Random Forest from Scratch in R

Nikolay Oskolkov, MRG Group Leader, LIOS, Riga, Latvia

R course, 13.02.2026

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Image generated by ChatGPT

Session content

Topics we'll cover in this session:

1) Decision tree cost function: Gini index

2) Computing optimal split via Gini index

3) Getting subset for further splitting

4) Code decision tree from scratch and validate model

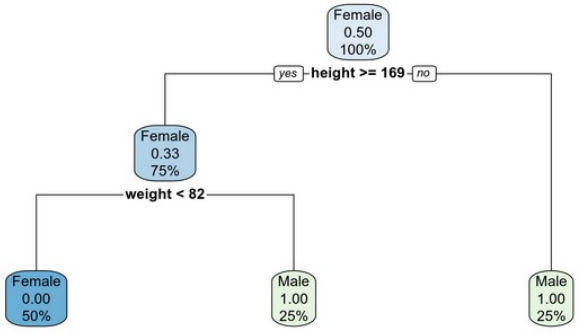
5) Difference between decision tree and Random Forest

Decision tree from scratch in R: problem formulation

```r
X<-data.frame(height=c(183,167,178,171),weight=c(78,73,85,67))
y<-as.factor(c("Female", "Male", "Male", "Female"))
data.frame(X, sex = y)
```

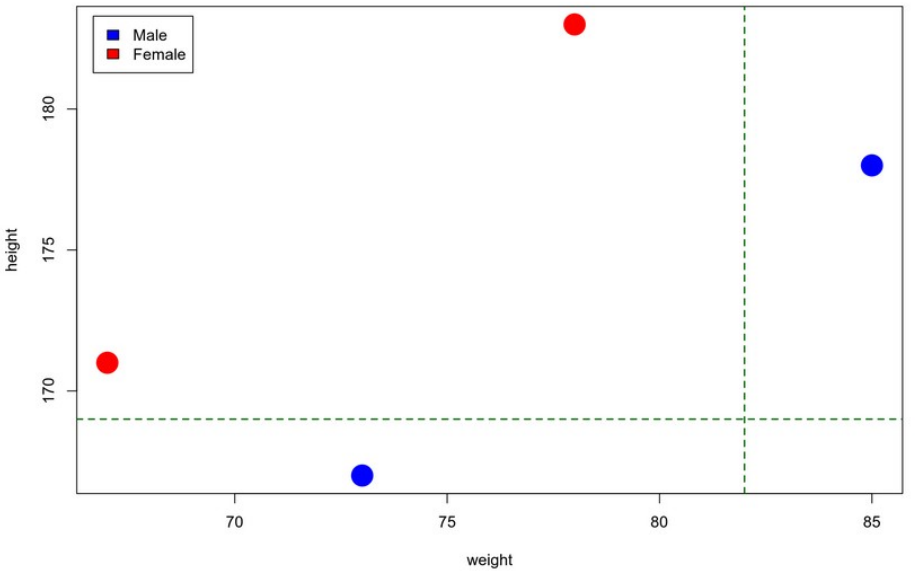
height	weight	sex
183	78	Female
167	73	Male
178	85	Male
171	67	Female

```r
library("rpart"); library("rpart.plot")
fit<-rpart(y~height+weight,data=X,method="class",minsplit=-1)
rpart.plot(fit)
```

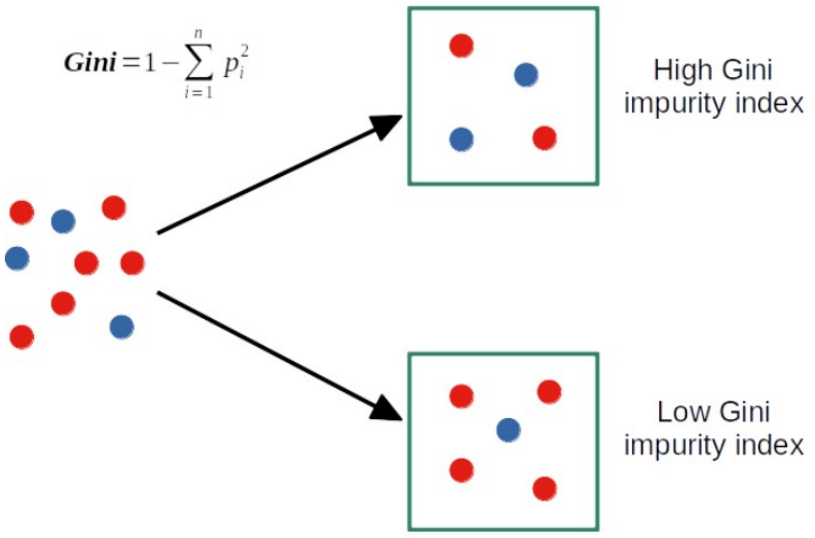


- Let us visualize what the classifier has learnt

```r
color <- c("red", "blue", "blue", "red")
plot(height ~ weight, data = X, col = color, pch = 19, cex = 3)
legend("topleft",c("Male","Female"),fill=c("blue","red"),inset=.02)

abline(h = 169, lty = 2, col = "darkgreen", lwd = 1.5)
abline(v = 82, lty = 2, col = "darkgreen", lwd = 1.5)
```



Decision tree from scratch in R: Gini index and split




```r
gini <- function(x)
{
  return(1 - sum((table(x) / length(x))^2))
}
gini(c(1, 0, 1, 0))
```

```
[1] 0.5
```

```r
gini(c(1, 1, 0, 1, 1))
```

```
[1] 0.32
```

```r
get_best_split <- function(X, y)
{
  mean_gini <- vector(); spl_vals <- vector(); spl_names <- vector()
  for(j in colnames(X)) # for each variable in X data frame
  {
    spl <- vector() # vector of potential split candidates
    sort_X <- X[order(X[, j]), ]; sort_y <- y[order(X[, j])] # sort by variable
    for(i in 1:(dim(X)[1]-1)) # for each observation of variable in X data frame
    {
      spl[i] <- (sort_X[i, j] + sort_X[(i + 1), j]) / 2 # variable consecutive means
      g1_y <- sort_y[sort_X[, j] > spl[i]] # take labels for group above split
      g2_y <- sort_y[sort_X[, j] < spl[i]] # take labels for group below split
      mean_gini <- append(mean_gini, (gini(g1_y) + gini(g2_y))/2) # two groups mean Gini
      spl_vals <- append(spl_vals, spl[i])
      spl_names <- append(spl_names, j)
    }
  }
  min_spl_val <- spl_vals[mean_gini == min(mean_gini)][1] # get best split variable
  min_spl_name <- spl_names[mean_gini == min(mean_gini)][1] # get best split value
  sort_X <- X[order(X[, min_spl_name]), ] # sort X by best split variable
  sort_y <- y[order(X[, min_spl_name])] # sort y by best split variable
  g1_y <- sort_y[sort_X[, min_spl_name] > min_spl_val] # labels above best split
  g2_y <- sort_y[sort_X[, min_spl_name] < min_spl_val] # labels below best split
  if(gini(g1_y) == 0){sex <- paste0("Above: ", as.character(g1_y))}
  else if(gini(g2_y) == 0){sex <- paste0("Below: ", as.character(g2_y))}

  return(list(spl_name = min_spl_name, spl_value = min_spl_val, sex = sex))
}
get_best_split(X, y)
```

```
$spl_name
[1] "height"

$spl_value
[1] 169

$sex
[1] "Below: Male"
```

Decision tree from scratch in R: code implementation

- After we have found the best split, let us check what group we can split further: **get_new_data** function

```r
get_new_data <- function(X, y)
{
  spl_name <- get_best_split(X, y)$spl_name
  spl_val <- get_best_split(X, y)$spl_value

  # Sort X and y by the variable of the best split
  sort_X <- X[order(X[, spl_name]), ]; sort_y <- y[order(X[, spl_name])]

  # get X and y for the first group of samples above the best split value
  g1_y <- sort_y[sort_X[, spl_name] > spl_val]
  g1_X <- sort_X[sort_X[, spl_name] > spl_val,]

  # get X and y for the second group of samples below the best split value
  g2_y <- sort_y[sort_X[, spl_name] < spl_val]
  g2_X <- sort_X[sort_X[, spl_name] < spl_val,]

  # return new data (subset of X and y) for a group with Gini index > 0
  if(gini(g1_y) > 0){return(list(new_X = g1_X, new_y = g1_y))}
  else if(gini(g2_y) > 0){return(list(new_X = g2_X, new_y = g2_y))}
  else{return(0)}
}
get_new_data(X, y)
```

```
$new_X
  height weight
4    171     67
3    178     85
1    183     78

$new_y
[1] Female Male   Female
Levels: Female Male
```

- We can train a decision tree of max_depth = 2

```r
decision_tree <- function(X, y, max_depth = 2)
{
  new_X <- X; new_y <- y
  df <- data.frame(matrix(ncol = 5, nrow = max_depth))
  colnames(df) <- c("spl_num", "spl_name", "sign", "spl_val", "label")
  for(i in 1:max_depth)
  {
    best_split_output <- get_best_split(new_X, new_y)
    sex <- unlist(strsplit(best_split_output$sex,": "))
    df[i, "spl_num"] <- i
    df[i, "spl_name"] <- best_split_output$spl_name
    df[i, "sign"] <- ifelse(sex[1] == "Below", "<", ">")
    df[i, "spl_val"] <- best_split_output$spl_value
    df[i, "label"] <- sex[2]

    new_data_output <- get_new_data(new_X, new_y)
    if(length(new_data_output) != 1)
    {
      new_X <- new_data_output$new_X
      new_y <- new_data_output$new_y
    }
    else
    {
      print("All terminal nodes have perfect purity")
      break
    }
  }
  return(df)
}
decision_tree(X, y)
```

```
[1] "All terminal nodes have perfect purity"
```

spl_num	spl_name	sign	spl_val	label
1	height	<	169.0	Male
2	weight	>	81.5	Male

Decision tree from scratch in R: prediction

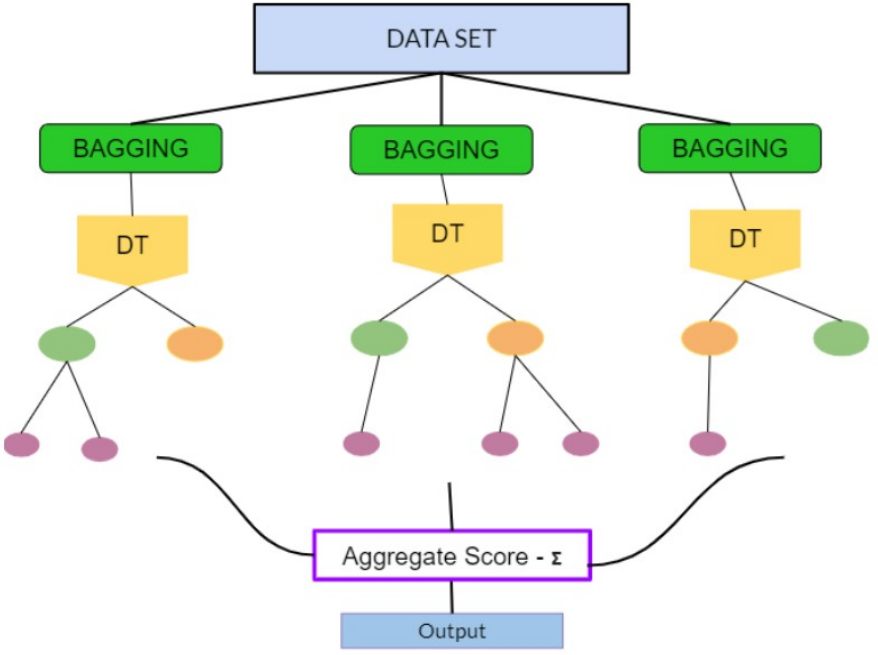
- Finally, after we have trained the decision tree, we can try to make predictions, and check whether we can reconstruct the labels of the data points

```r
predict_decision_tree <- function(X, y)
{
  # Train a decision tree
  t <- decision_tree(X, y, max_depth = 2)

  # Parse the output of decision tree and code it via if, else if and else
  pred_labs <- vector()
  for(i in 1:dim(X)[1])
  {
    if(eval(parse(text=paste0(X[i,t$spl_name[1]],t$sign[1],t$spl_val[1]))))
    {
      pred_labs[i] <- t$label[1]
    }
    else if(eval(parse(text=paste0(X[i,t$spl_name[2]],t$sign[2],t$spl_val[2]))))
    {
      pred_labs[i] <- t$label[2]
    }
    else{pred_labs[i] <- ifelse(t$label[2] == "Male", "Female", "Male")}
  }

  return(cbind(cbind(X, y), pred_labs))
}
predict_decision_tree(X, y)
```

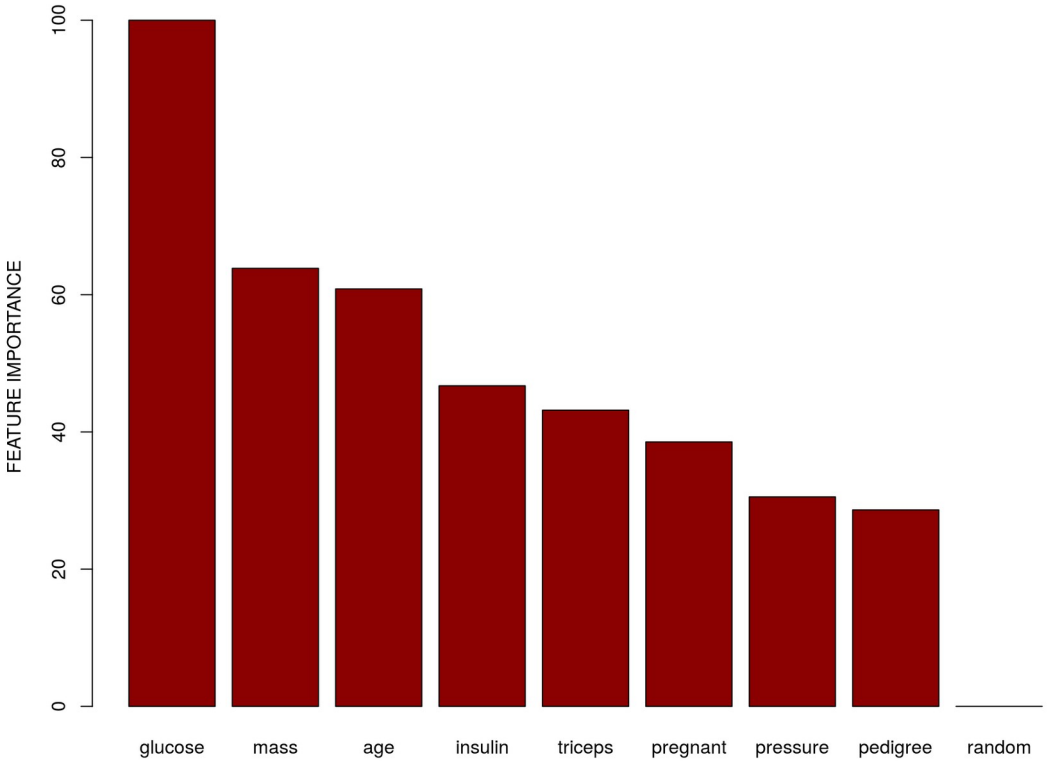
```
[1] "All terminal nodes have perfect purity"
```

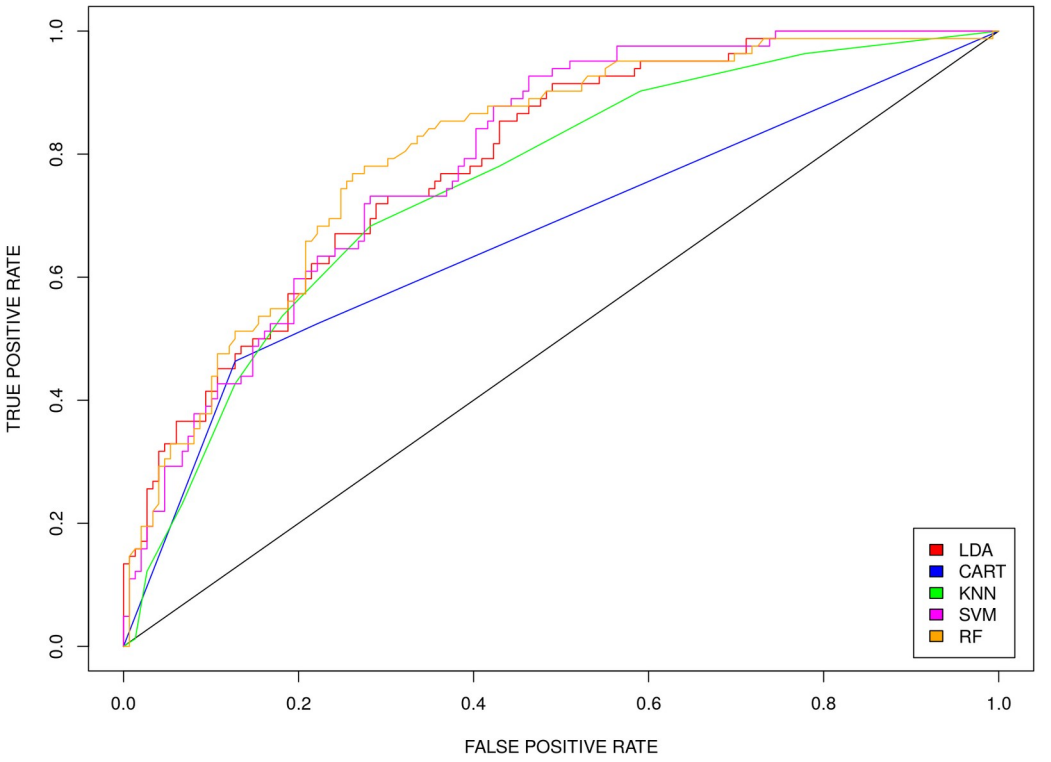
height	weight	y	pred_labs
183	78	Female	Female
167	73	Male	Male
178	85	Male	Male
171	67	Female	Female

- **Random Forest** has two key differences:
  - train multiple decision trees (**bagging**)
  - train trees on **fractions** of input features



Feature importances model comparison via ROC-curves





Session summary

Take home messages of the session:

1) Gini index is the internal decision tree cost function

2) Computing best split is done for each variable separately

3) Figuring out the subset to split further is done via Gini index

4) Ensemble learning is the main power of Random Forest which makes it quite different from regular decision trees

Acknowledgments: LIOS + TARGETWISE

Latvian Institute of Organic Synthesis

2027 National Development Plan

Funded by the European Union