PureNode(data)) {
    child <- node$AddChild(unique(data[,ncol(data)]))
    node$feature <- tail(names(data), 1)
    child$obsCount <- nrow(data)
    child$feature <- ''
  } else {
    # if data set is not pure calculate the information gain
    ig <- sapply(colnames(data)[-ncol(data)],
                 function(x) Information_Gain(
                   table(data[,x], data[,ncol(data)])
                 )
    )
    # Chose the feature with the highest information gain
    # If more than one feature have the same information gain, then take the first
        one
    feature <- names(which.max(ig))
    node$feature <- feature
    #Create a subset from a feature
    childObs <- split(data[ ,names(data) != feature, drop = FALSE],
                      data[ ,feature],
                      drop = TRUE)

    for(i in 1:length(childObs)) {
      #construct a child for the feture
      child <- node$AddChild(names(childObs)[i])

      #call the algorithm recursively
      TrainID3(child, childObs[[i]])
    }
  }
}

# Build a tree using the train data and print it
tree <- Node$new("train_data")
TrainID3(tree, train_data)
print(tree, "feature", "obsCount")

# Prediction
Predict <- function(tree, features) {
  if (tree$children[[1]]$isLeaf) return (tree$children[[1]]$name)
  child <- tree$children[[features[[tree$feature]]]]
  return ( Predict(child, features))}

# Make a prediction for each row in the test data and print the resulst
for(i in 1:nrow(test)){
val = Predict(tree,test[i,])
print(val)}
```