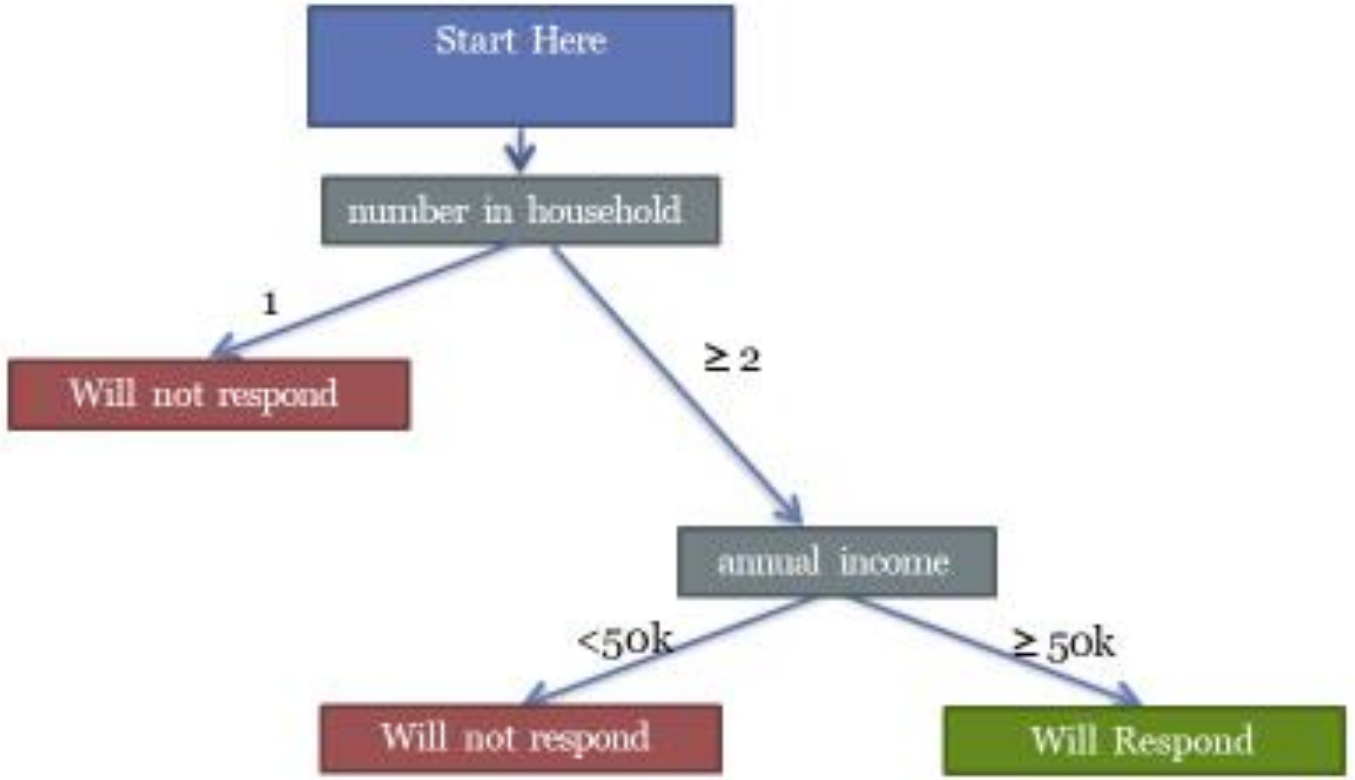
Wank Mountain, Germany

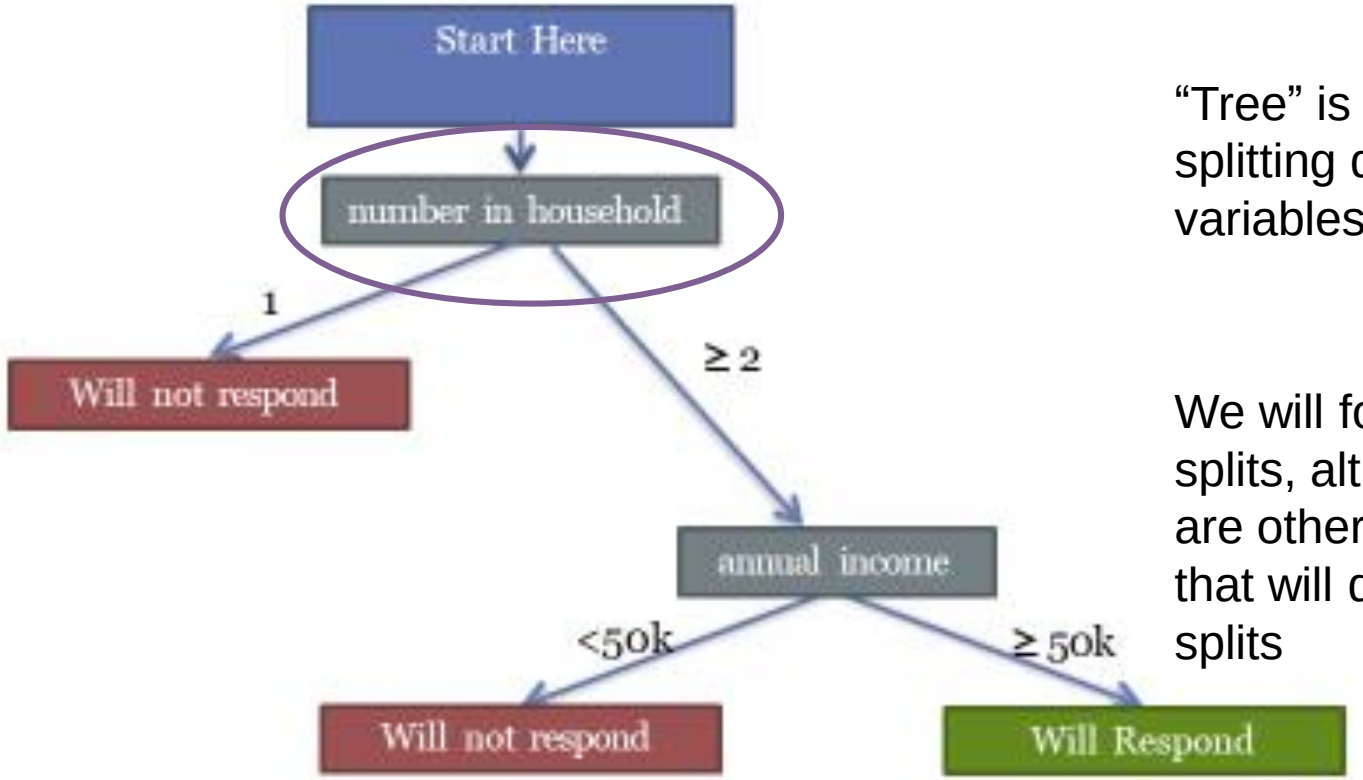
“If you don’t like something, change it. If you can’t change it, change your attitude.” – Maya Angelou

Classification and Regression Trees (CART)

A Decision Tree Model



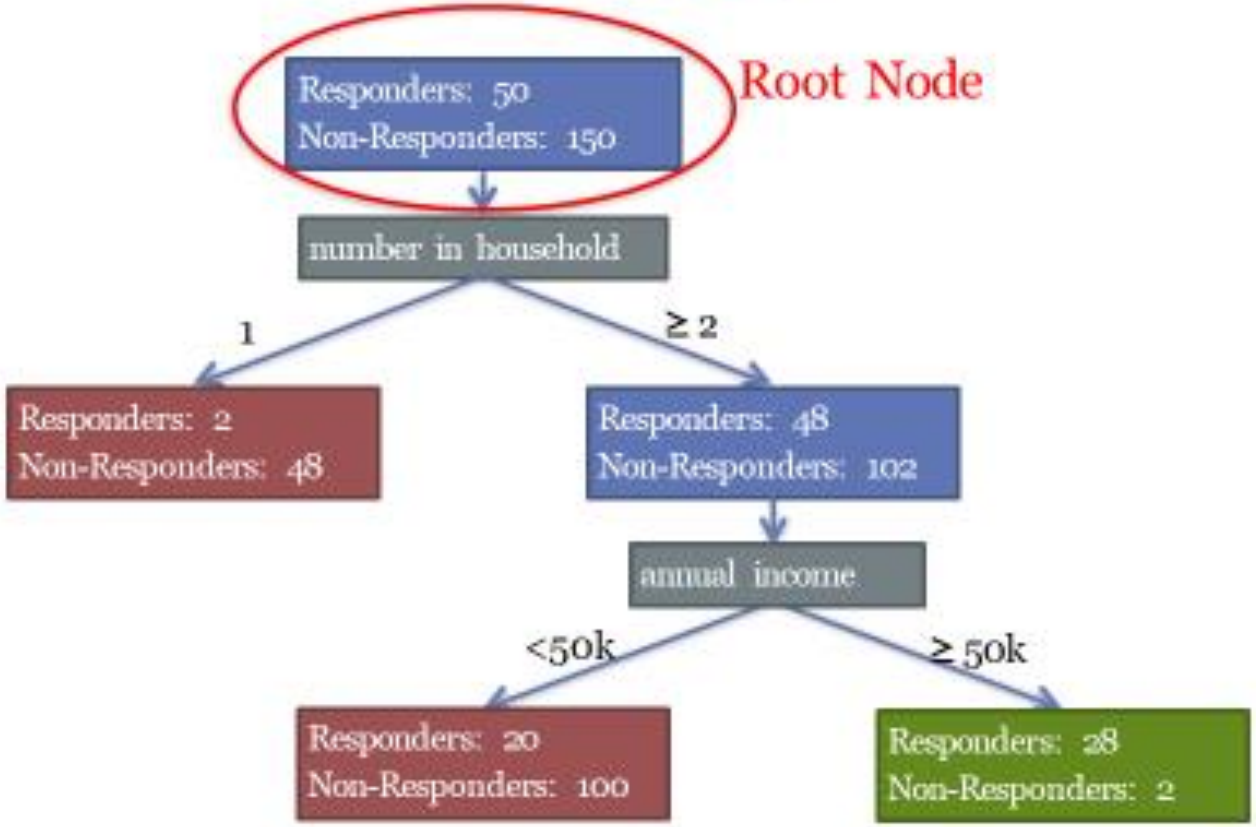
A Decision Tree Model



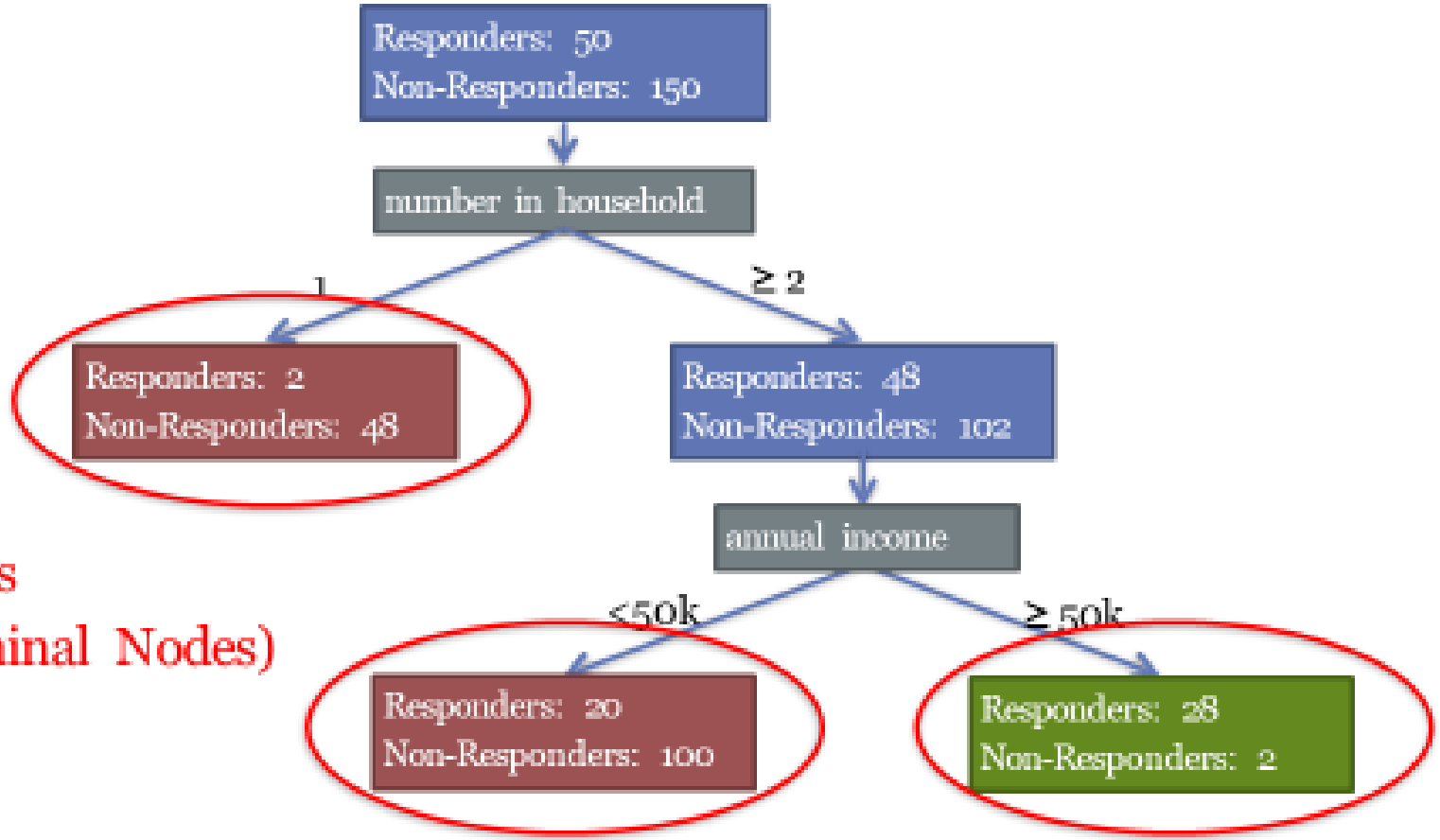
"Tree" is defined by splitting data based on variables in data set

We will focus on binary splits, although there are other algorithms that will do multi-way splits

Decision Tree Model Creation

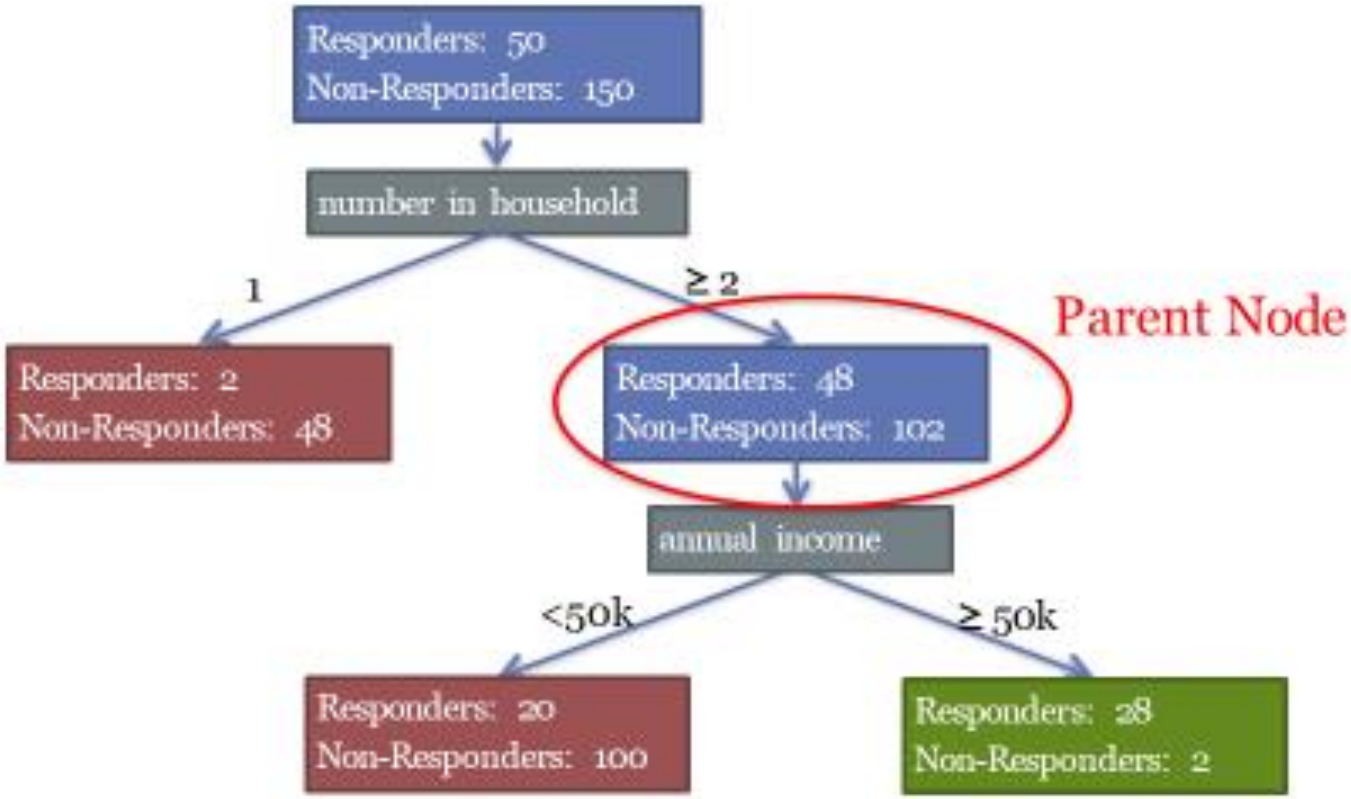


Decision Tree Model Creation

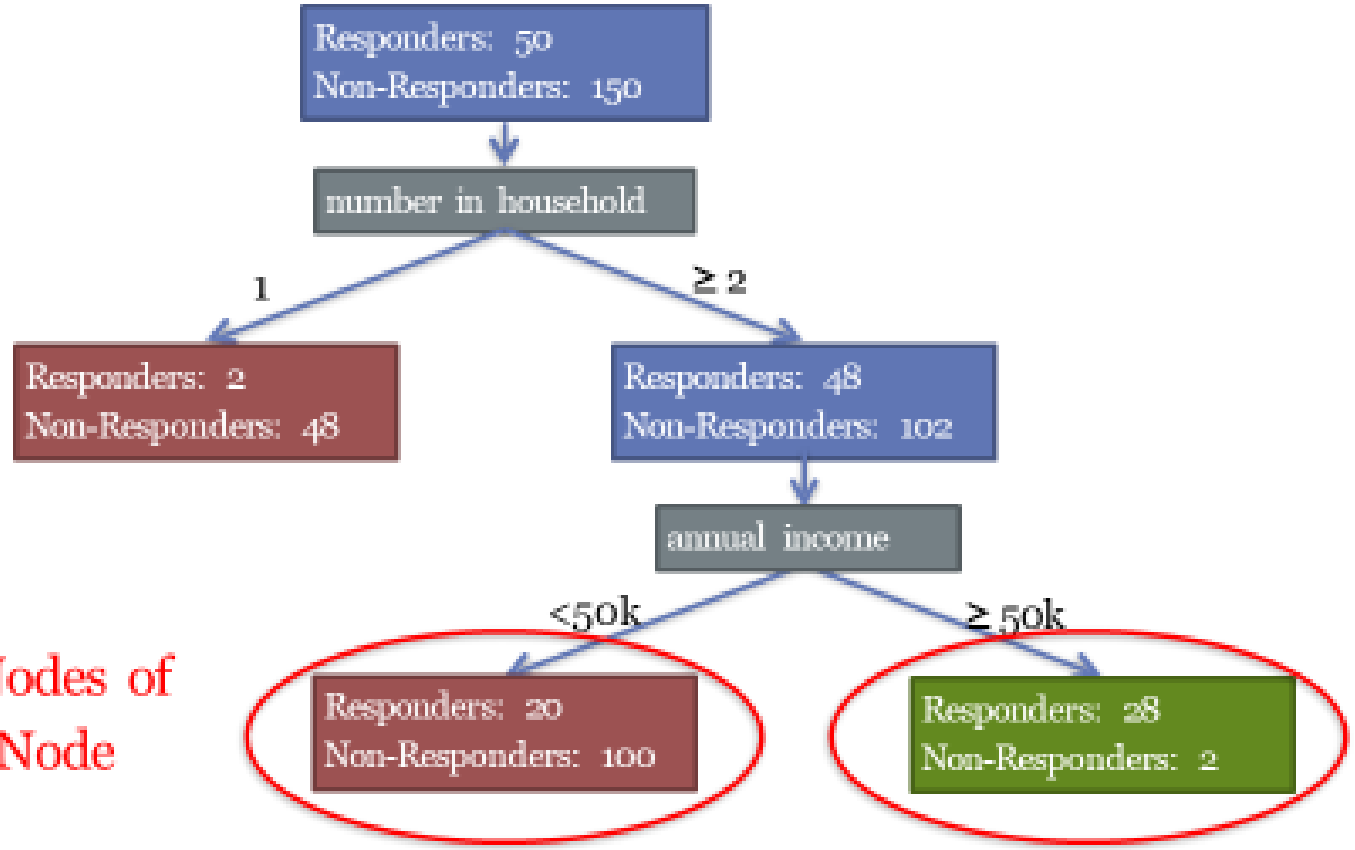


Leaves
(Terminal Nodes)

Decision Tree Model Creation



Decision Tree Model Creation



Child Nodes of Parent Node

Decision Trees

Are INTERPRETABLE!!

Have a list of decisions that predict the outcome (easy to implement)

Allow for nonlinear associations

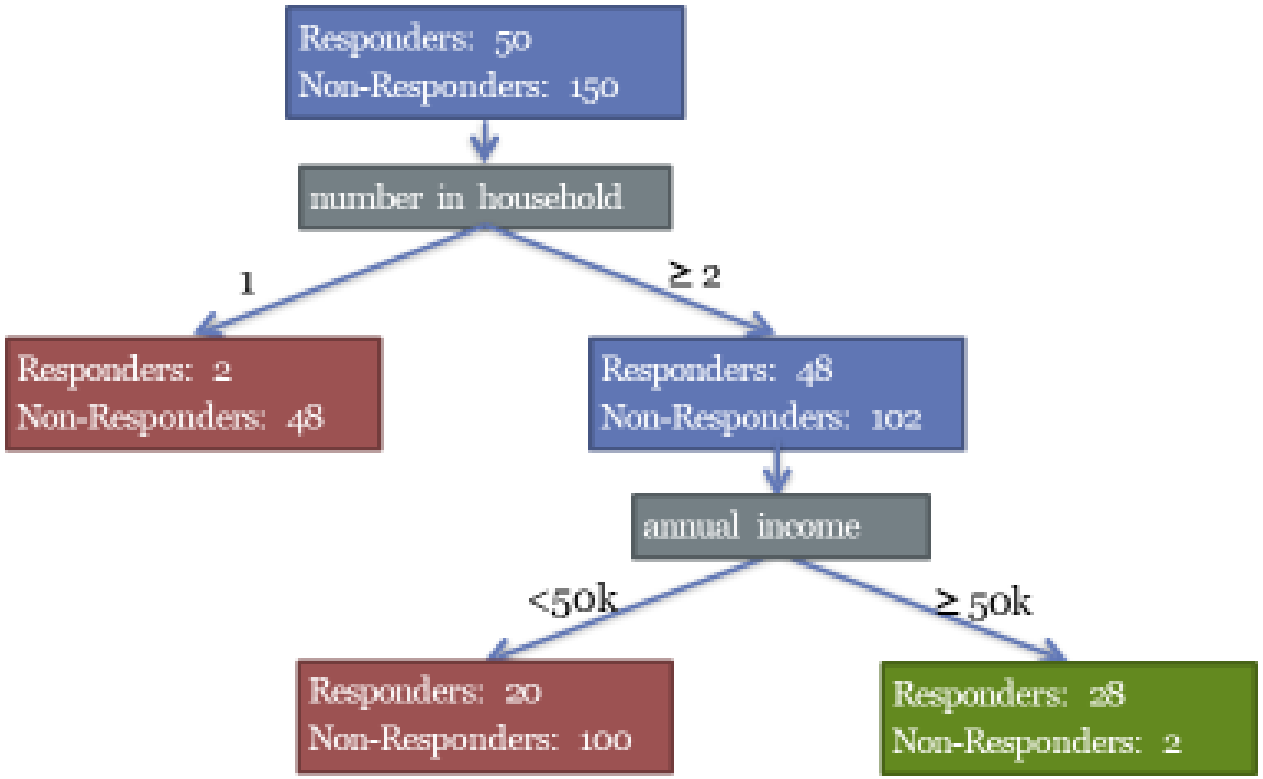
Allow for interactions

Can handle missing values

Are greedy algorithms

Classification Trees (CART method)

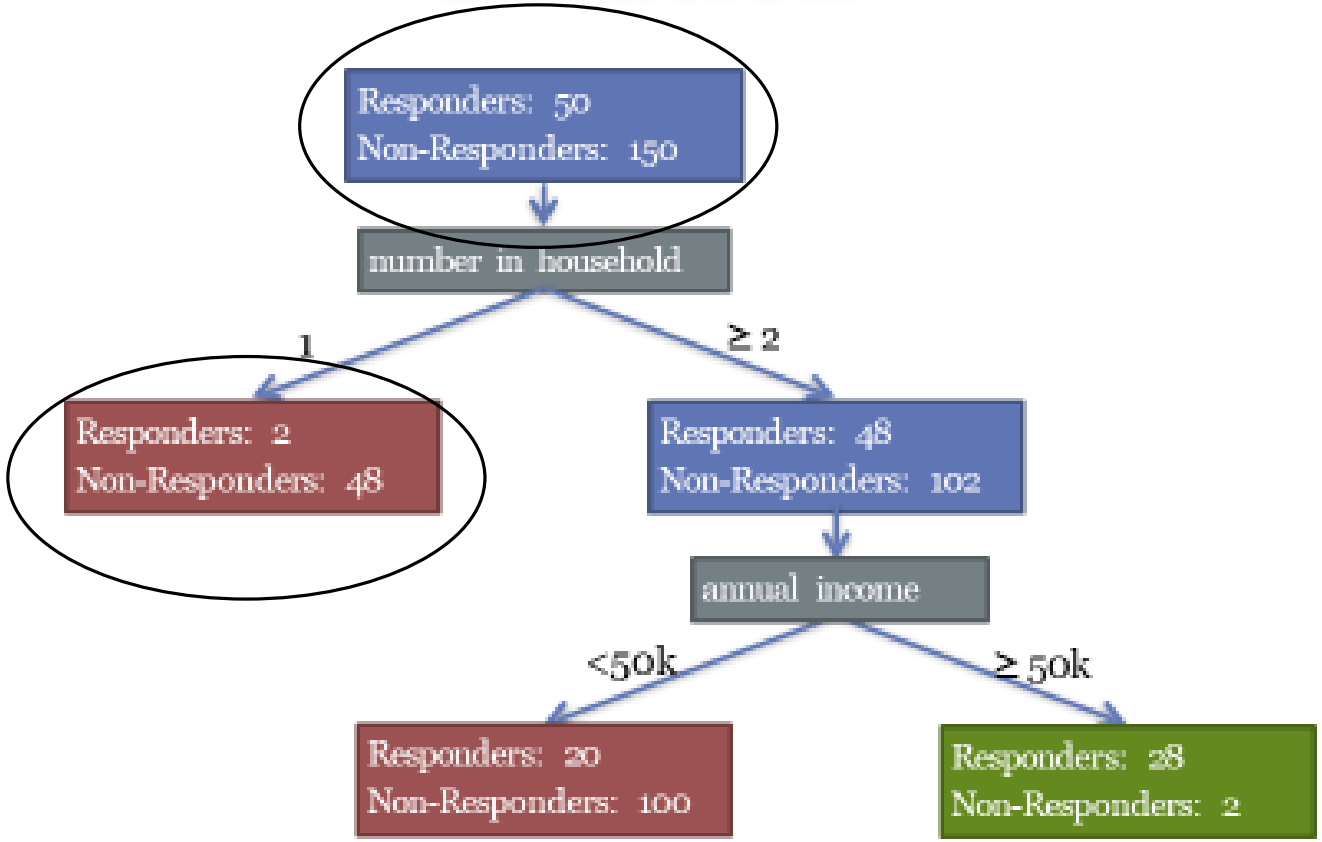
Decision Tree Model Creation



Target is a “classification” or ordinal/categorical variable

We will focus on the number of individuals in each “class level” in each node

Decision Tree Model Creation



The purity of a node is looking at how “homogeneous” the node is with respect to the target variable.

If we look at the purity of the root node to one of its child nodes:

Root node: 150/200 = 0.75

Child node: 48/50=0.96

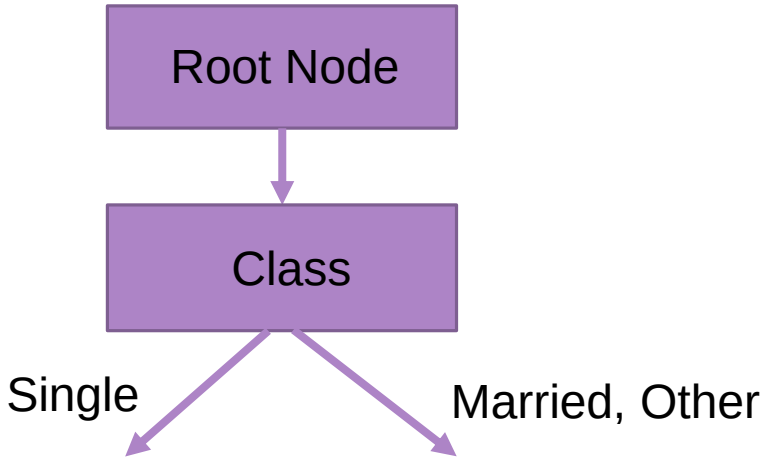
Creating the tree…

- A tree is built by recursively partitioning the training data into successively **purer** subsets.
  - (Having mostly No's **or** mostly Yes's for the target.)
- Partitioning is done according to some condition.
- Most trees will use **binary** splits (only allow for two options each time a variable is chosen)

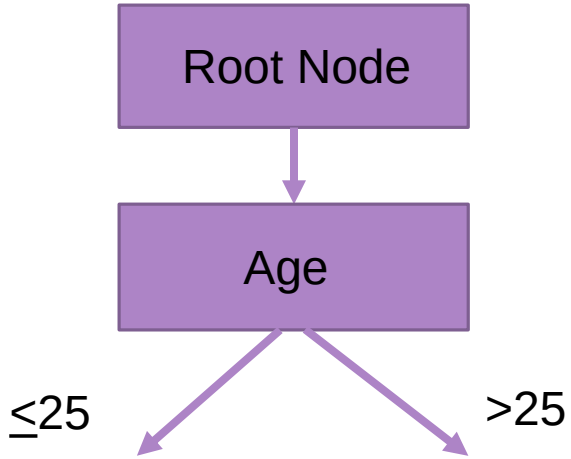
EXAMPLE

Possible splits for Marital Status = {Single, Married, Other}

LEAF 1	LEAF 2
Single	Married, Other
Married	Single, Other
Other	Single, Married



For categorical predictors, need to put categories into 2 groups (best split for defining two groups)

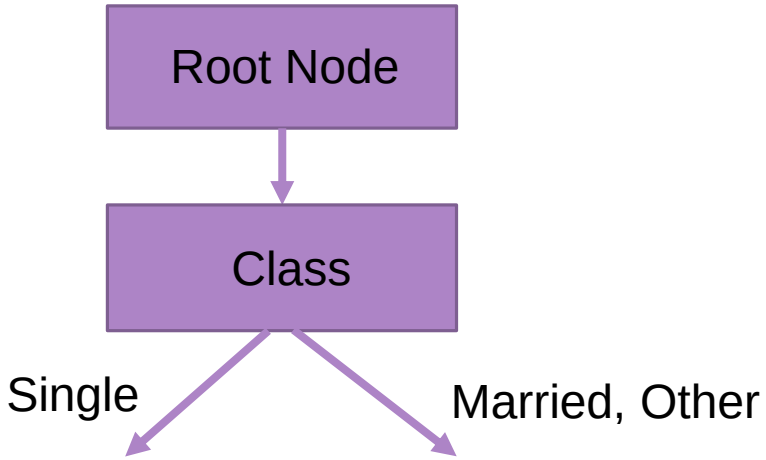


For ordinal and quantitative variables, need to find the best value to split on

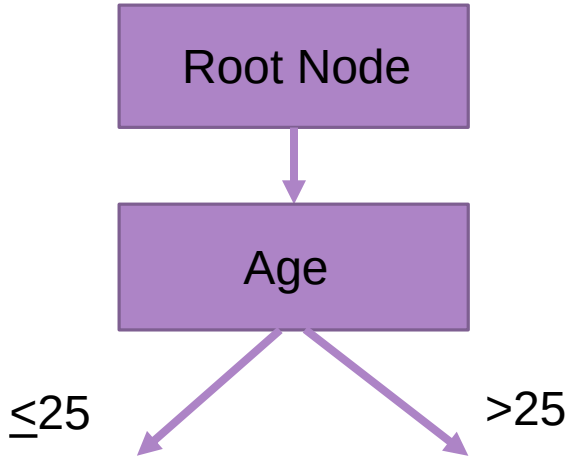
EXAMPLE

Possible splits for Marital Status = {Single, Married, Other}

LEAF 1	LEAF 2
Single	Married, Other
Married	Single, Other
Other	Single, Married



For categorical predictors, need to put categories into 2 groups (best split for defining two groups)

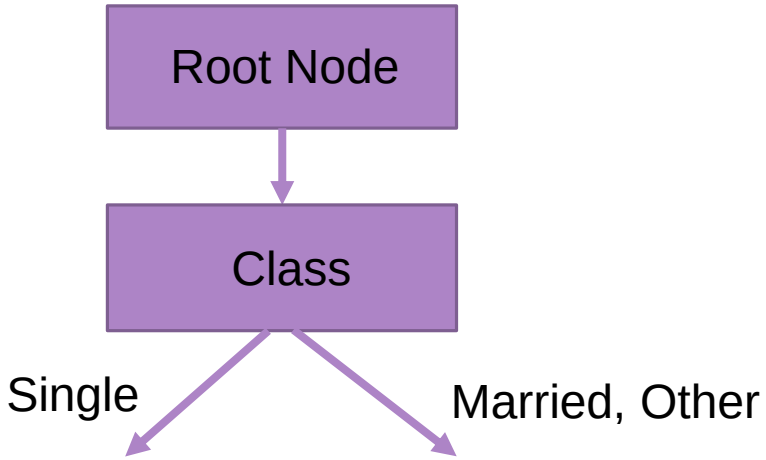


For ordinal and quantitative variables, need to find the best value to split on

EXAMPLE

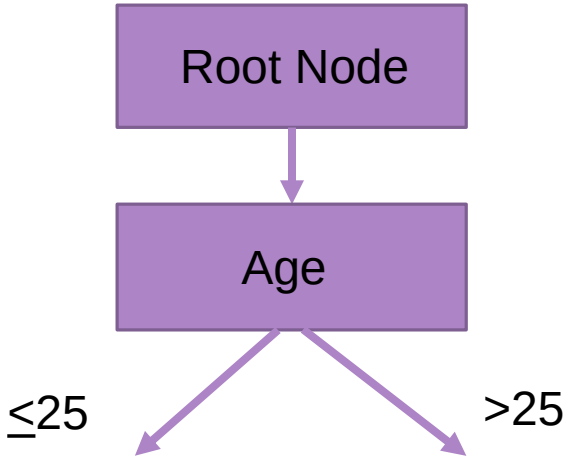
Possible splits for Marital Status = {Single, Married, Other}

LEAF 1	LEAF 2
Single	Married, Other
Married	Single, Other
Other	Single, Married



For categorical predictors, need to put categories into 2 groups (best split for defining two groups)

Data is “binned” into two groups…find best way to bin based on target



For ordinal and quantitative variables, need to find the best value to split on

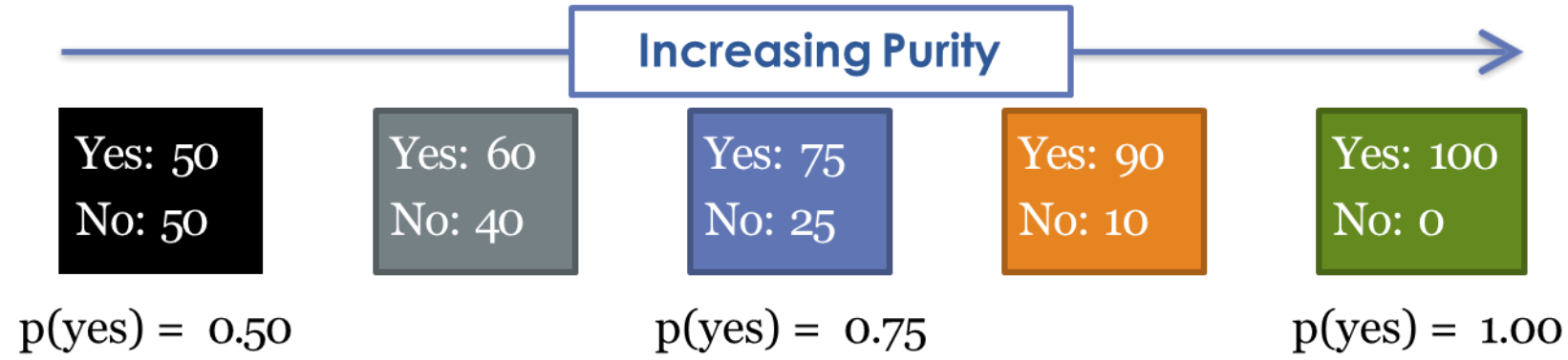
Missing Values

Decision trees can handle missing values easily (they can be placed into one of the “bins”)

In R, the rpart package has a nice imputation that it also tries

Selecting the Best Split

- There are several measures used to select the best split.
- All are similar, but not identical
- All measure the **purity** of a node



- The more pure a leaf is, the less *training* error we make in that leaf.

How do we choose the best split?

Let p(*i*|*t*) = p(class = *i*| node =*t*) be the fraction of records belonging to class *i* at a given node *t*. Let *c* be the number of classes in target variable.

MEASURES OF IMPURITY (referred to as I(t):

- Entropy(t) = $-\sum_{i=1}^{c} p(i|t) log_2 p(i|t)$
- Gini(t) = 1-$\sum_{i=1}^{c} [p(i|t)]^2$

Gain (Worth)

$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

Δ:= Gain

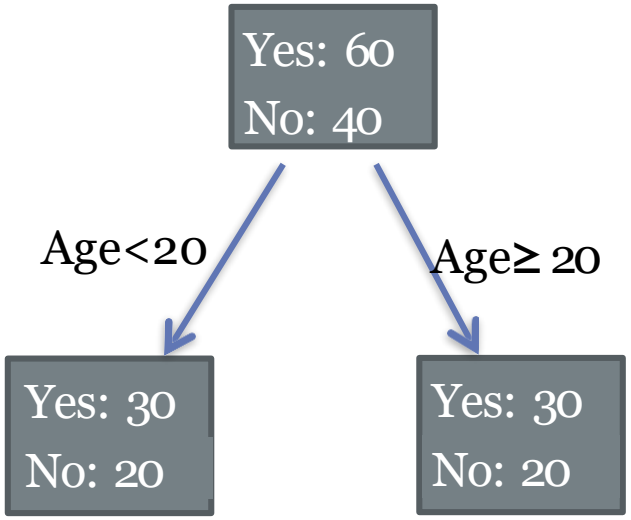
I(t):= Impurity of parent node

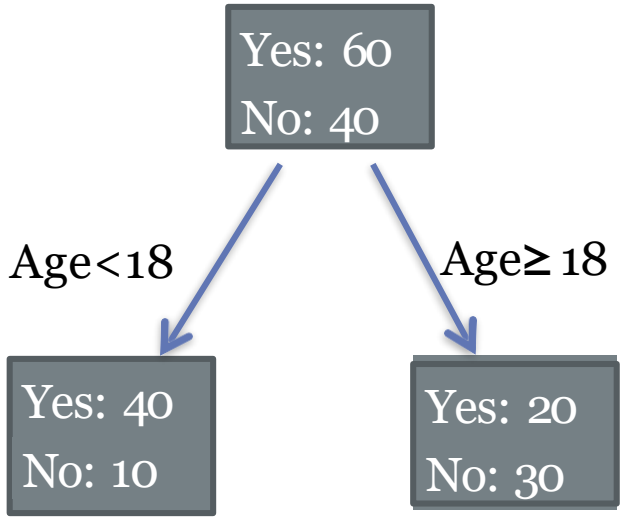
$I(t_L)$ and $I(t_R)$:= Impurity of left/right child nodes

n:= Number of observations in parent

n_L and n_R := Number of observations in left/right child

Example: Comparing 2 splits with Gain, Impurity Measure Gini

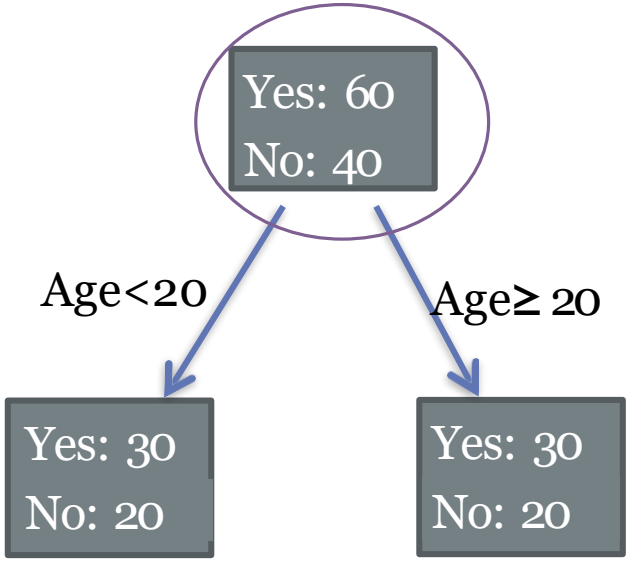


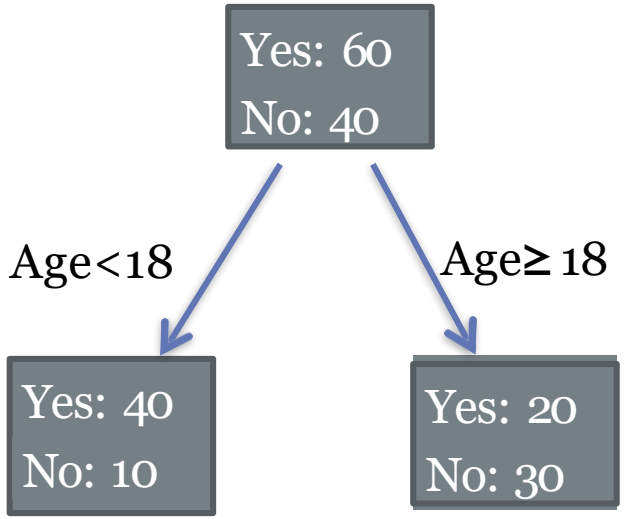


$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

$$I(t) = Gini(t) = 1 - \sum_{i=1}^{c} [p(i \mid t)]^2$$

Example: Comparing 2 splits with Gain, Impurity Measure Gini

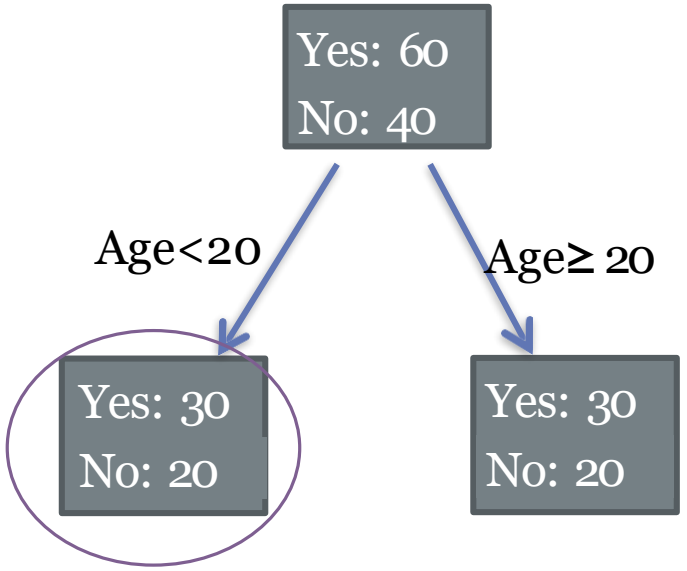


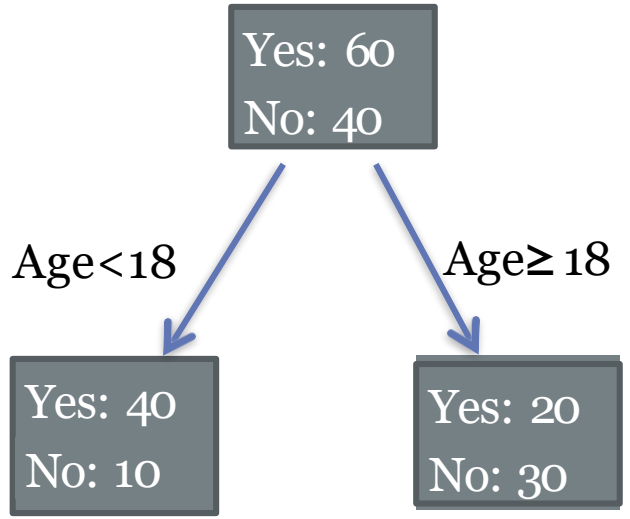


$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

$$I(t) = 1 - \left[\left(\frac{60}{100} \right)^2 + \left(\frac{40}{100} \right)^2 \right] = 0.48$$

Example: Comparing 2 splits with Gain, Impurity Measure Gini



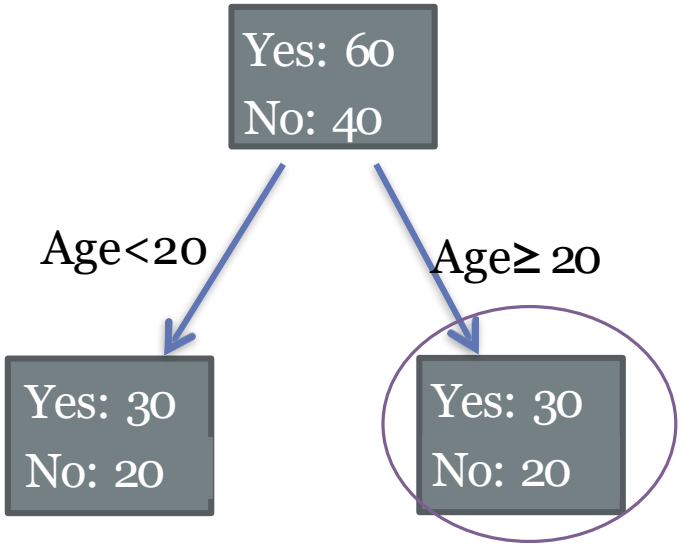


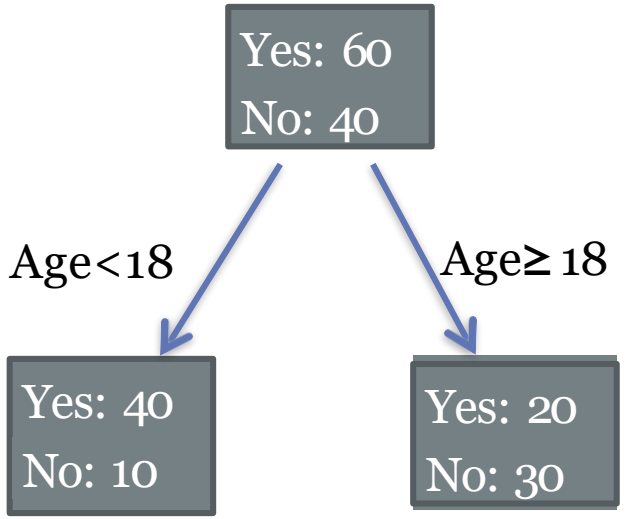
0.48

$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

$$I(t_L) = 1 - \left[\left(\frac{30}{50} \right)^2 + \left(\frac{20}{50} \right)^2 \right] = 0.48$$

Example: Comparing 2 splits with Gain, Impurity Measure Gini



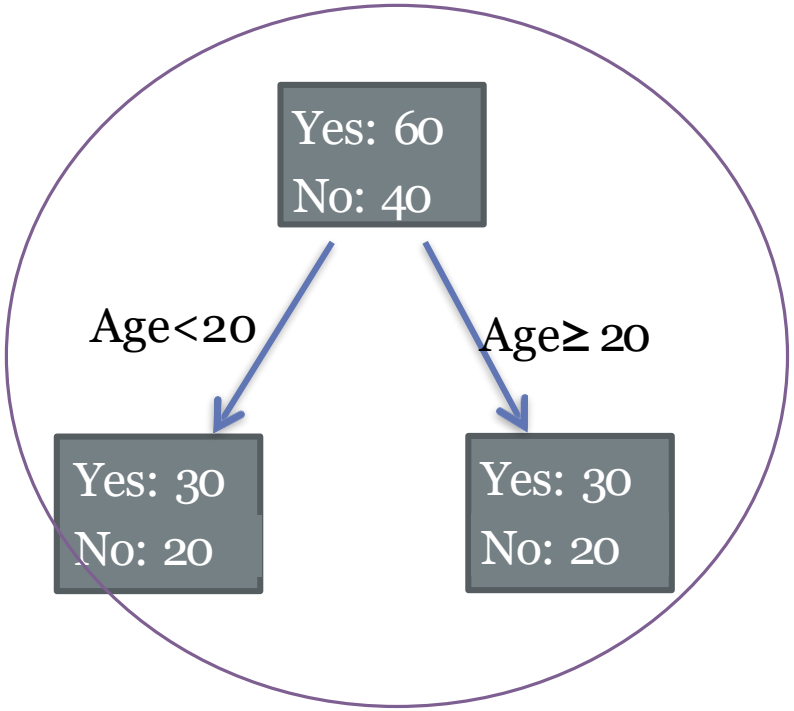


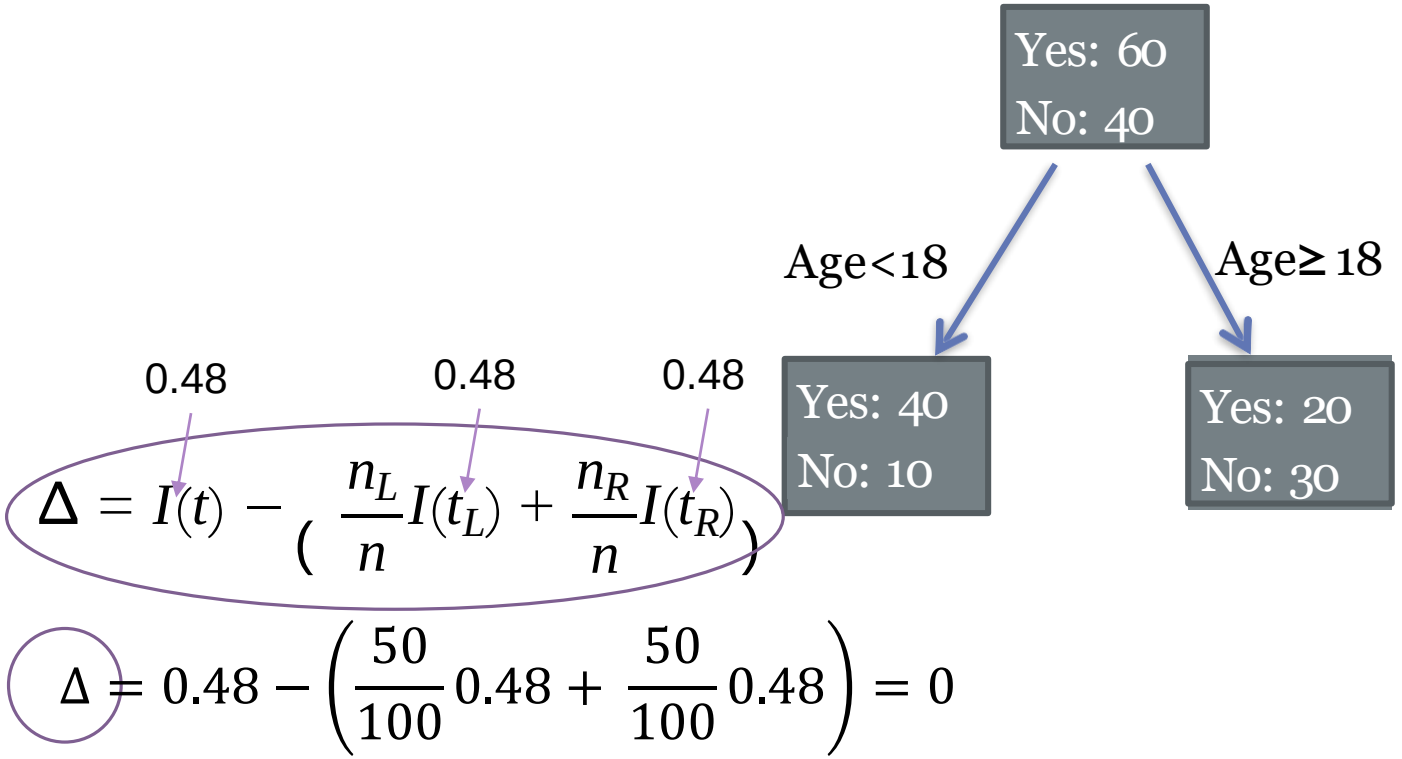
0.48 0.48

$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

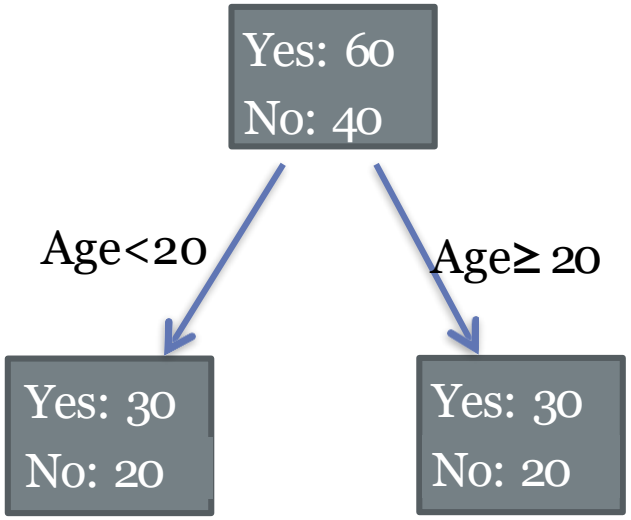
$$I(t_R) = 1 - \left[\left(\frac{30}{50} \right)^2 + \left(\frac{20}{50} \right)^2 \right] = 0.48$$

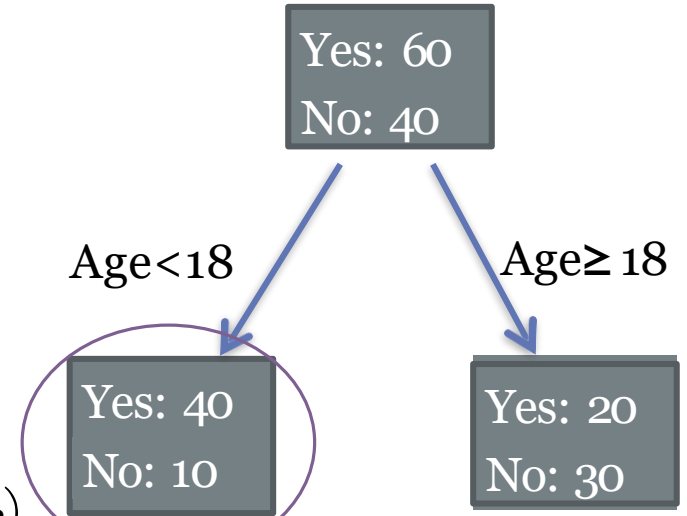
Example: Comparing 2 splits with Gain, Impurity Measure Gini





Example: Comparing 2 splits with Gain, Impurity Measure Gini

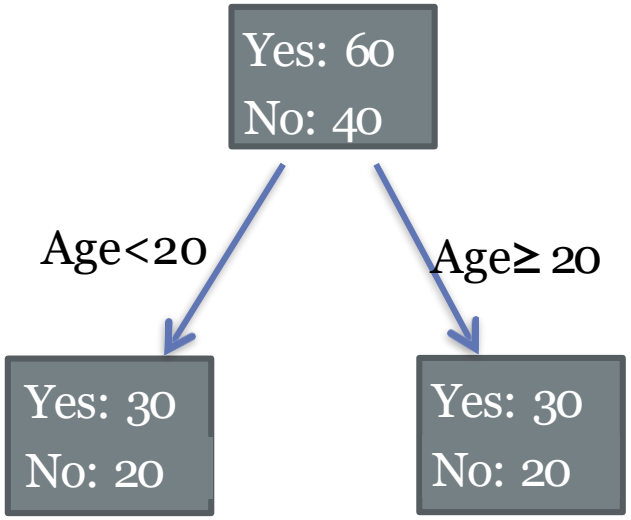


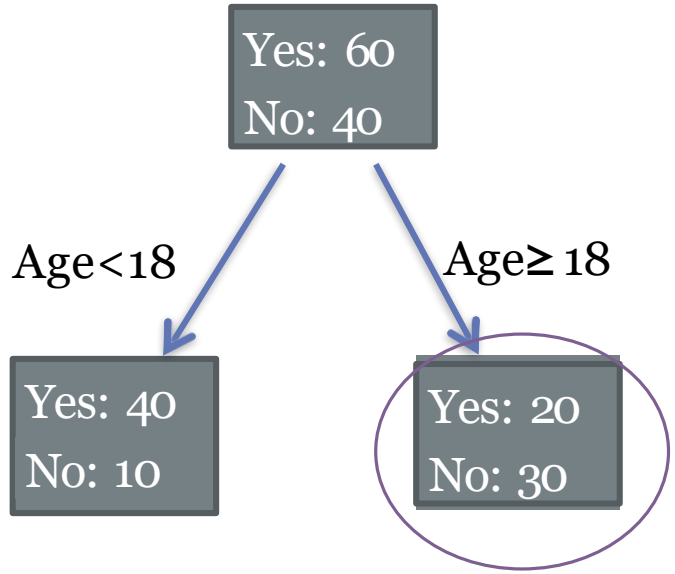


$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

$$I(t_L) = 1 - \left[\left(\frac{40}{50} \right)^2 + \left(\frac{10}{50} \right)^2 \right] = 0.32$$

Example: Comparing 2 splits with Gain, Impurity Measure Gini



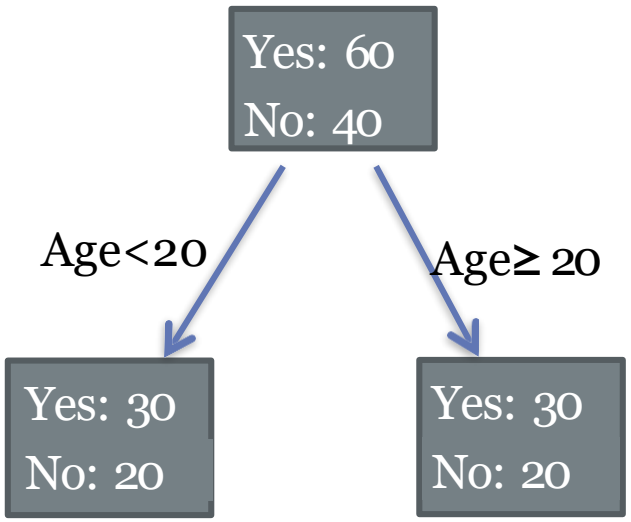


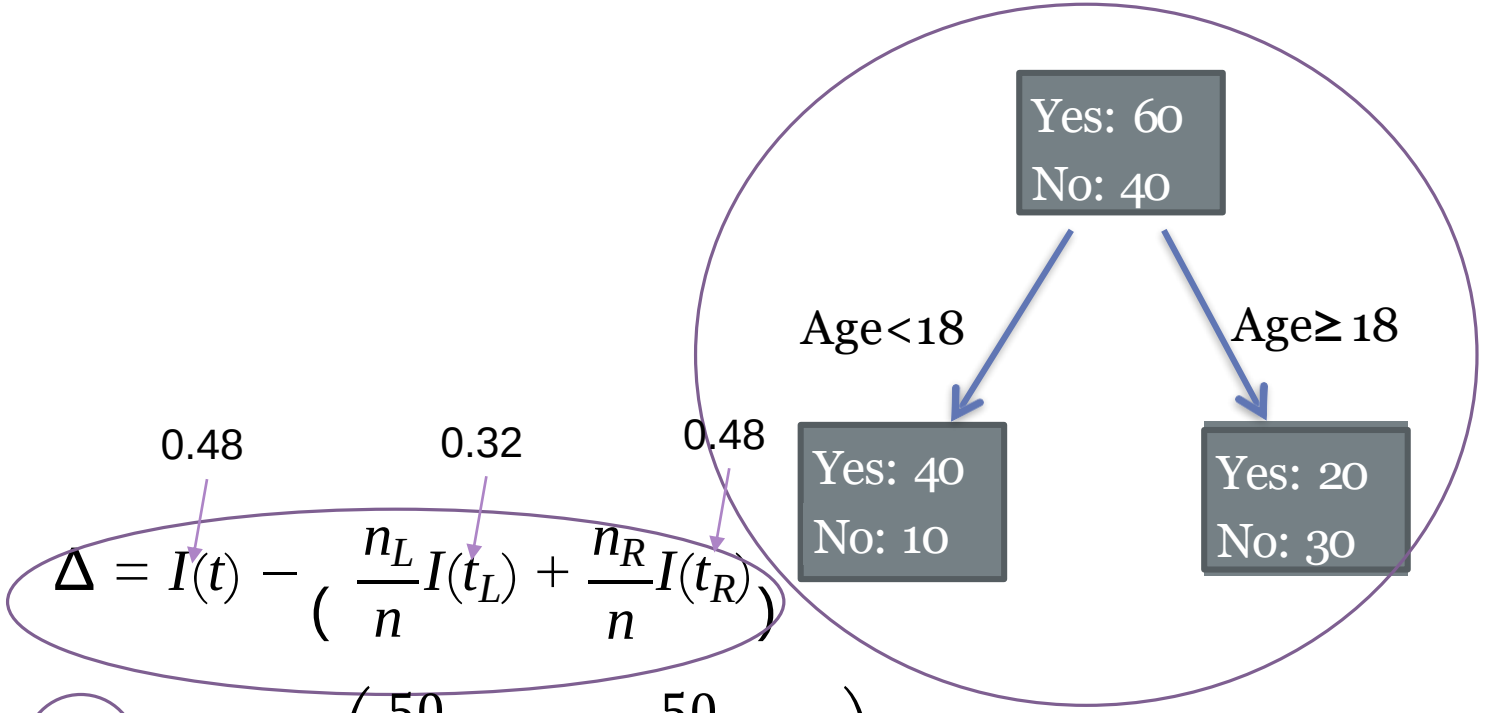
0.48 0.32

$$\Delta = I(t) - \left(\frac{n_L}{n} I(t_L) + \frac{n_R}{n} I(t_R) \right)$$

$$I(t_R) = 1 - \left[\left(\frac{20}{50} \right)^2 + \left(\frac{30}{50} \right)^2 \right] = 0.48$$

Example: Comparing 2 splits with Gain, Impurity Measure Gini





So the split on the right has a higher gain (bigger reduction in “Impurity”)

Creating the tree

- Compute the gain for all possible splits and select the best one.
- Repeat process recursively until some stopping condition is met
  - No splits meet some minimum Gain
  - All leaves have some minimum number of observations
  - A stopping condition is a way of *prepruning* the tree
- Prune Tree
  - Generally difficult to choose the right thresholds in prepruning
  - Can grow a larger tree and prune back branches in supervised fashion. (Essentially picking the threshold after the fact.)

Pruning the tree

- Simplifies the model
  - Occam's razor –law of parsimony
  - "Plurality is not to be posited without necessity" (Duns Scotus 1290)
- Prevents overfitting the training data
  - An accurate model on training: one bin for each leaf! #TerribleIdea
- **Simply remove leaves/nodes** in a bottom-up fashion, cutting splits with lowest gain first, while **optimizing performance on validation data**

Making a smaller tree

You can either grow a tree to its full length (IF your data is large, this might take awhile!)

And/or specify some controls before creating the tree (to make sure it does not grow too big)

Even if you add some controls, you can still prune more after you get the output

Rpart controls (just a few of the big ones)

MAIN OPTIONS

subset – only a subset of the rows will be used

method - "anova", "poisson", "class" or "exp“

split - gini or information (only for classification trees)

IN RPART.CONTROL OPTIONS

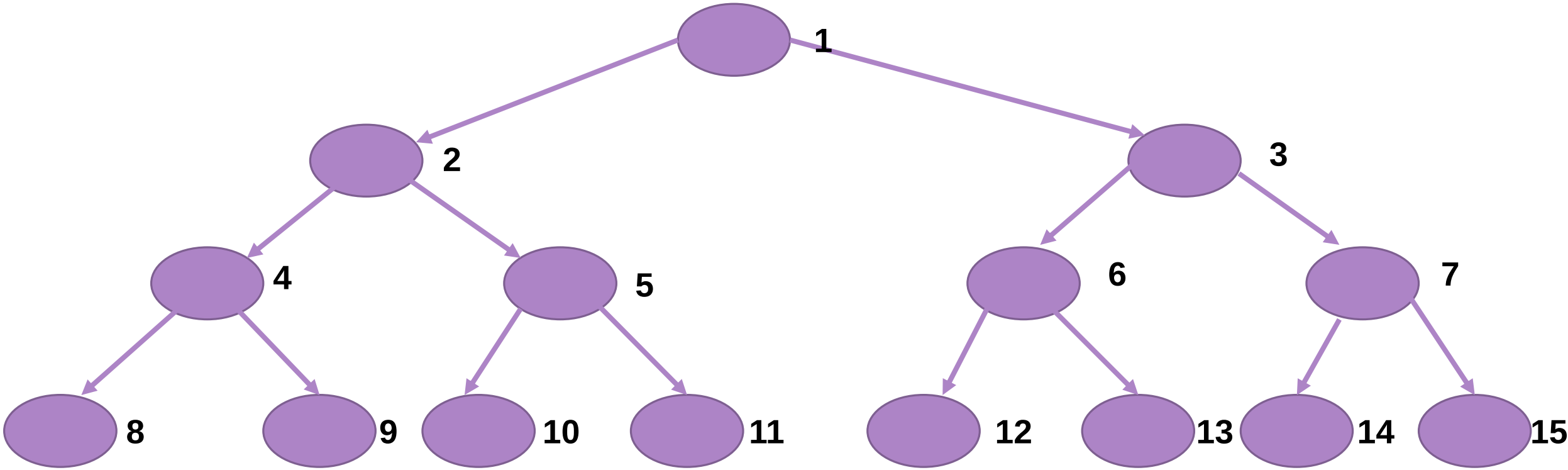
minsplit – minimum # of observations in a node needed to consider a split

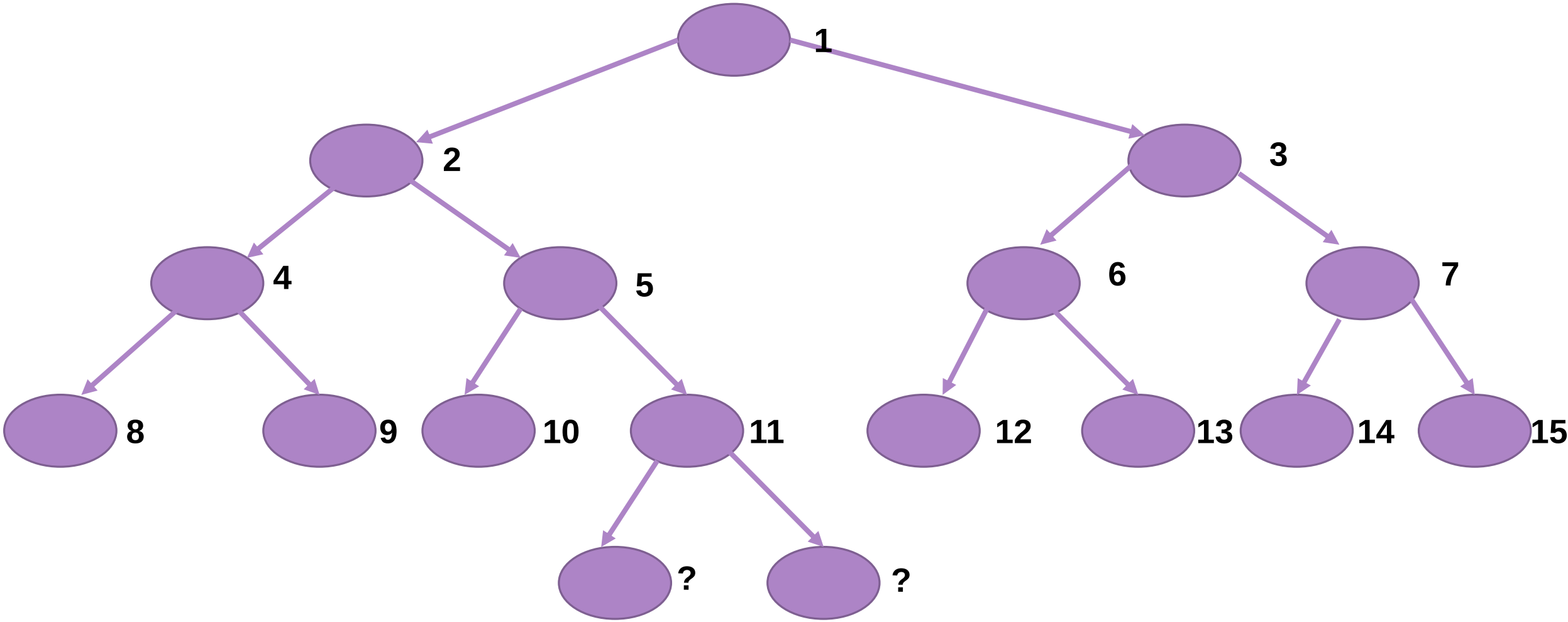
minbucket – minimum number of observations required in a terminal node

cp – complexity parameter (any split that does not decrease the lack of fit by this amount will not be tried…can save time!)

xval – number of crossvalidations

maxdepth – set maximum depth of nodes


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
?
?

Example: UCI Breast Cancer data (BCdata.R)

1. ID number
2. Clump Thickness (1 – 10)
3. Uniformity of Cell Size (1 – 10)
4. Uniformity of Cell Shape (1 – 10)
5. Marginal Adhesion (1 – 10)
6. Single Epithelial Cell Size (1 – 10)
7. Bare Nuclei (1 – 10)
8. Bland Chromatin (1 – 10)
9. Normal Nucleoli (1 – 10)
10. Mitoses (1 – 10)
11. Target: (0 for benign, 1 for malignant…524 benign and 241 malignant)

Train/test data

```
set.seed(7515)
perm=sample(1:699)
BC_randomOrder=BCdata[perm,]
train = BC_randomOrder[1:floor(0.75*699),]
test = BC_randomOrder[(floor(0.75*699)+1):699,]
```

524 in training data
175 in test data

Running the tree

```
BC.tree = rpart(Target ~ . - ID, data=train, method='class',
 parms = list(split='gini'))
```

	CP	nsplit	rel error	xerror	xstd
1	0.79781421	0	1.00000000	1.0000000	0.05963291
2	0.07650273	1	0.20218579	0.2622951	0.03608339
3	0.01639344	2	0.12568306	0.1693989	0.02951125
4	0.01000000	4	0.09289617	0.1639344	0.02906079

Variable importance

Size	Shape	Normal	Chromatin	Epithelial	Margin
22	18	16	15	14	13

CT	Bare
2	1

```
node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 524 183 0 (0.65076336 0.34923664)
   2) Size< 3.5 360  28 0 (0.92222222 0.07777778)
     4) Normal< 3.5 334   8 0 (0.97604790 0.02395210)
       8) Bare< 2.5 308   0 0 (1.00000000 0.00000000) *
       9) Bare>=2.5 26   8 0 (0.69230769 0.30769231)
        18) CT< 3.5 16   0 0 (1.00000000 0.00000000) *
        19) CT>=3.5 10   2 1 (0.20000000 0.80000000) *
     5) Normal>=3.5 26   6 1 (0.23076923 0.76923077) *
   3) Size>=3.5 164   9 1 (0.05487805 0.94512195) *
```

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 524 183 0 (0.65076336 0.34923664)
  2) Size< 3.5 360 28 0 (0.92222222 0.07777778)
    4) Normal< 3.5 334 8 0 (0.97604790 0.02395210)
      8) Bare< 2.5 308 0 0 (1.00000000 0.00000000) *
      9) Bare>=2.5 26 8 0 (0.69230769 0.30769231)
       18) CT< 3.5 16 0 0 (1.00000000 0.00000000) *
       19) CT>=3.5 10 2 1 (0.20000000 0.80000000) *
    5) Normal>=3.5 26 6 1 (0.23076923 0.76923077) *
  3) Size>=3.5 164 9 1 (0.05487805 0.94512195) *

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 524 183 0 (0.65076336 0.34923664)
2) Size< 3.5 360 28 0 (0.92222222 0.07777778)
4) Normal< 3.5 334 8 0 (0.97604790 0.02395210)
8) Bare< 2.5 308 0 0 (1.00000000 0.00000000) *
9) Bare>=2.5 26 8 0 (0.69230769 0.30769231)
18) CT< 3.5 16 0 0 (1.00000000 0.00000000) *
19) CT>=3.5 10 2 1 (0.20000000 0.80000000) *
5) Normal>=3.5 26 6 1 (0.23076923 0.76923077) *
3) Size>=3.5 164 9 1 (0.05487805 0.94512195) *

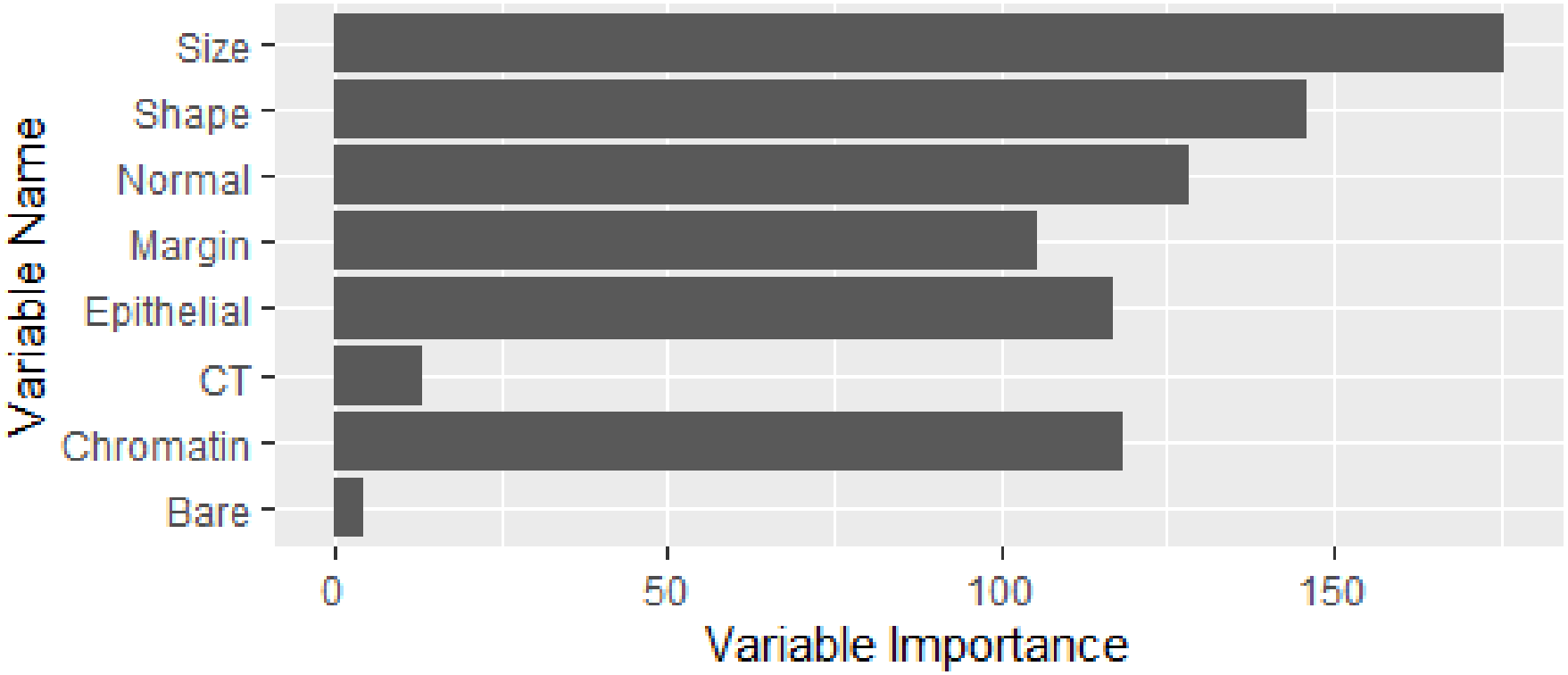
node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 524 183 0 (0.65076336 0.34923664)
2) Size< 3.5 360 28 0 (0.92222222 0.07777778)
4) Normal< 3.5 334 8 0 (0.97604790 0.02395210)
8) Bare< 2.5 308 0 0 (1.00000000 0.00000000) *
9) Bare>=2.5 26 8 0 (0.69230769 0.30769231)
18) CT< 3.5 16 0 0 (1.00000000 0.00000000) *
19) CT>=3.5 10 2 1 (0.20000000 0.80000000) *
5) Normal>=3.5 26 6 1 (0.23076923 0.76923077) *
3) Size>=3.5 164 9 1 (0.05487805 0.94512195) *

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 524 183 0 (0.65076336 0.34923664)
   2) Size< 3.5 360 28 0 (0.92222222 0.07777778)
     4) Normal< 3.5 334 8 0 (0.97604790 0.02395210)
       8) Bare< 2.5 308 0 0 (1.00000000 0.00000000) *
       9) Bare>=2.5 26 8 0 (0.69230769 0.30769231)
        18) CT< 3.5 16 0 0 (1.00000000 0.00000000) *
        19) CT>=3.5 10 2 1 (0.20000000 0.80000000) *
     5) Normal>=3.5 26 6 1 (0.23076923 0.76923077) *
   3) Size>=3.5 164 9 1 (0.05487805 0.94512195) *

```
varimp.data=data.frame(BC.tree$variable.importance)
varimp.data$names=as.character(rownames(varimp.data))

ggplot(data=varimp.data,aes(x=names,y=BC.tree.variable.importance))+geom_bar(stat="identity")+coord_flip()
+labs(x="Variable Name",y="Variable Importance")
```

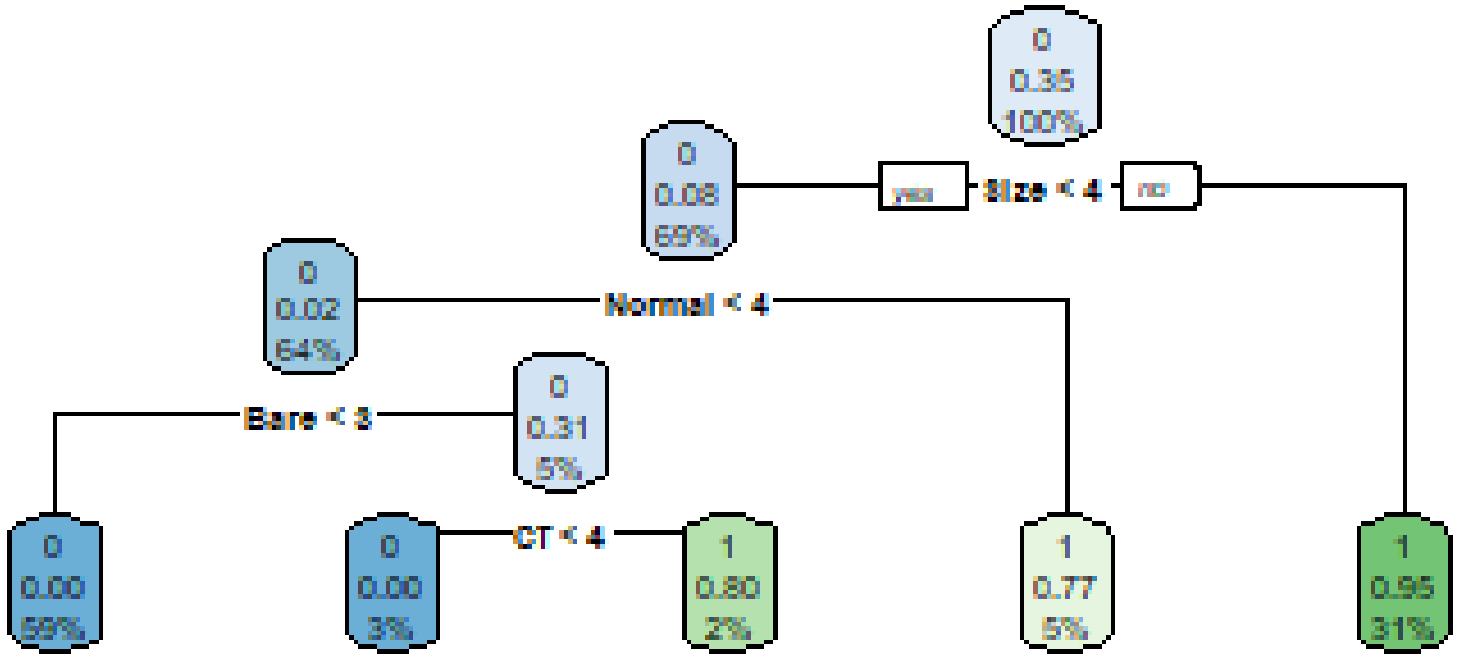


```
tscores = predict(BC.tree,type='class')
scores = predict(BC.tree, test, type='class')

##Training misclassification rate:
sum(tscores!=train$Target)/nrow(train)

### Test data:
sum(scores!=test$Target)/nrow(test)
```

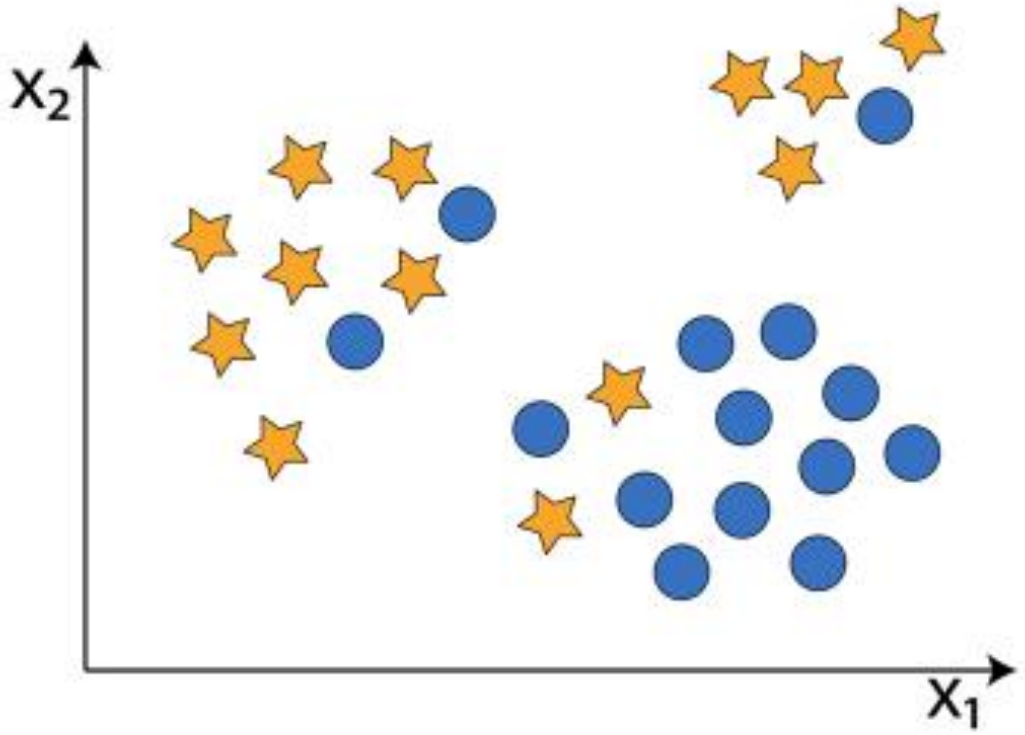
```
> sum(tscores!=train$Target)/nrow(train)
[1] 0.03244275
>
> ### Test data:
> sum(scores!=test$Target)/nrow(test)
[1] 0.05714286
```

rpart.plot(BC.tree)

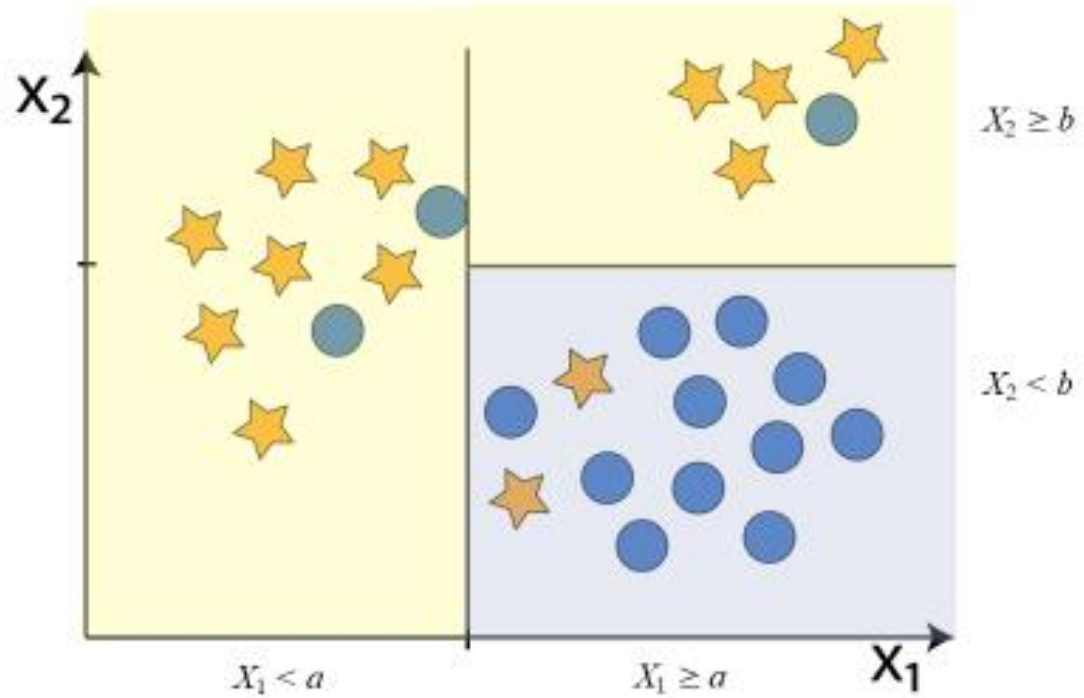
Predicted class
P(target=1)
Percent of all cases in node



Decision Tree Boundaries



Decision Tree Boundaries



Regression Trees

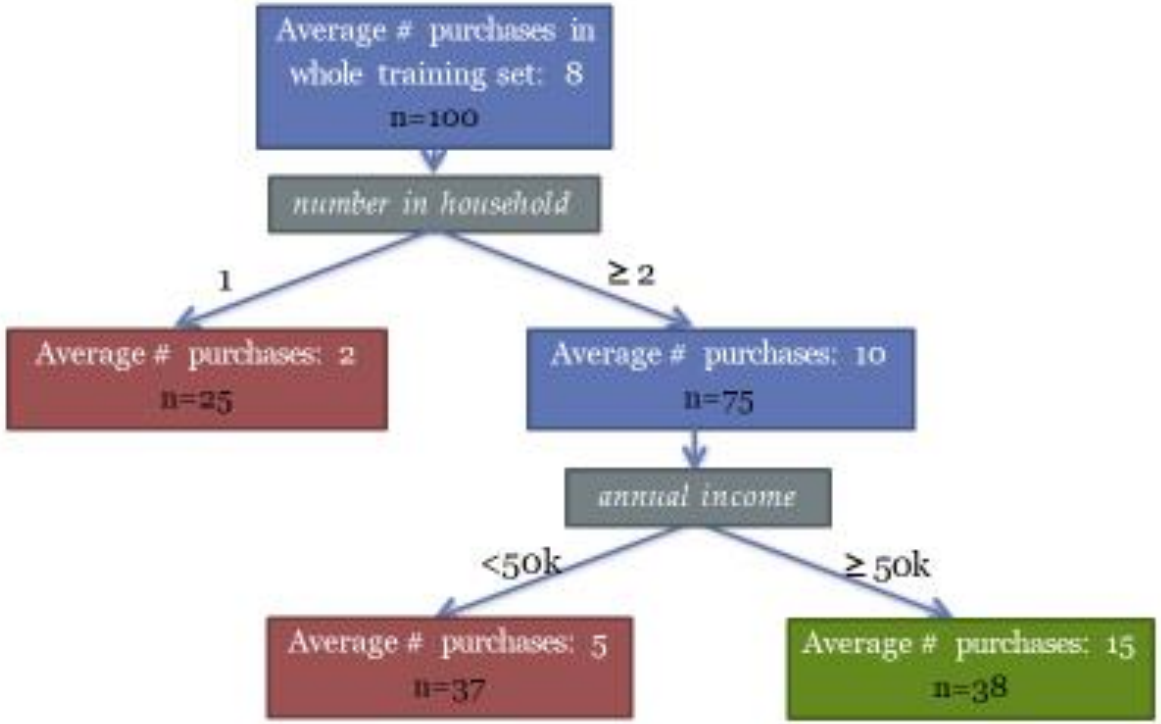
Same type of algorithm (but target is now continuous/quantitative)

Information/Gini no longer makes sense in this case

Instead of reducing impurity, we are now going to try to reduce the average sum of squares in each leaf (looking at the variance within each leaf….want variance to get smaller)

In each node, the average of the observations will be calculated (that will be the predicted value for that node)

Regression Tree Model Creation



Example: Bodyfat data set

Age - age in years.

DEXfat- body fat measured by DXA (response variable).

Waistcirc- waist circumference.

Hipcirc- hip circumference.

Elbowbreadth- breadth of the elbow.

Kneebreadth- breadth of the knee.

Anthro3a- sum of logarithm of three anthropometric measurements.

Anthro3b- sum of logarithm of three anthropometric measurements.

Anthro3c- sum of logarithm of three anthropometric measurements.

Anthro4- sum of logarithm of three anthropometric measurements.

```
body_model<-rpart(DEXfat ~ age + waistcirc + hipcirc +
  elbowbreadth + kneebreadth, data = train,
  control = rpart.control(minsplit = 10))
 summary(body_model)
 printcp(body_model)
```

```
Variable importance
   waistcirc  kneebreadth      hipcirc elbowbreadth          age
          31           28           25           10            6
```

Pruning tree

To prune back a tree, there are a number of different criteria that you can use. Two of the most commons are:

- 1. Go to the minimum value of “xerror” (crossvalidation error..used in previous example)
- 2. Use the 1-SE rule
  - Use the standard error of the crossvalidation error to find what is within 1 standard error of the lowest value and prune to that value
  - In the case of the bodyfat data, this would be 1-SE Rule: 0.318+0.074=0.392

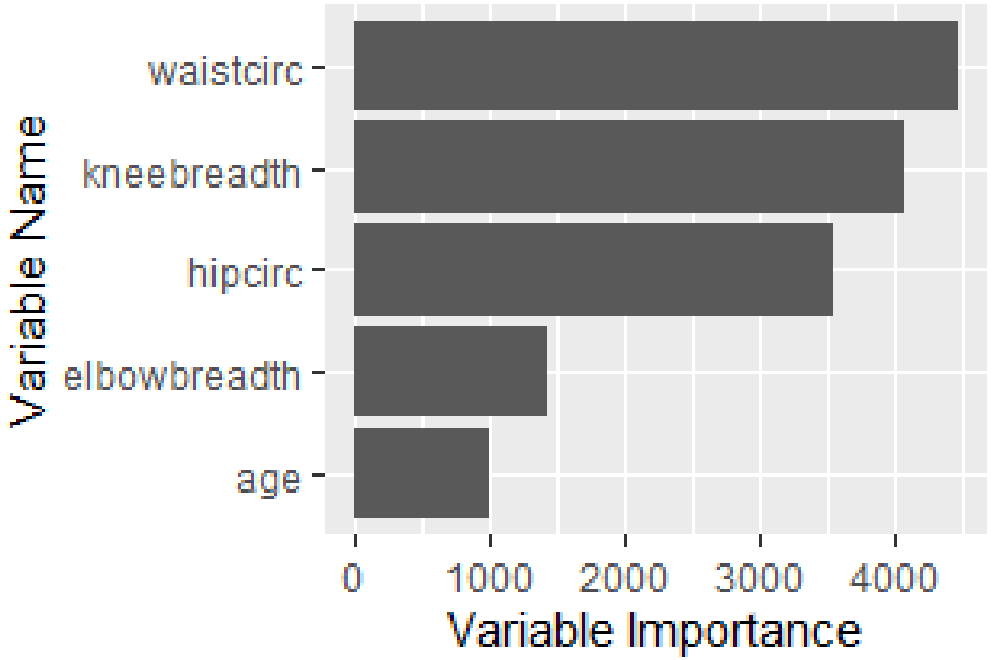
cp in the regression case tells us how much R^2 increases!!

	CP	nsplit	rel error	xerror	xstd
1	0.64291360	0	1.00000000	1.0379195	0.21139450
2	0.13509525	1	0.35708640	0.4379330	0.11128809
3	0.05556731	2	0.22199115	0.4052596	0.07270908
4	0.05175731	3	0.16642384	0.3886477	0.06530928
5	0.01951316	4	0.11466653	0.3505196	0.07310936
6	0.01438697	5	0.09515338	0.3424052	0.07387012
7	0.01000000	6	0.08076640	0.3182467	0.07357547

```
body_model2<-prune(body_model,cp=0.05175731)
 printcp(body_model2)
```

n= 53

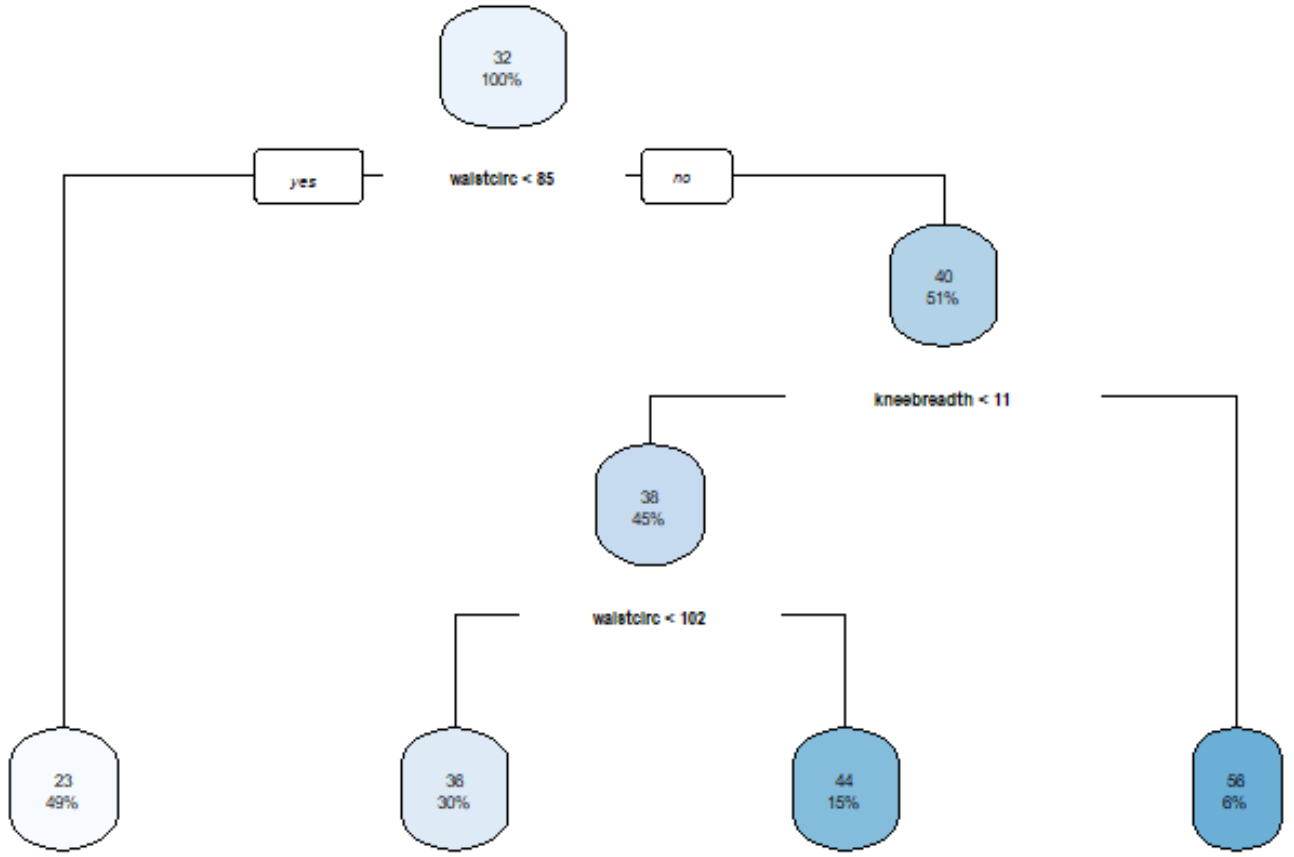
	CP	nsplit	rel error	xerror	xstd
1	0.642914	0	1.00000	1.03792	0.211394
2	0.135095	1	0.35709	0.43793	0.111288
3	0.055567	2	0.22199	0.40526	0.072709
4	0.051757	3	0.16642	0.38865	0.065309

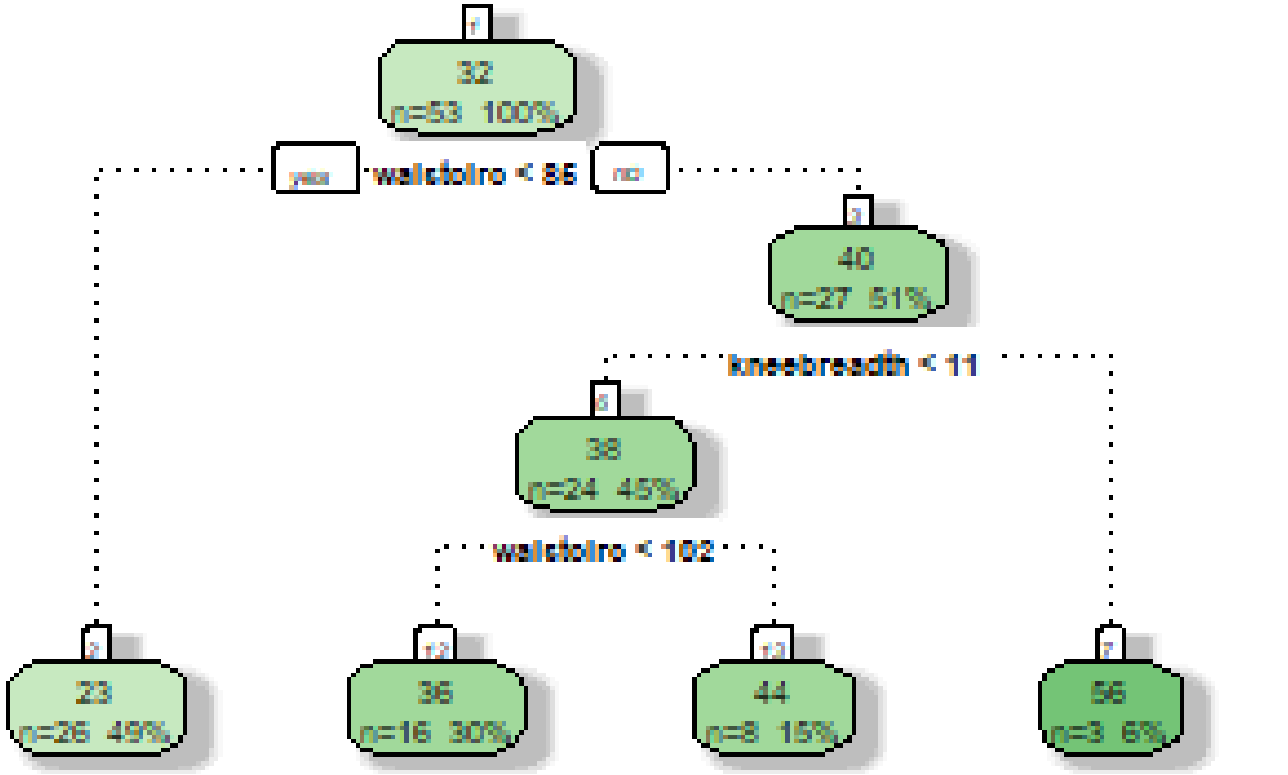


Test MAE and MAPE

MAE for test data: 6.51

MAPE for test data: 27.3%





Rattle 2021-Sep-24 10:11:04 sjsimmo2

Advantages of tree models

Explainability

Can handle missing values

Can be used for variable selection

Great for ensembles
- (basis for Random Forest and Gradient Boosting!)

No assumptions to verify

Generally immune to scale of input variables/standardization
- (less effort in preprocessing)

Generally immune to the effect of outliers or high leverage observations

Can handle correlated inputs

Disadvantages

Simplistic Regression/Decision Surface (not good estimation when only a few predictor variables and continuous response)

All variables are forced to interact

- Only the top split acts independently
- Inefficient

Greedy algorithm

- Cannot return the globally optimal tree

Can be unstable (sensitive to small changes in input) – both when training the model AND when making predictions

Conditional Trees

The package partykit can create decision trees through ctree

With conditional trees, first the best variable is selected…which variable is MOST associated with the response (using a significance test….Bonferroni adjustment is used as default)

Once a variable has been selected, the optimal split is chosen for that variable

This procedure can be done on either categorical responses or ordinal/continuous responses

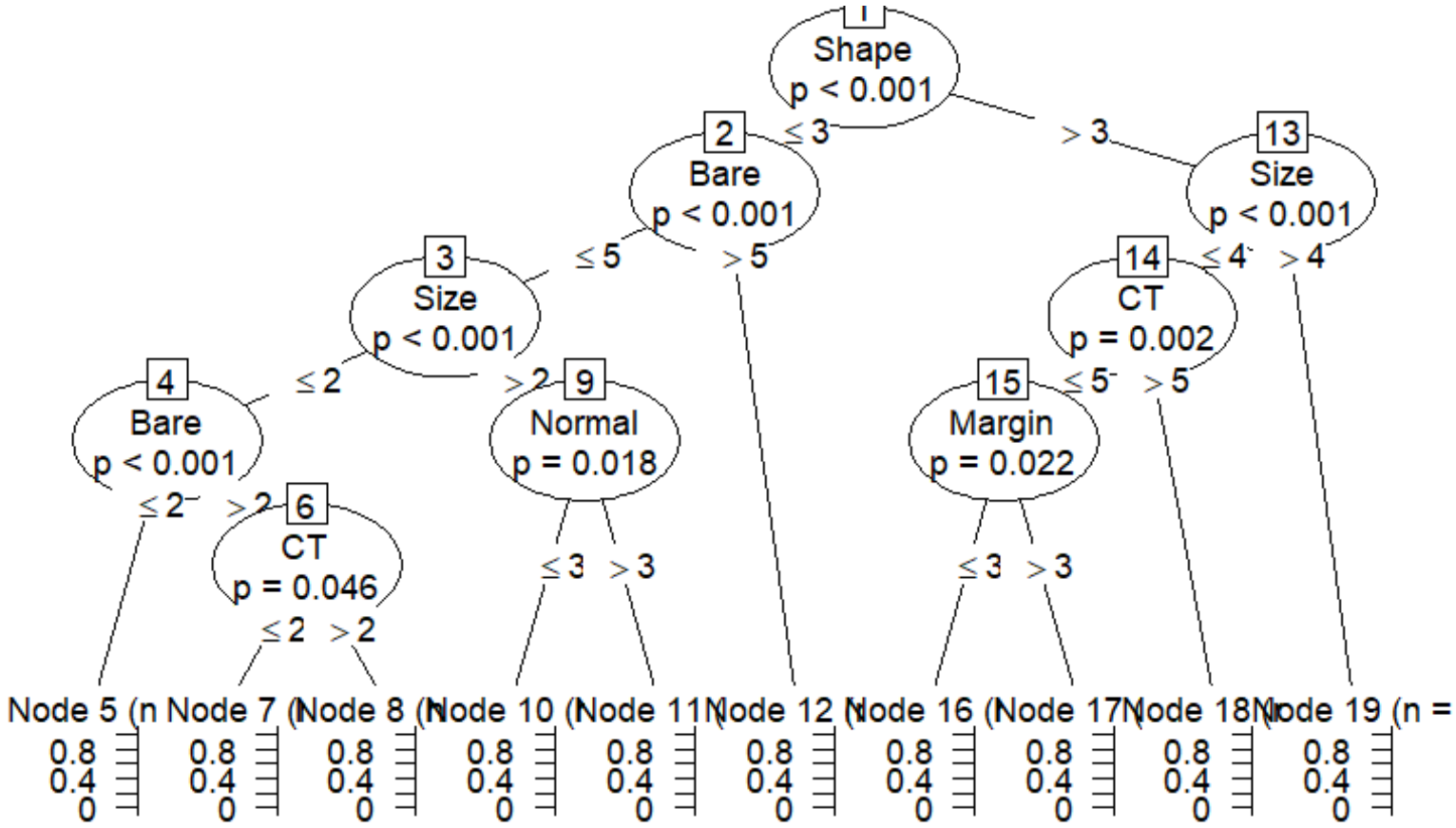
Classification example:

```
set.seed(7515)

perm=sample(1:699)

BC_randomOrder=BCdata[perm,]

train = BC_randomOrder[1:floor(0.75*699),]

model1=ctree(Target ~ . - ID, data=train)

model1

plot(model1)
```

Model formula:
Target ~ CT + Size + Shape + Margin + Epithelial + Bare + Chromatin + Normal + Mitoses

Fitted party:
[1] root
| [2] Shape <= 3
| | [3] Bare <= 5
| | | [4] Size <= 2
| | | | [5] Bare <= 2: 0.000 (n = 289, err = 0.0)
| | | | [6] Bare > 2
| | | | | [7] CT <= 2: 0.000 (n = 9, err = 0.0)
| | | | | [8] CT > 2: 0.200 (n = 10, err = 1.6)
| | | [9] Size > 2
| | | | [10] Normal <= 3: 0.062 (n = 16, err = 0.9)
| | | | [11] Normal > 3: 0.714 (n = 7, err = 1.4)
| | [12] Bare > 5: 0.889 (n = 18, err = 1.8)
| [13] Shape > 3
| | [14] Size <= 4
| | | [15] CT <= 5
| | | | [16] Margin <= 3: 0.083 (n = 12, err = 0.9)
| | | | [17] Margin > 3: 0.778 (n = 9, err = 1.6)
| | | [18] CT > 5: 0.963 (n = 27, err = 1.0)
| | [19] Size > 4: 0.984 (n = 127, err = 2.0)

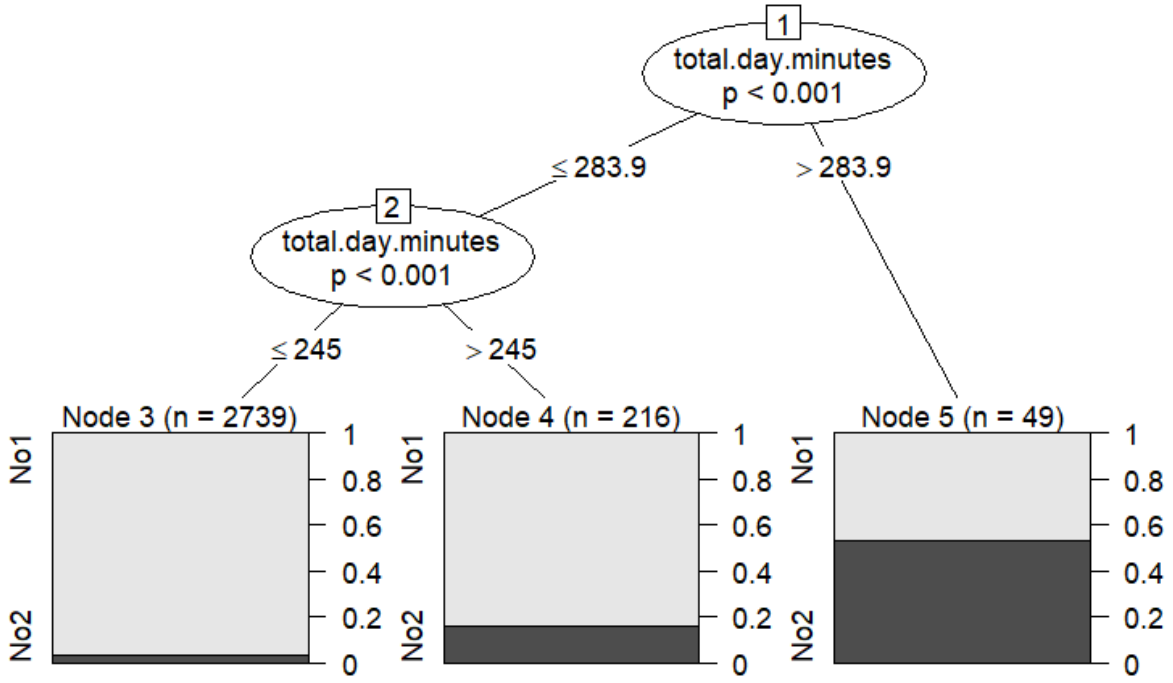
Number of inner nodes: 9
Number of terminal nodes: 10



Can be used to find bins:

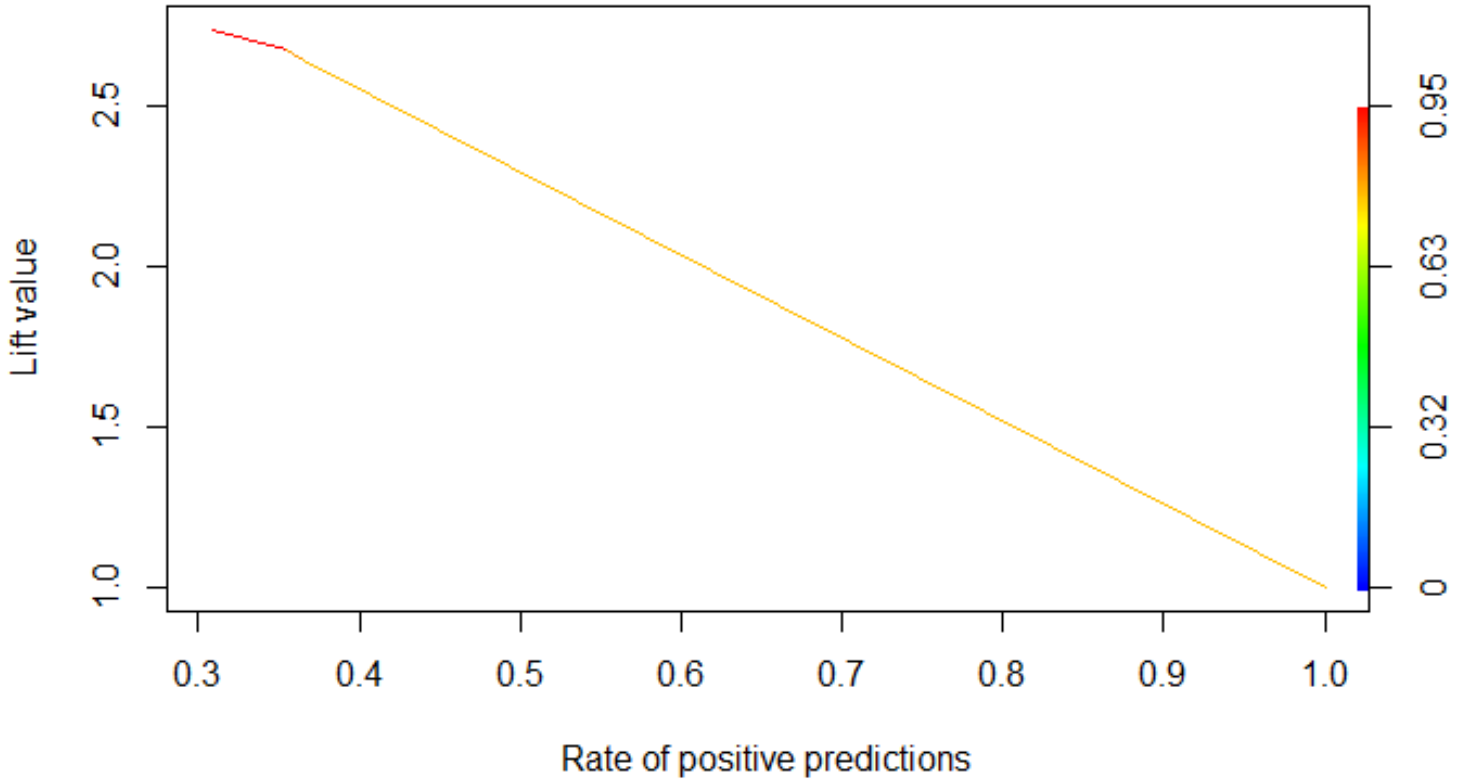
```
### Example for binning data:
churn=read.csv("Q:\\My Drive\\Data Mining\\Data\\tele_churn.csv")
churn$y<-ifelse(churn$churn=="TRUE",1,0)
churn$y<-ordered(churn$y,levels=c(0,1),labels="No","Yes")
model1<-ctree(y~total.day.minutes,data=churn)
model1
plot(model1)
```

Bins:
Total Day Minutes $\leq$ 245
245 < Total Day Minutes $\leq$ 283.9
Total Day Minutes > 283.9

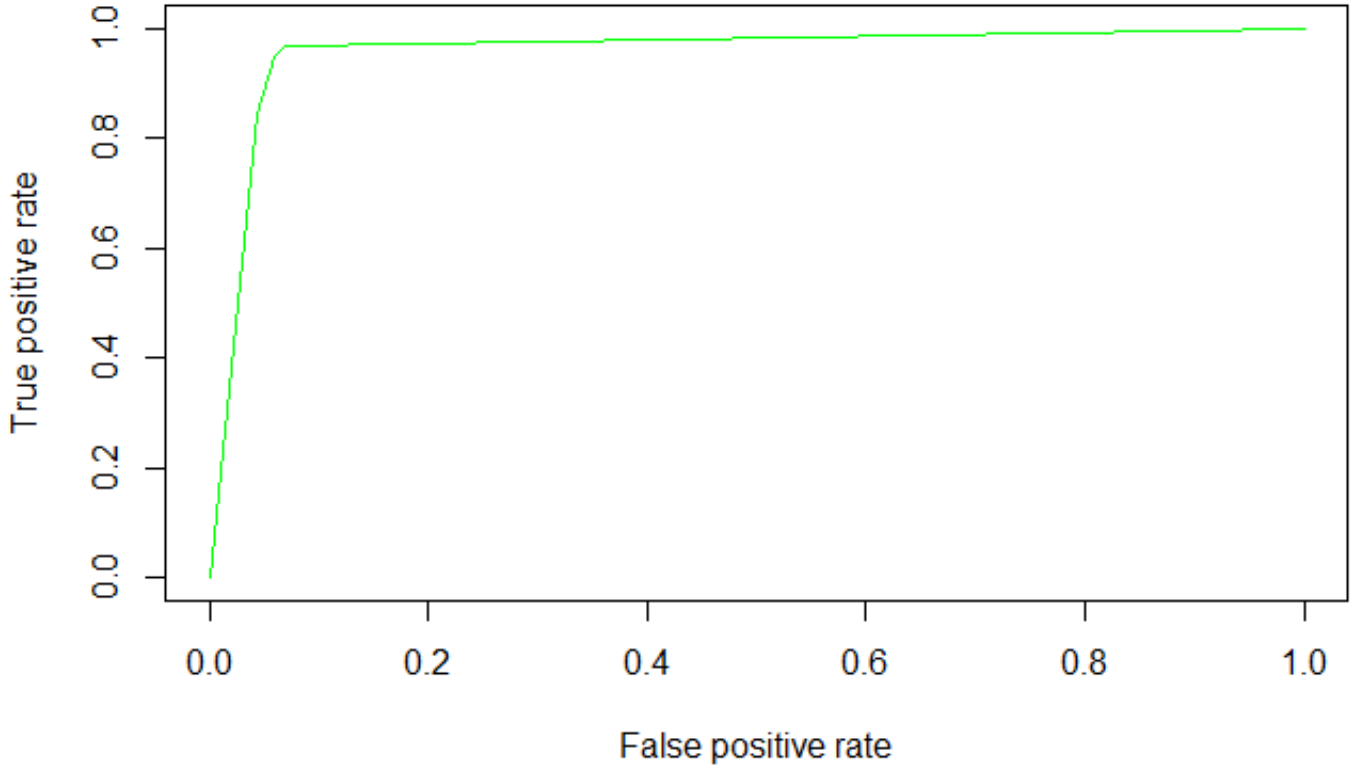


Some interesting extras

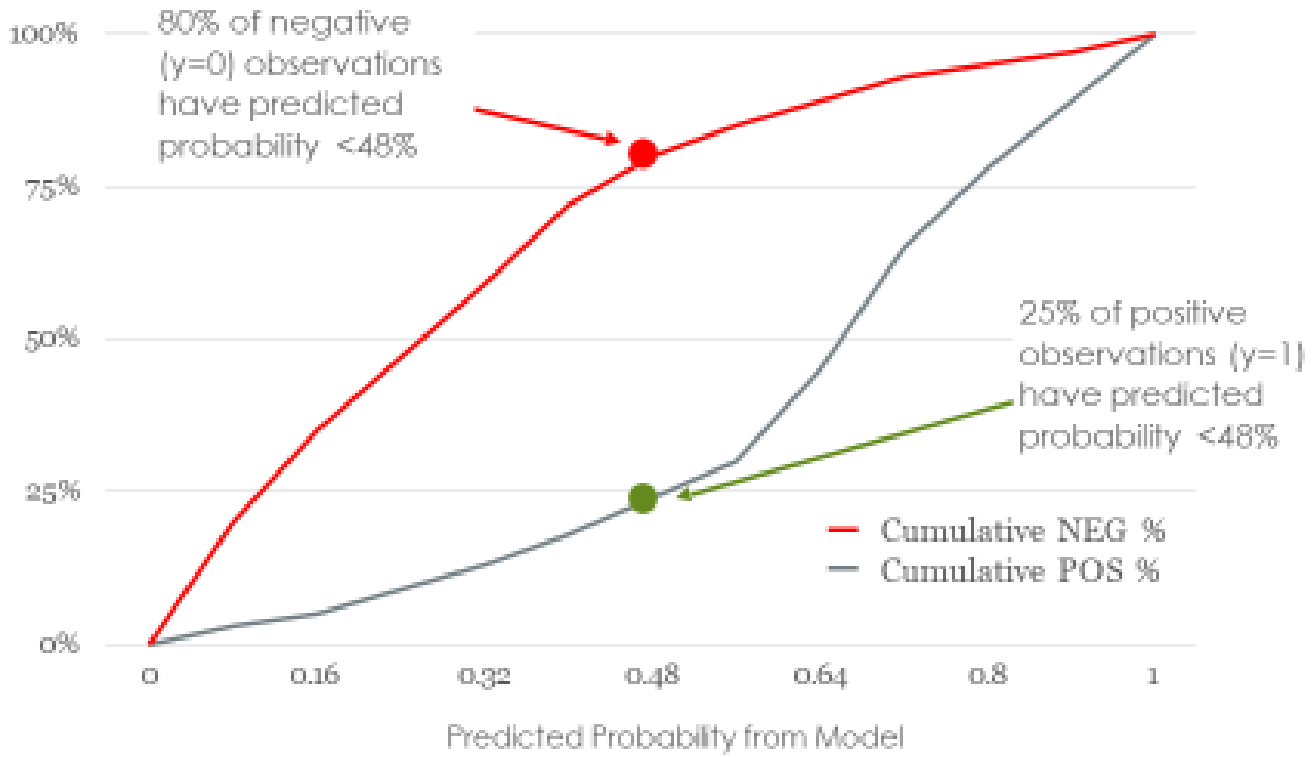
```
scores1=predict(BC.tree,test,type="prob")
pred_val <-prediction(scores1[,2],test$Target)
plot(performance(pred_val, measure="lift", x.measure="rpp"), colorize=TRUE)
```



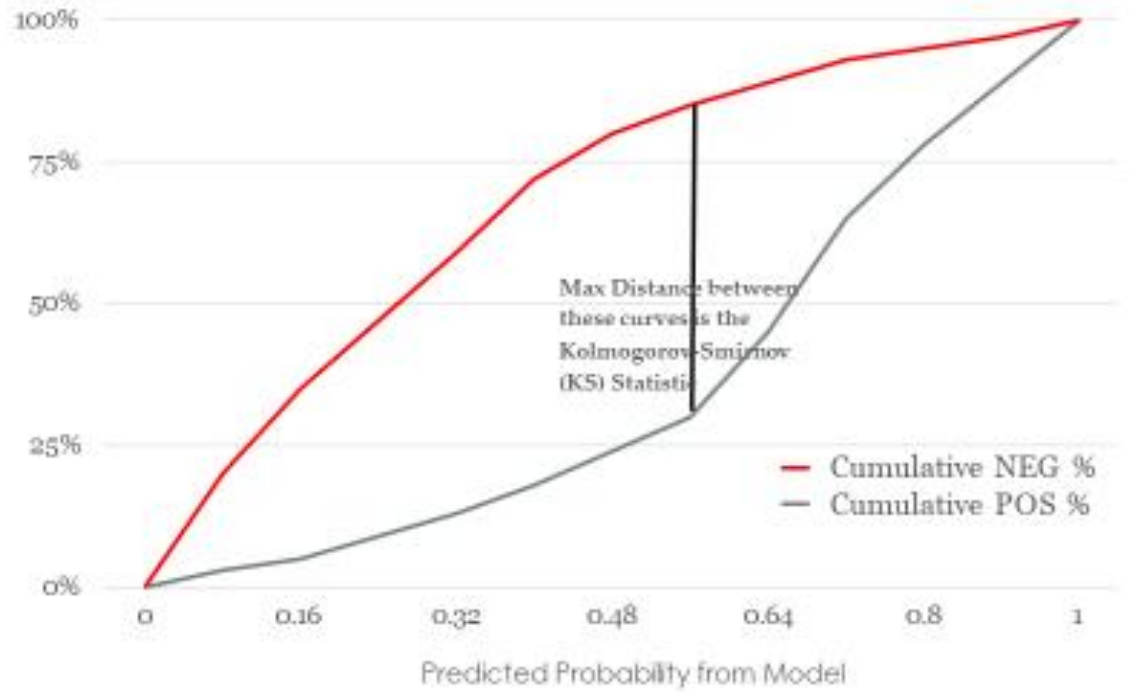
```
# Calculating True Positive and False Positive Rate
perf_val <- performance(pred_val, "tpr", "fpr")
#Plot the ROC curve
plot(perf_val, col = "green", lwd = 1.5)
```



Kolmogorov-Smirnov (KS) Statistic



Kolmogorov-Smirnov (KS) Statistic



```
#Calculating KS statistics
ks1.tree <- max(attr(perf_val, "y.values")[[1]] - (attr(perf_val, "x.values")[[1]]))
ks1.tree
[1] 0.8971412
```

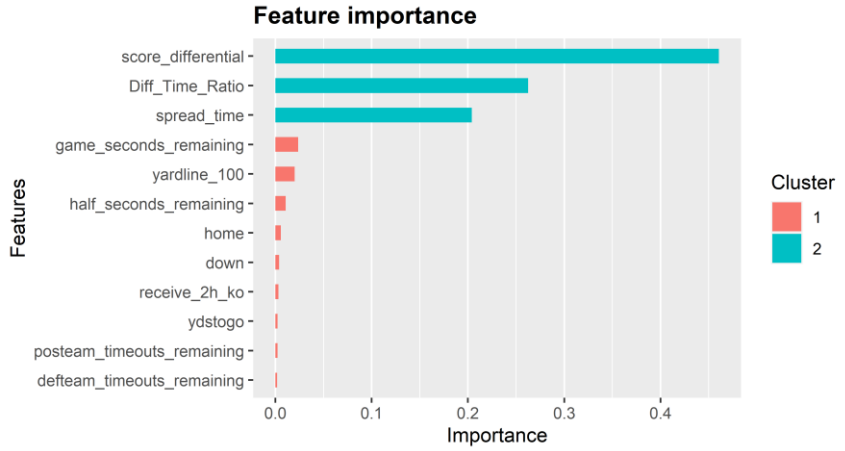
Model Reliance (MR)

Model Reliance is a model agnostic procedure

Calculates the ratio of expected loss when “noise” is introduced versus the expected loss of the original data set

For example, if we are trying to understand the importance of X_1, we can permute values of X_1 and compare loss of this permuted value of X_1 to the original data set (the bigger the loss, the more important X_1 was to the predictions)

$$MR = \frac{Expected\ loss\ under\ noise}{Expected\ loss\ original\ model}$$



Using the original BC data set

```
VI <- vector(length=ncol(train)-2)

loss.model=mean(abs(train$Target-
as.numeric(as.character(predict(BC.tree,type
="class")))))

for (j in 2:10)
  {temp1=train
   temp1[,j]=sample(train[,j])
   loss.noise = mean(abs(train$Target-
as.numeric(as.character(predict(BC.tree,
newdata=temp1,type="class")))))
   VI[(j-1)] = loss.noise/loss.model
}

VI<-data.frame(VI)
rownames(VI)<-colnames(train[2:10])
VI
```

```
## CT          1.764706
## Size        6.588235
## Shape       1.000000
## Margin      1.000000
## Epithelial  1.000000
## Bare        3.352941
## Chromatin   1.000000
## Normal      6.117647
## Mitoses     1.000000
```

```
## CT          1.764706
## Size        6.588235
## Shape       1.000000
## Margin      1.000000
## Epithelial  1.000000
## Bare        3.352941
## Chromatin   1.000000
## Normal      6.117647
## Mitoses     1.000000
```