Decision Trees (DT)

Ruxandra Stoean

Additional bibliography

- Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen, Classification and Regression Trees, Chapman and Hall/CRC, 1984

- Clinton Sheppard, Tree-based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting, 2017

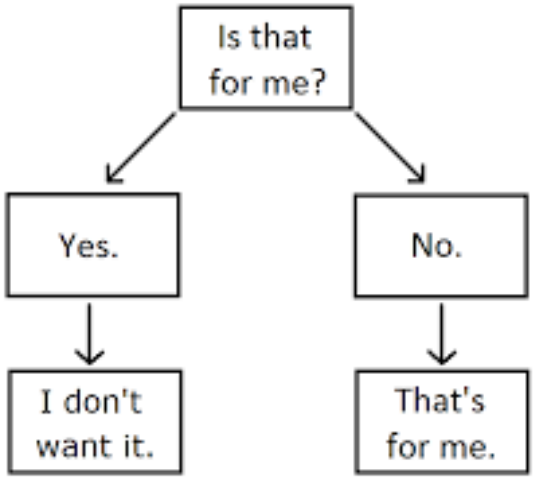
- Lior Rokach, Oded Maimon, Data Mining with Decision Trees: Theory and Applications (Second Edition), World Scientific, 2014

Introduction

- Very appreciated technique :
  - Possibility to visualize the model learning the data
  - White-box learning, as opposed to support vector machines, neural networks (black box)
  - Data type independent

- The learning procedure:
  - Training data are recursively partitioned
  - Process stops when homogeneous terminal nodes are obtained (classes for classification or thresholds for regression)

My Cat's Decision-Making Tree.



Algorithms

- ID3 – categorical features; greedy split by largest information gain for classification; pruning after tree completion
- C4.5 – in addition, also for continuous features, by partitioning into discrete intervals; output converted to IF-THEN rules, rule accuracy gives order of application
- C5.0 – increased accuracy; smaller rule sets
- CART – in addition, also for regression; no rule sets

DT structure

- Every internal node expresses a test in relation to an attribute of the problem, which partitions the objects.
- Attributes: discrete or continuous
  - If continuous, a threshold is used for partitioning
- Every edge signifies the partition resulting after the application of the test that corresponds to that node
- Terminal (leaf) nodes represent the dominant class/group that resulted on that branch

Node split criterion

- The measure of an efficient partitioning of the data in the current node, on the basis of a pair (attribute, value), is given by the following criteria:
- Classification
  - **Gini index -> minimum**
    - CART
  - **Information gain -> maximum**
    - i.e., **entropy -> minimum**
    - ID3, C4.5, C5.0
- Regression
  - **Sum of squared errors**

Algorithm

Root node = the entire data set

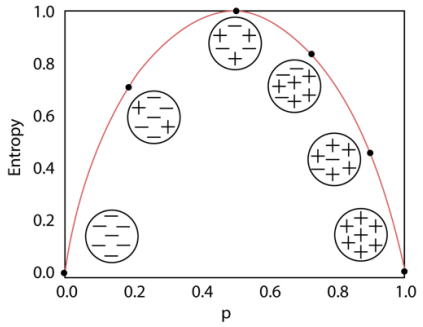
Repeat

1. For each (attribute) split, calculate the efficiency value of each child node.
2. Calculate the efficiency of the split as the weighted average of the efficiency of the children nodes.
3. Select the split with the highest increase in efficiency

Until homogeneous nodes are achieved.

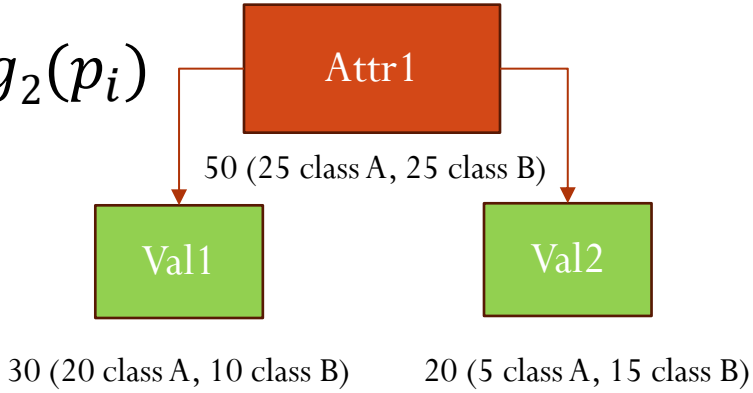
Information gain

$$E = \sum_{k=1}^{C} -p_i log_2(p_i)$$



- Entropy
  - The degree of disorder, uncertainty or randomness within a system or data; result in the interval [0, 1]
  - Also called impurity
  - p_i - the proportion of data points of class *i* in a node (probability of class *i*)
    - If all data in a node belong to one class (purity), then entropy is 0
    - If distributed evenly among two classes ($p_i = 0.5$), then entropy is 1.
- Information Gain
  - Measures the reduction of entropy through the split
  - Entropy of the split (attribute) is WeightedAvg(Entropy(children nodes))
  - IG = Entropy(parent node) – Entropy(split node)
  - Select the split (attribute) with the highest IG

$$E = \sum_{k=1}^{C} -p_i log_2(p_i)$$



Example IG

- Entropy(Root)=-(25/50)log_2(25/50)-(25/50)log_2(25/50) = 1
- Entropy(Val1)=-(20/30)log_2(20/30)-(10/30)log_2(10/30) = 0.917
- Entropy(Val2)=-(5/20)log_2(5/20)-(15/20)log_2(15/20) = 0.811
- Entropy(Attr1)=(30/50)*0.917+(20/50)*0.811 = 0.87
- IG(Attr1)=Entropy(Root)-Entropy(Attr1) = 0.13
- Calculate IG for the other attributes, pick the one with highest IG

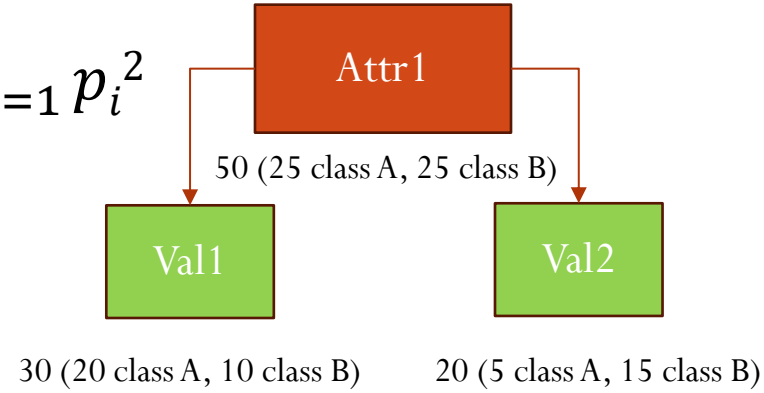
Gini impurity

$$G = \sum_{k=1}^{C} p_i(1 - p_i)$$
$$= \sum_{k=1}^{C} p_i - \sum_{k=1}^{C} {p_i}^2$$
$$= 1 - \sum_{k=1}^{C} {p_i}^2$$

- The degree of non-homogeneity in the data; the result in the interval [0, 0.5]
  - p_i - again probability of class *i*
    - If all data in a node belong to one class (purity), then Gini is 0
    - If distributed evenly among two classes ($p_i = 0.5$), Gini is 0.5
- Aims for the decrease in impurity through the split
- Gini impurity of the split (attribute) is WeightedAvg(Gini(children nodes))
- Select the split (attribute) with the lowest Gini index (highest decrease in impurity)

$$G = 1 - \sum_{k=1}^{C} p_i^2$$

Example Gini



- Gini(Root) = 1 – ((0.5)^2+(0.5)^2) = 0.5

- Gini(Val1) = 1 – ((20/30)^2+(10/30)^2) = 4/9
- Gini(Val2) = 1 - ((5/20)^2+(15/20)^2) = 3/8
- Gini(Attr1) = (30/50)*(4/9)+(20/50)*(3/8) = 0.283

- Impurity decrease = 0.5 – 0.283 = 0.217
- Calculate Gini for the other attributes, pick the one with lowest Gini (highest impurity decrease)

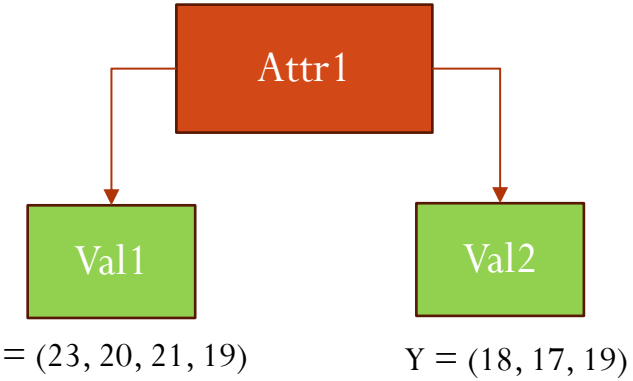
Node split criterion in regression

- **Sum of squared errors** at each (attribute) split
- Aims for error reduction through the split
- Select the split (attribute) with the lowest sum of squared errors
- $\bar{y}$ - mean of outputs

$$E = \sum^{groups} \sum_{k=1}^{N_g} (y_i - \bar{y})^2$$

Example SSE

$$E = \sum^{groups} \sum_{k=1}^{N_g} (y_i - \bar{y})^2$$



- $\bar{y} = 19.57$
- E(Root) = $(23-19.57)^2+(20-19.57)^2+(21-19.57)^2+(19-19.57)^2+(18-19.57)^2+(17-19.57)^2+(19-19.57)^2 = 28.82$
- $\bar{y}_{Val1} = 20.75$
- E(Val1) = $(23-20.75)^2+(20-20.75)^2+(21-20.75)^2+(19-20.75)^2 = 8.74$
- $\bar{y}_{Val2} = 18$
- E(Val2) = $(18-18)^2+(17-18)^2+(19-18)^2 = 2$
- E(Attr1) = 8.74 + 2 = 10.74
- SSE = 28.82-10.74 = 18.08
- Calculate SSE for the other attributes, pick the one with lowest SSE (highest reduction)

Further aspects

- Prior probabilities:
  - The probability of an object belonging to a class can be specified
  - Usually, this parameter is proportional to the number of objects from each class
- Misclassification costs:
  - It can be specified if it is important to more accurately determine one class than another
  - Usually, this parameter is equal to 1 – equal costs for all classes
- Decision trees and ensemble methods do not require feature scaling, as they are based on partitioning the data, where the current feature is not influenced by other features.

Extracting rules from DT

- Rules of the type IF-THEN
- By descending on the edges
- A conjunction between the values (tests) of intermediary nodes up to the leaves is performed for the conditional part
- The conclusion is given by the leaf touched during the descent

Package rpart in R

- Implements classification and regression trees
- It uses the CART algorithm
- Parameters: *parms* = (…)
  - Partitioning criterion (parameter *split*)
    - Based on the Gini index (default) or information gain for classification
    - Anova splitting for regression
  - *list*(*prior*(…)) the list of prior probabilities; implicit – proportional to class cardinal

Method and predict

- If data output is factor, then classification *class* is implicit for the parameter *method*.
  - For regression, *method* = "*anova*"
- It is recommended, however, that the method is specified.

- The function *predict* returns class probabilities by default, if it detects classification
  - Specify type = "*class*" for direct output of predicted class
- Otherwise, assumes regression and returns type *vector*, that can be also specified for safety.

DT for classification in R

```r
library(neuralnet)
library(datasets)
library(rpart)
library(rpart.plot)
library(e1071)
library(caret)

data(iris)
dat <- iris

repeats <- 30
classColumn <- 5
accuracies <- vector(mode="numeric", length = repeats)
index <- 1:nrow(dat)


for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index) / 3))
  testset <- dat[testindex, ]
  trainset <- dat[-testindex, ]

  dt_model <- rpart(Species ~ ., data = trainset, method = "class")
  dt_pred <- predict(dt_model, testset[, -classColumn], type = "class")
  contab <- table(pred = dt_pred, true = testset[, classColumn])
  accuracies[i] <- classAgreement(contab)$diag
}
```

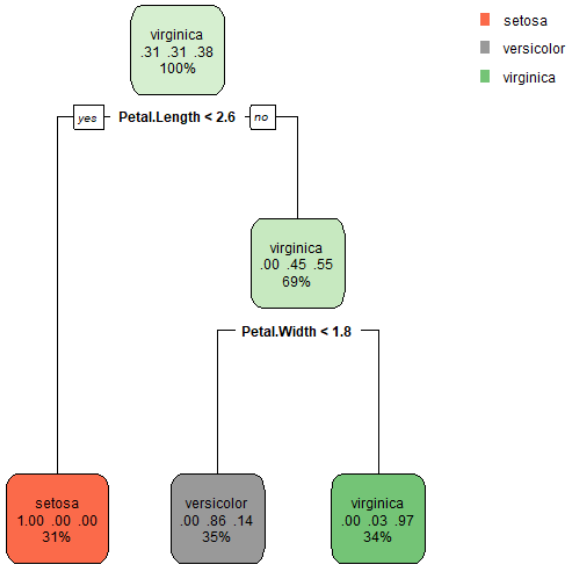
DT classification model

```
36  print(accuracies)
37  print(mean(accuracies))
38  print(sqrt(var(accuracies)))
39  print(summary(accuracies))
40
41  print("Confusion matrix of last run")
42  print(contab)
```

```
 [1] 0.96 0.90 0.94 0.96 0.94 0.94 0.94 0.92 0.94 0.94 0.96 0.94 0.92 0.96 0.92
[16] 0.96 0.96 0.94 0.92 0.96 0.96 0.92 0.92 0.94 0.90 0.98 0.92 0.94 0.92 0.92
[1] 0.938
[1] 0.01989628
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.900   0.920   0.940   0.938   0.960   0.980
[1] "Confusion matrix of last run"
            true
pred         setosa versicolor virginica
  setosa         19          0         0
  versicolor      0         13         3
  virginica       0          1        14
```

Classification tree

```
32    rpart.plot(dt_model)
```



Classification tree in Python sklearn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets

repeats = 30
accuracies = []

ix, iy = datasets.load_iris(return_X_y=True)

for i in range(0, repeats):
    #random generation of training and test, 66%-33%
    ix_train, ix_test, iy_train, iy_test = train_test_split(ix, iy, test_size = 0.33)

    dtm = DecisionTreeClassifier()
    dtm.fit(ix_train,iy_train)

    iy_pred = dtm.predict(ix_test)
    accuracies.append(accuracy_score(iy_test, iy_pred))

print(accuracies)
```

Default split criterion Gini

Results and plot

- sklearn implements an optimized version of the CART algorithm.

```
print("Mean accuracy", np.mean(accuracies))
print("Standard deviation", np.std(accuracies))
confusion_matrix(iy_test,iy_pred)
```

✓ 0.0s

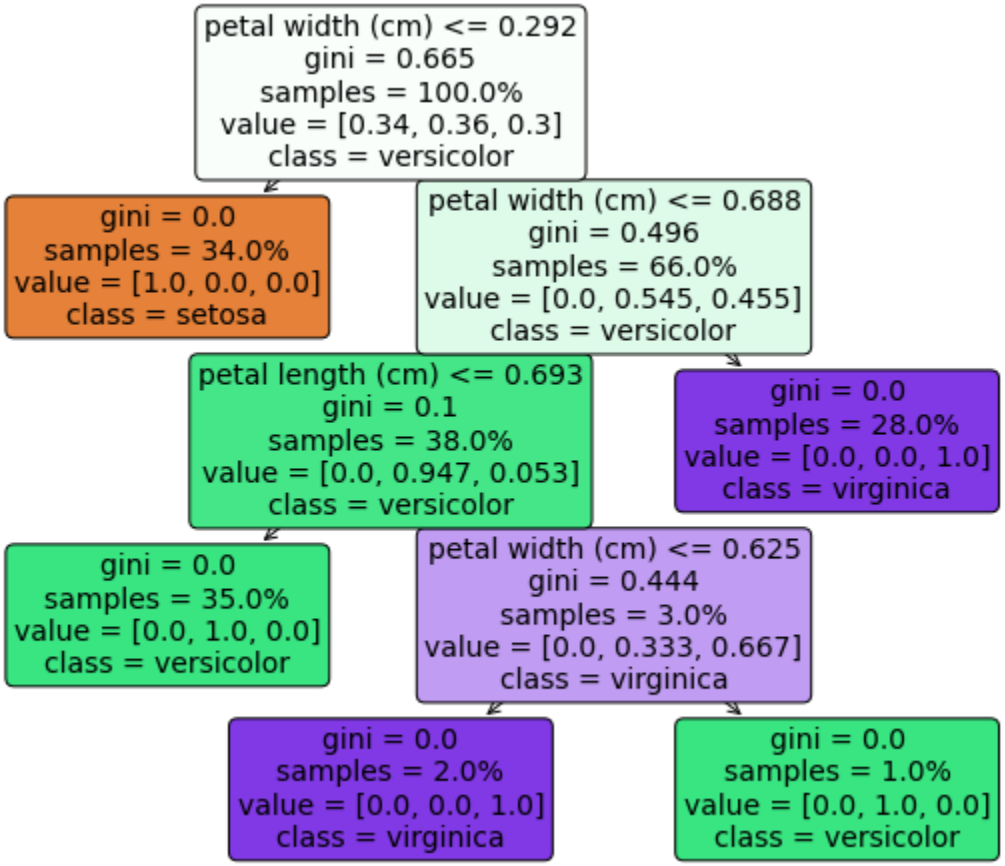
```
Mean accuracy 0.9413333333333335
Standard deviation 0.024184476196289387

array([[19,  0,  0],
       [ 0, 19,  1],
       [ 0,  1, 10]], dtype=int64)
```

```
data = datasets.load_iris(return_X_y=False)
features = data.feature_names
classes = data.target_names
```

Bunch, dictionary-like object; several attributes can be derived

```
from sklearn import tree

plt.figure(figsize=(10, 8))

# for rounding edges rounded = True
# for adding color for class
# for displaying proportions of class samples instead of cardinal
tree.plot_tree(dtm, feature_names=features, class_names=classes,rounded = True, filled = True, proportion = True);
```

Classification tree



The *dtreeviz* library

- A library for decision tree fancy visualization and interpretation
- https://github.com/parrt/dtreeviz
- Install *graphviz* windows package from: https://graphviz.gitlab.io/_pages/Download/Download_windows.html
  - Within installation do not forget to add folders to user path
- Install python *graphviz* and *dtreeviz* packages
  - pip install graphviz
  - pip install dtreeviz

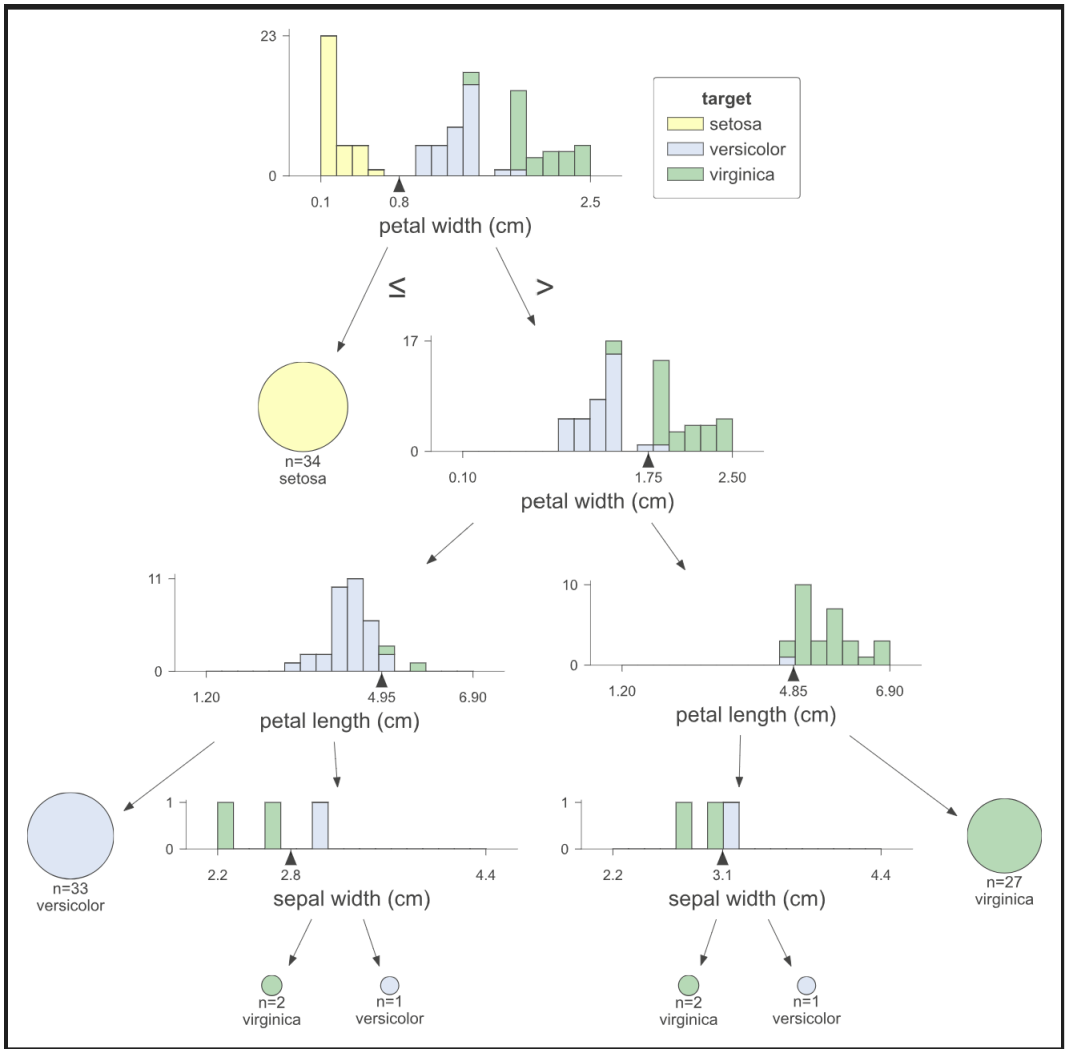
Code for dtree vizualization

```
data = datasets.load_iris(return_X_y=False)
features = data.feature_names
classes = data.target_names
```

Classification/regression target

```
import dtreeviz

dfx = pd.DataFrame(ix_train)
dfx.columns = features
dfy = pd.Series(iy_train)

viz = dtreeviz.model(dtm, X_train=dfx, y_train=dfy, target_name = "target", feature_names = features, class_names = classes)

viz.view(scale = 1.2)
```

Iris result


target
setosa
versicolor
virginica
petal width (cm)
≤
>
n=34
setosa
petal width (cm)
petal length (cm)
petal length (cm)
n=33
versicolor
sepal width (cm)
sepal width (cm)
n=27
virginica
n=2
virginica
n=1
versicolor
n=2
virginica
n=1
versicolor

Regression tree in R

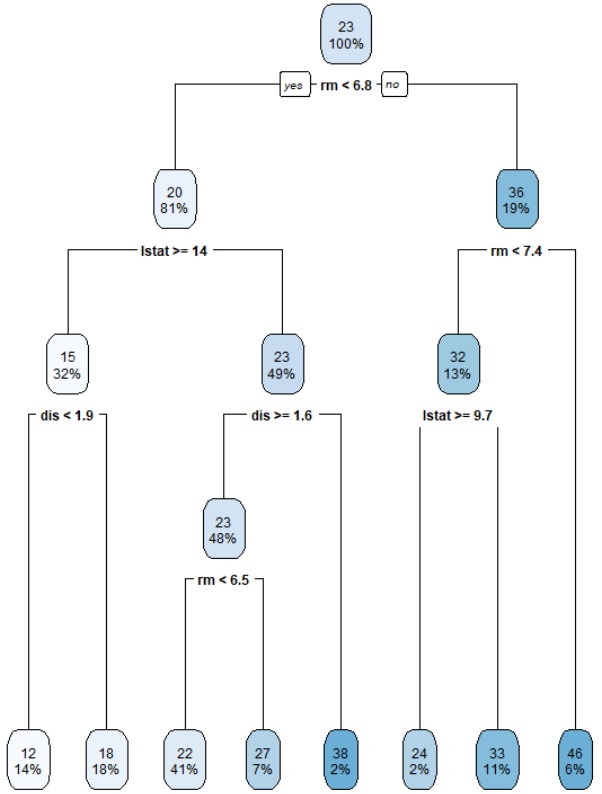
```r
library(rpart)
library(mlbench)
library(caret)
library(rpart.plot)

data(BostonHousing)

repeats <- 30
classColumn <- 14
mses <- vector(mode = "numeric", length = 30)
index <- 1:nrow(BostonHousing)

for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index) / 4))
  testset <- BostonHousing[testindex, ]
  trainset <- BostonHousing[-testindex, ]

  dt_model <- rpart(medv ~ ., data = trainset, method = "anova")
  dt_pred <- predict(dt_model, testset[, -classColumn], type = "vector")
  mses[i] <- mean((dt_pred - testset[, classColumn])^2)
}
```

Prediction results

```
32    rpart.plot(dt_model)
33
34    print(mses)
35    print(mean(mses))
36    print(sqrt(var(mses)))
```

```
 [1] 21.15770 18.82563 27.00894 26.48942 19.16676 20.16366 28.01161 21.23998
 [9] 31.42535 19.87904 16.86186 17.67457 22.51695 25.21261 29.57352 22.76660
[17] 28.94082 24.14389 25.99608 23.92855 28.01958 25.95241 25.12979 16.11526
[25] 24.59842 20.53429 25.13088 21.30044 22.53747 28.87830
[1] 23.63935
[1] 4.008768
```

Regression tree



Regression in sklearn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import datasets

repeats = 30
mses = []
r2s = []

bx, by = datasets.load_boston(return_X_y=True)

for i in range(0, repeats):
    #random generation of training and test, 75-25%
    bx_train, bx_test, by_train, by_test = train_test_split(bx, by, test_size = 0.25)

    # set value for parameter max_depth to keep the tree small
    dtm = DecisionTreeRegressor(max_depth = 3)
    dtm.fit(bx_train,by_train)

    by_pred = dtm.predict(bx_test)
    mses.append(mean_squared_error(by_test, by_pred))
    r2s.append(r2_score(by_test, by_pred))

print(mses)
print(r2s)
```

Default split criterion MSE

Results and plot

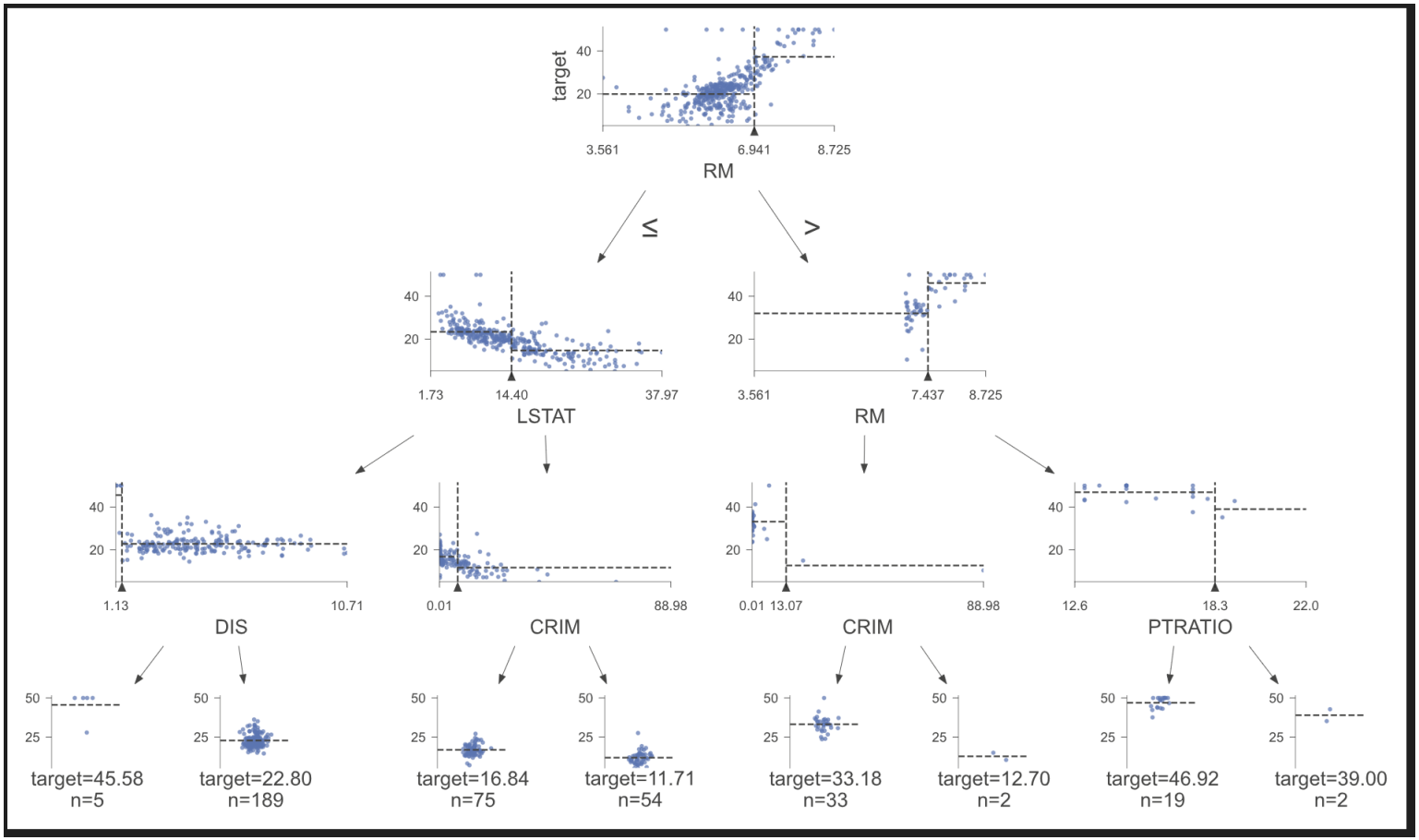
```
print("Mean MSE:", np.mean(mses))
print("Standard deviation:", np.std(mses))
print("R^2:", np.mean(r2s))
```

✓ 0.0s

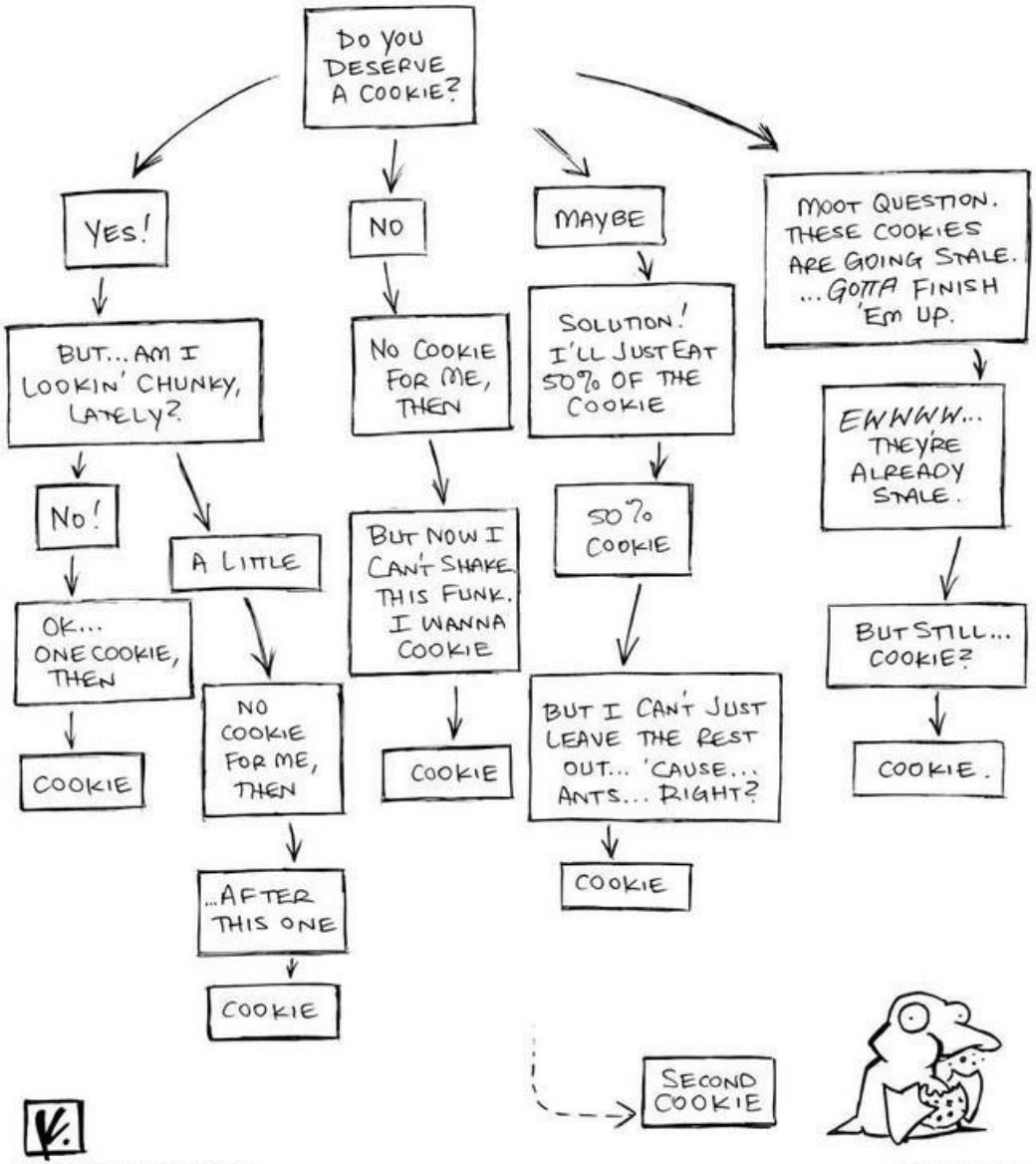
```
Mean MSE: 19.66285039370079
Standard deviation: 7.032766066172616
R^2: 0.7528294750147327
```

```
import dtreeviz

data = datasets.load_boston(return_X_y=False)
features = data.feature_names

dfx = pd.DataFrame(bx_train)
dfx.columns = features
dfy = pd.Series(by_train)

viz = dtreeviz.model(dtm, X_train=dfx, y_train=dfy, target_name = "target", feature_names = features)

viz.view(scale = 1.2)
```

Boston result



Should I Have A Cookie?



WWW.SHELDONCOMICS.COM