Aprendizado de Máquinas
Árvore de decisão

Douglas Rodrigues

Universidade Federal Fluminense

Árvores com rpart()

- Vamos construir árvores de decisão usando o comando rpart().

```
> library(rpart)
> library(rpart.plot) #Para desenhar a árvore
> playTree<-rpart(Play~.,data=golf)
> rpart.plot(playTree) #Para desenhar a árvore
```

Árvores com rpart()

- Vamos construir árvores de decisão usando o comando rpart().

```
> library(rpart)
> library(rpart.plot) #Para desenhar a árvore
> playTree<-rpart(Play~.,data=golf)
> rpart.plot(playTree) #Para desenhar a árvore
```

- Observe que a árvore ficou "vazia": assuma **Yes** sempre, e acerte com precisão de 64%.

- Isso ocorre devido aos valores iniciais do comando rpart.control(), que ajusta os parâmetros da função rpart().

```
> rpart.control
```

rpart.control()

Principais parâmetros:

- **`minsplit:`** o número mínimo de observações que devem existir em um nó para que uma divisão seja tentada. Padrão: minsplit=20.
- **`minbucket:`** o número mínimo de observações em qualquer nó terminal (folhas). Padrão: minbucket=minsplit/3.
- **`cp (complexity parameter):`** O mínimo de ganho de ajuste que devemos ter em cada divisão. O principal papel desse parâmetro é economizar tempo de computação removendo as divisões que obviamente não valem a pena. Padrão: cp=0.01
- **`maxdepth:`** Profundidade máxima da árvore (a profundidade da raiz é zero). Não pode ser maior que 30.

- Exemplo 1

```
> ctrl=rpart.control(minsplit=0)
>playTree<-rpart(Play∼.,data=golf,control=ctrl)
> rpart.plot(playTree)
```

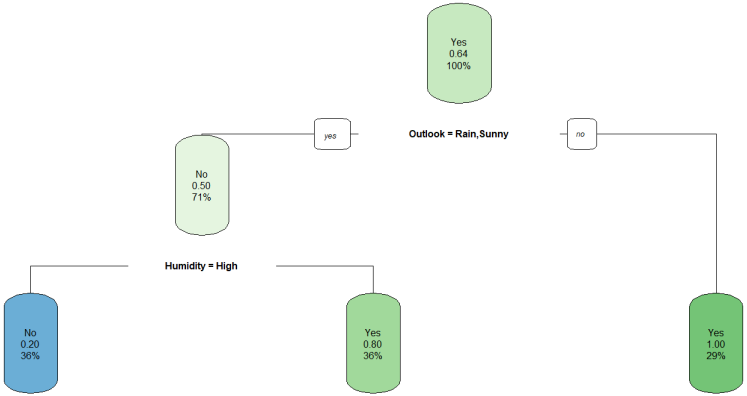
- Exemplo 2

```
> ctrl=rpart.control(minsplit=0,cp=0.1)
> playTree<-rpart(Play∼.,data=golf,control=ctrl)
> rpart.plot(playTree)
```

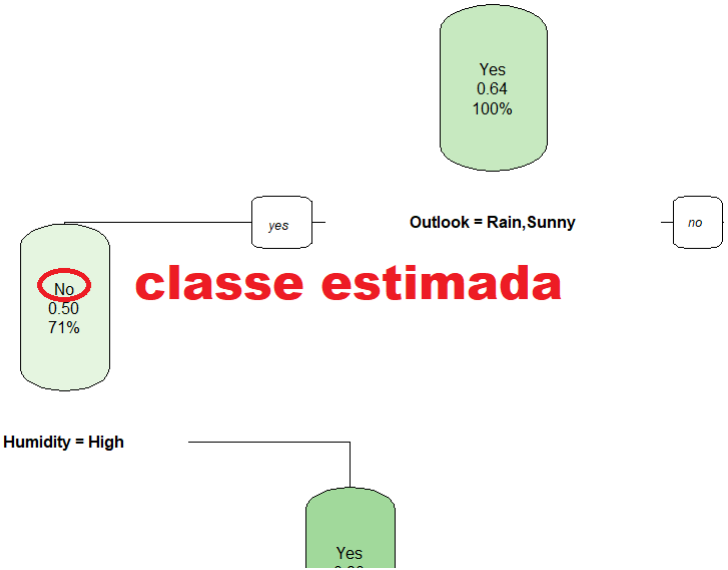
- Exemplo 3

```
> ctrl=rpart.control(minsplit=0,maxdepth=3)
> playTree<-rpart(Play∼.,data=golf,control=ctrl)
> rpart.plot(playTree)
```

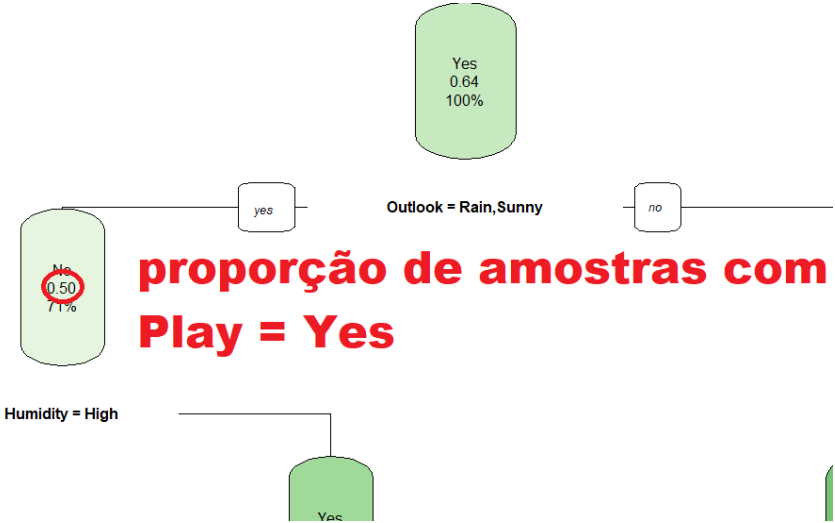
Entendendo rpart.plot



Entendendo rpart.plot

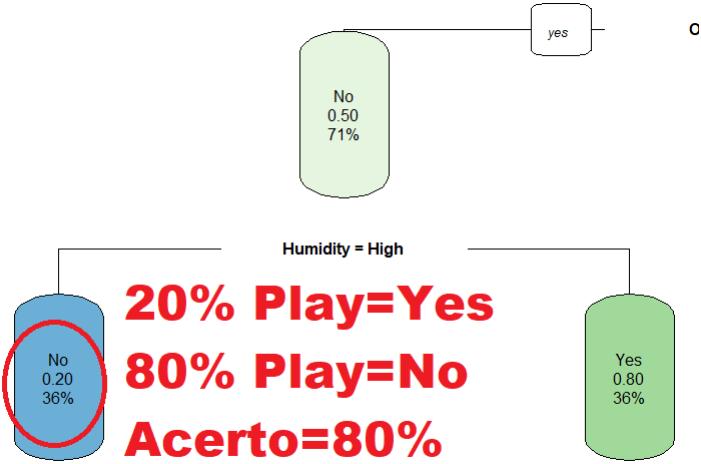


Entendendo rpart.plot



Entendendo rpart.plot

Entendendo rpart.plot



Árvores com `train()`

- Carregar banco de dados. Separar amostras teste/treino.

```
> load("heart_disease.rda")
> set.seed(100)
> inTrain <- createDataPartition(data$hd, p=0.7,list=F)
> training <- data[inTrain,]
> testing <- data[-inTrain,]
```

- Treinar o modelo.

```
> modFit<-train(hd~., method="rpart", data=training)
```

- Observe que através do `train()`, são testados vários valores de cp (complexity parameter), e é eleito aquele com maior taxa de acurácia.

```
> modFit
```

Árvores com train()

```
CART

208 samples
 13 predictor
  2 classes:  'Healthy', 'Unhealthy'

No pre-processing
Resampling:  Bootstrapped (25 reps)
Summary of sample sizes:  208, 208, 208, 208, ...
Resampling results across tuning parameters:

  cp         Accuracy  Kappa
  0.0312500 0.7392669 0.4702968
  0.0625000 0.7251036 0.4430745
  0.4895833 0.6477880 0.2759826

Accuracy was used to select the optimal model using the largest
value.
```

Árvores com train()

```
CART

208 samples
 13 predictor
  2 classes:  'Healthy', 'Unhealthy'

No pre-processing
Resampling:  Bootstrapped (25 reps)
Summary of sample sizes:  208, 208, 208, 208, ...
Resampling results across tuning parameters:

  cp         Accuracy  Kappa
  0.0312500  0.7392669 0.4702968
  0.0625000  0.7251036 0.4430745
  0.4895833  0.6477880 0.2759826

Accuracy was used to select the optimal model using the largest
value.
```

Árvores com `train()`

- Aplicar o modelo na amostra teste.
  `> hdFit<-predict(modFit,testing)`
- Avaliar o erro na amostra treino.
  `> confusionMatrix(testing$hd,hdFit)`
- Plota árvore.
  `> rpart.plot(modFit$finalModel)`

Árvores com `train()`

- Limitação de utilizar `rpart` com `train`: o único argumento que podemos alterar é o cp (complexity parameter).
  `> modelLookup("rpart")`
- Para fixa-lo, utilizamos os comandos a seguir:
  `> rpart.grid <- expand.grid(.cp=0.2)`

  `> modFit<-train(hd∼., method="rpart", data=training, tuneGrid=rpart.grid)`

  `> modFit`