INSTITUTO FEDERAL
Santa Catarina
Câmpus São José

# Aplicação de Técnicas de Aprendizagem de Máquina utilizando R

Mário de Noronha Neto e Richard Demo Souza

# Classificação utilizando Árvores e Regras de Decisão

- Estes métodos tem o objetivo de transformar decisões complexas em um conjunto de escolhas simples

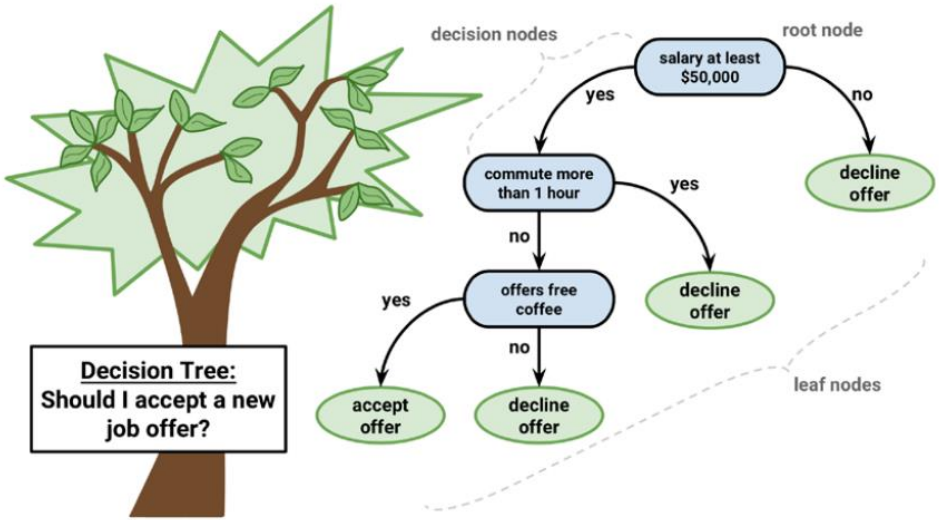
- As informações dos dados podem ser apresentadas na forma de estruturas lógicas, as quais podem ser compreendidas sem detalhamento de informações estatísticas

- Estas características tornam estes métodos interessantes para aplicações em estratégias de negócios e melhoria de processos, por exemplo, ou em situações que necessitem transparência nas tomadas de decisão.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Árvore de Decisão: Conceito básico

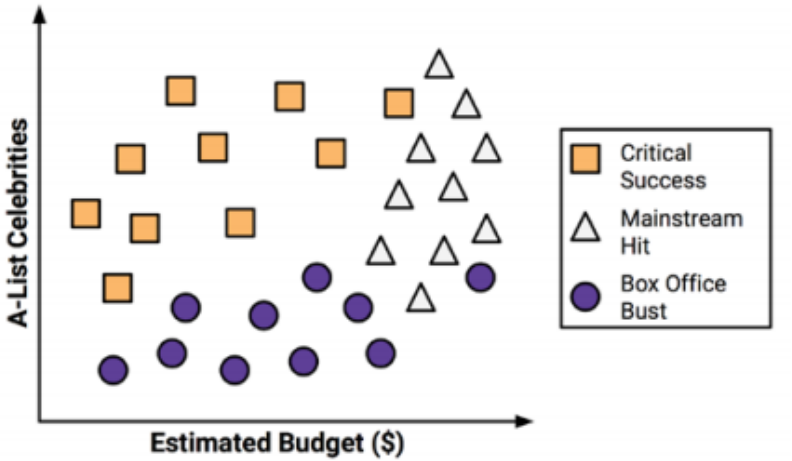
Utilizar uma **estrutura de árvore** para modelar a relação entre características de entrada e possíveis saídas.




Fonte: Brett Lantz – Machine Learning with R – Second Edition

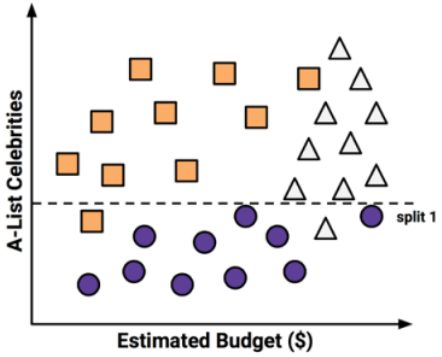
# Método ‘Dividir e Conquistar’

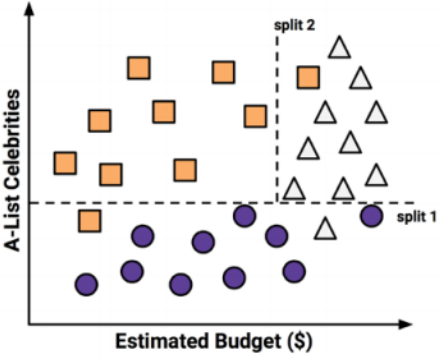
As árvores de decisão são construídas utilizando um particionamento recursivo, também conhecido como método **dividir** e **conquistar**.

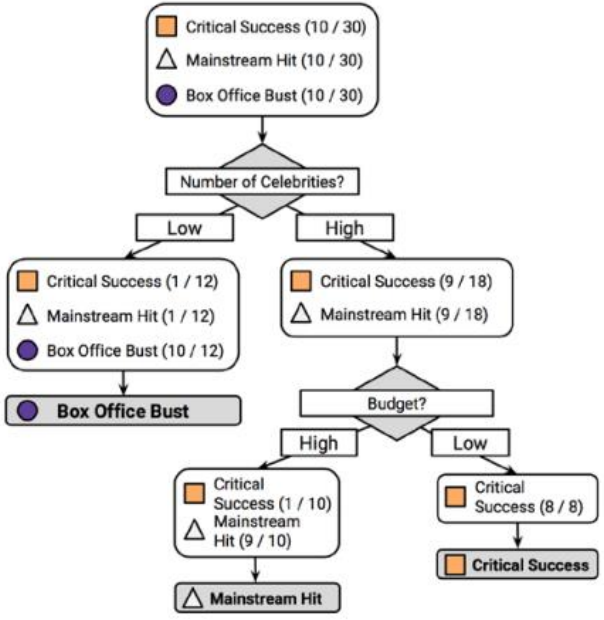



Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Dividir e Conquistar




Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus São José

# Algoritmo C5.0 - Árvore de Decisão

O C5.0 é um dos algoritmos mais conhecidos na implementação da técnica de árvore de decisão e se tornou um padrão na indústria.

| Strengths | Weaknesses |
|---|---|
| • An all-purpose classifier that does well on most problems<br>• Highly automatic learning process, which can handle numeric or nominal features, as well as missing data<br>• Excludes unimportant features<br>• Can be used on both small and large datasets<br>• Results in a model that can be interpreted without a mathematical background (for relatively small trees)<br>• More efficient than other complex models | • Decision tree models are often biased toward splits on features having a large number of levels<br>• It is easy to overfit or underfit the model<br>• Can have trouble modeling some relationships due to reliance on axis-parallel splits<br>• Small changes in the training data can result in large changes to decision logic<br>• Large trees can be difficult to interpret and the decisions they make may seem counterintuitive |

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Algoritmo C5.0 - Árvore de Decisão

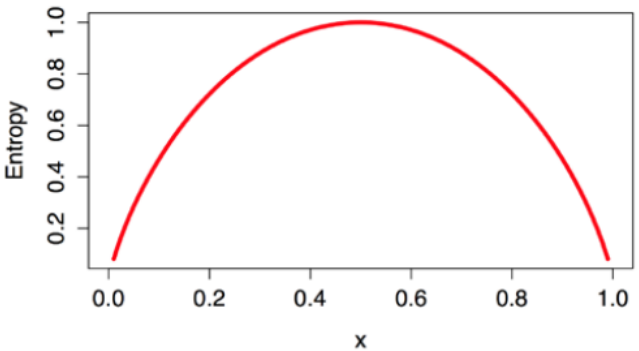
**Escolhendo a melhor forma para dividir/particionar:** O C5.0 utiliza o conceito de **entropia** para determinar o grau de *pureza* de um conjunto dos dados de entrada. Conjuntos com valores elevados de entropia são muito diversos e trazem pouca informação sobre outros elementos que também podem estar no conjunto. Desta forma, o algoritmo busca reduzir a entropia para realizar a divisão, aumentando com isso a homogeneidade do conjunto.

$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \, log_2(p_i)$$

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Algoritmo C5.0 - Árvore de Decisão

```r
# calculate entropy of a two-class segment
curve(-x * log2(x) - (1 - x) * log2(1 - x),
      col = "red", xlab = "x", ylab = "Entropy", lwd = 4)
```




F: Característica

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

n. de partições

$$\text{Entropy}(S) = \sum_{i=1}^{n} w_i \, \text{Entropy}(P_i)$$

proporção de elementos nas partições

**Ganho de informação**: Diferença entre a entropia antes e depois de uma divisão. Como após uma divisão são criados mais de uma partição, o cálculo da entropia deve ser considerado em todas as partições, ponderadas pela quantidade de elementos em cada partição.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Algoritmo C5.0 - Árvore de Decisão

- Quanto maior o ganho de informação, mais homogêneos serão os grupos após uma determinada divisão. Se o ganho de informação for zero, não há redução da entropia na ação de divisão realizada.

- A árvore de decisão pode continuar crescendo até que cada exemplo seja perfeitamente classificado. Entretanto, isto tornará a árvore muito grande e o modelo muito específico.

- Uma estratégia é parar com a divisão quando se chega a um determinado número de decisões ou quando os *nós* de decisão contém apenas um número pequeno de elementos. Esta técnica é chamada de *early stopping* ou *pre-pruning*

- Outra alternativa, chamada *post-pruning*, envolve crescer a árvore intencionalmente para depois podá-la, reduzindo seu tamanho para níveis mais apropriados.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Exemplo: Identificação de risco em empréstimo bancários

Passo 1: Coleta de dados

**Dataset utilizado**: http://archive.ics.uci.edu/ml

O *dataset* utilizado neste exemplo (*credit.csv*) foi ligeiramente modificado com relação ao original para eliminar algumas etapas de pré-processamento. O *dataset* possui 1000 exemplos de empréstimos realizados por um determinado banco contendo um conjunto de características numéricas e nominais sobre o empréstimo e o beneficiário.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Passo 2: Explorando e preparando os dados

```
## Example: Identifying Risky Bank Loans ----
## Step 2: Exploring and preparing the data ----
credit <- read.csv("credit.csv")
str(credit)
```

```
'data.frame':1000 obs. of  17 variables:
 $ checking_balance : Factor w/ 4 levels "< 0 DM","> 200 DM",..
 $ months_loan_duration: int  6 48 12 ...
 $ credit_history     : Factor w/ 5 levels "critical","good",..
 $ purpose            : Factor w/ 6 levels "business","car",..
 $ amount             : int  1169 5951 2096 ...
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Passo 2: Explorando e preparando os dados

```
table(credit$checking_balance)
table(credit$savings_balance)

summary(credit$months_loan_duration)
summary(credit$amount)

table(credit$default)
```

```
> table(credit$checking_balance)

    < 0 DM    > 200 DM 1 - 200 DM     unknown
       274          63         269         394
> table(credit$savings_balance)

     < 100 DM     > 1000 DM  100 - 500 DM 500 - 1000 DM       unknown
          603            48           103             63           183

> summary(credit$months_loan_duration)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    4.0    12.0    18.0    20.9    24.0    72.0
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18424
```

```
> table(credit$default)

 no yes
700 300
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Passo 2: Explorando e preparando os dados

As amostras do *dataset* estão distribuídas em uma ordem específica gerada pelo banco. Como é importante que as sequências de teste e treinamento contenham amostras representativas do *dataset,* as amostras que irão compor essas sequências serão sorteadas de forma aleatória.

```
# create a random sample for training and test data
# use set.seed to use the same random number sequence as the tutorial
set.seed(123)
train_sample <- sample(1000, 900)
credit_train <- credit[train_sample, ]
credit_test  <- credit[-train_sample, ]

# check the proportion of class variable
prop.table(table(credit_train$default))
prop.table(table(credit_test$default))
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Passo 3: Treinando o modelo

Para a classificação utilizando o método da árvore de decisão, utilizaremos a função *c5.0 ()* do pacote *C50*

**C5.0 decision tree syntax**

using the `C5.0()` function in the `C50` package

**Building the classifier:**

```
m <- C5.0(train, class, trials = 1, costs = NULL)
```

- `train` is a data frame containing training data
- `class` is a factor vector with the class for each row in the training data
- `trials` is an optional number to control the number of boosting iterations (set to `1` by default)
- `costs` is an optional matrix specifying costs associated with various types of errors

The function will return a C5.0 model object that can be used to make predictions.

**Making predictions:**

```
p <- predict(m, test, type = "class")
```

- `m` is a model trained by the `C5.0()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.
- `type` is either `"class"` or `"prob"` and specifies whether the predictions should be the most probable class value or the raw predicted probabilities

The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

**Example:**

```
credit_model <- C5.0(credit_train, loan_default)
credit_prediction <- predict(credit_model,
  credit_test)
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Passo 3: Treinando o modelo

```
## Step 3: Training a model on the data ----
# build the simplest decision tree
library(C50)
credit_model <- C5.0(credit_train[-17], credit_train$default)

# display simple facts about the tree
credit_model
```

```
> credit_model

Call:
C5.0.default(x = credit_train[-17], y = credit_train$default)

Classification Tree
Number of samples: 900
Number of predictors: 16

Tree size: 57

Non-standard options: attempt to group attributes
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Analisando o modelo

```
# display detailed information about the tree
summary(credit_model)

C5.0 [Release 2.07 GPL Edition]
-------------------------------

Class specified by attribute `outcome'

Read 900 cases (17 attributes) from undefined.data

Decision tree:

checking_balance in {> 200 DM,unknown}: no (412/50)
checking_balance in {< 0 DM,1 - 200 DM}:
:...credit_history in {perfect,very good}: yes (59/18)
    credit_history in {critical,good,poor}:
    :...months_loan_duration <= 22:
        :...credit_history = critical: no (72/14)
        :   credit_history = poor:
        :   :...dependents > 1: no (5)
        :   :   dependents <= 1:
        :   :   :...years_at_residence <= 3: yes (4/1)
        :   :       years_at_residence > 3: no (5/1)
```

O modelo pode gerar regras que não fazem muito sentido!

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Analisando o modelo

```
  Decision Tree
----------------
Size      Errors
  56  133(14.8%)   <<

 (a)   (b)    <-classified as
----  ----
 598    35    (a): class no
  98   169    (b): class yes
```

Este método tende a deixar o modelo muito específico para a sequência de treinamento. Desta forma, a taxa de erro reportada nesta função é bem otimista!

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Passo 4: Avaliando o desempenho do modelo

```
## Step 4: Evaluating model performance ----
# create a factor vector of predictions on test data
credit_pred <- predict(credit_model, credit_test)

# cross tabulation of predicted versus actual classes
library(gmodels)
CrossTable(credit_test$default, credit_pred,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual default', 'predicted default'))
```

| | predicted default | | |
|---|---|---|---|
| actual default | no | yes | Row Total |
| no | 59 | 8 | 67 |
| | 0.590 | 0.080 | |
| yes | 19 | 14 | 33 |
| | 0.190 | 0.140 | |
| Column Total | 78 | 22 | 100 |

O banco vai perder dinheiro!

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Realizar testes para outras configurações

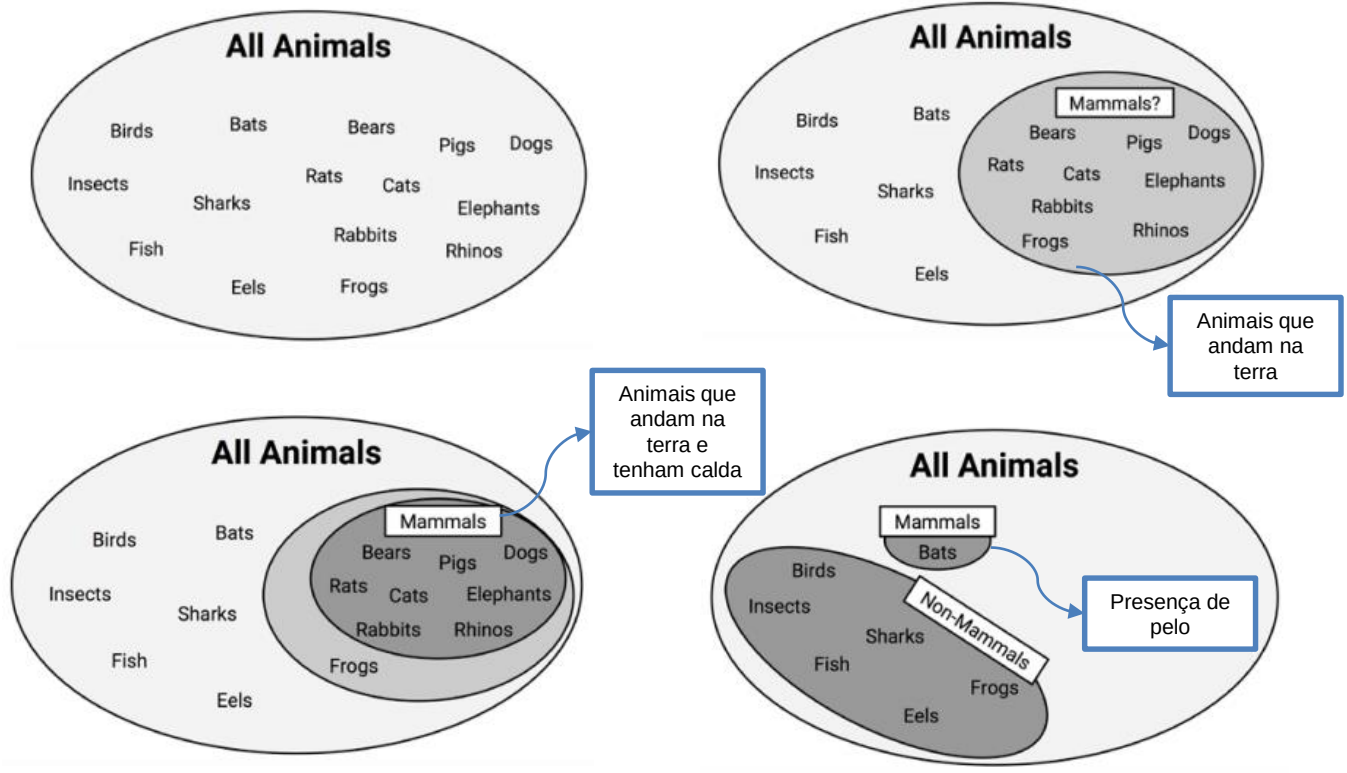
- O parâmetro *‘Trials’* possibilita melhorar o desempenho do algoritmo C5.0 fazendo com que mais de uma árvore de decisão seja construída. Neste caso, a tomada de decisão será feita com base no voto de cada uma das árvores, fazendo com que a classe mais votada seja a escolhida. Experimente avaliar o desempenho com *trials = 10 e 20.*

# Classificação por Regras de Decisão

As regras de decisão podem representar os dados na forma de instruções lógicas *if-else* as quais atribuem uma classe a um exemplo não rotulado. Ao contrário da técnica de árvore de decisão, que é aplicada de *'cima para baixo'* por uma série de decisões, as regras de decisão podem ser interpretadas/lidas em um formato mais simples, pois podem ser especificadas em termos de ações antecedentes e consequentes

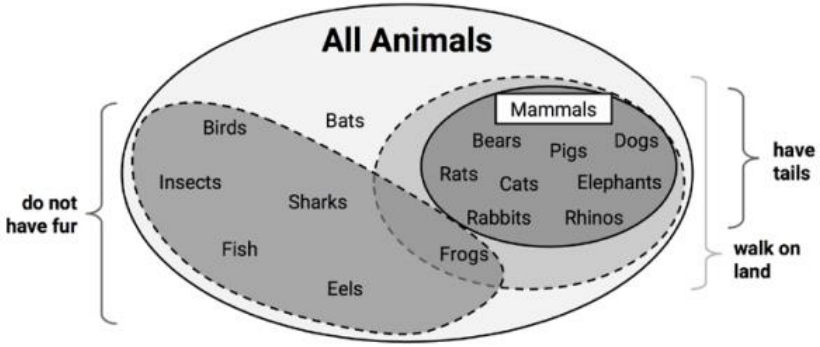
Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Separar e Conquistar




Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Diferenças entre Árvores e Regras de Decisão




**Regras de decisão**

1. O animal que anda na terra e possue cauda é mamíferos
2. Se o animal não possui pelo, não é mamífero
3. Caso contrário, o anima é mamífero

**Árvores de decisão**

1. Se o animal anda na terra e possue pelo, então é mamífero
2. Se o animal anda na terra e não tem pelo, então não é mamífero
3. Se o animal não anda na terra e tem pelo, então é mamífero
4. Se o animal não anda na terra e não tem pelo, então não é mamífero

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Diferenças entre Árvores e Regras de Decisão

**Árvores de decisão:** a partição criada pela divisão no método divide e conquista não pode ser reconquistada, apenas subdividida. Desta forma, a árvore fica limitada pelo histórico de divisões passadas.

**Regras de decisão:** o método separar e conquistar identifica a regra e qualquer exemplo não coberto pelas regras atuais pode ser reconquistado.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Algoritmo 1R (One Rule)

| Animal | Travels By | Has Fur | Mammal |
|---|---|---|---|
| Bats | Air | Yes | Yes |
| Bears | Land | Yes | Yes |
| Birds | Air | No | No |
| Cats | Land | Yes | Yes |
| Dogs | Land | Yes | Yes |
| Eels | Sea | No | No |
| Elephants | Land | No | Yes |
| Fish | Sea | No | No |
| Frogs | Land | No | No |
| Insects | Air | No | No |
| Pigs | Land | No | Yes |
| Rabbits | Land | Yes | Yes |
| Rats | Land | Yes | Yes |
| Rhinos | Land | No | Yes |
| Sharks | Sea | No | No |

**Full Dataset**

| Travels By | Predicted | Mammal | |
|---|---|---|---|
| Air | No | Yes | ✖ |
| Air | No | No | |
| Air | No | No | |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Land | Yes | No | ✖ |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Land | Yes | Yes | |
| Sea | No | No | |
| Sea | No | No | |
| Sea | No | No | |

**Rule for "Travels By"**
**Error Rate = 2 / 15**

| Has Fur | Predicted | Mammal | |
|---|---|---|---|
| No | No | No | |
| No | No | No | |
| No | No | Yes | ✖ |
| No | No | No | |
| No | No | No | |
| No | No | No | |
| No | No | Yes | ✖ |
| No | No | Yes | ✖ |
| No | No | No | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |
| Yes | Yes | Yes | |

**Rule for "Has Fur"**
**Error Rate = 3 / 15**

Utiliza uma única regra, a mais relevante. Neste exemplo a regra é baseada em ***'Travels By'.*** Esta técnica é fácil de entender, eficiente em *datasets* grandes e ruidosos, além de produzir um modelo mais simples quando comparado com o da árvore de decisão.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Algoritmo RIPPER

**Descrição:** O algoritmo RIPPER pode criar regras mais complexas do que o algoritmo 1R, pois leva em consideração mais de uma característica do *dataset*. Isto significa que ele consegue criar regras com múltiplos antecedentes. Por outro lado, as regras podem, rapidamente, se tornarem mais difíceis de serem interpretadas.

**Funcionamento:** O algoritmo RIPPER utiliza o método *separar e conquistar* para adicionar regras até que classifique perfeitamente um subconjunto ou que este fique sem atributos para realizar uma nova divisão *(passo 1)*. Similar à técnica de árvore de decisão, o ganho de informação é utilizado para realizar a próxima separação. Quando o aumento da especificidade não reduz mais a entropia, a regra é imediatamente podada *(passo 2*). Os passos 1 e 2 são executados repetidamente até que um critério de parada seja alcançado, para então ser realizado um processo de otimização.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Exemplo: Identificação de cogumelos venenosos

Passo 1: Coleta de dados

**Dataset utilizado**: http://archive.ics.uci.edu/ml

O *dataset* utilizado neste exemplo possui 8124 amostras de cogumelos de 23 espécies diferentes. Casa uma é classificada como sendo *"possivelmente venenoso"* e *"não recomendado para comer".* Cada amostra possui 22 características.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Passo 2: Explorando e preparando os dados

```
## Step 2: Exploring and preparing the data ----
mushrooms <- read.csv("mushrooms.csv", stringsAsFactors = TRUE)

# examine the structure of the data frame
str(mushrooms)

# drop the veil_type feature
mushrooms$veil_type <- NULL

# examine the class distribution
table(mushrooms$type)
```

```
> table(mushrooms$type)

  edible poisonous
    4208      3916
```

Como todas as características são nominais, podemos deixar a conversão habilitada

Nesta característica todos os dados estão com o mesmo nível (*partial).* Como esta característica não varia entre as amostras, não fará diferença na predição.

Para os propósitos deste exemplo, podemos considerar que o conjunto de 8214 amostras representar todos os possíveis cogumelos selvagens. Isto significa que não será necessário separar um subconjunto de amostras para realização da etapa de testes. O objetivo é identificar novas espécies de cogumelos e sim classificar os existentes.

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Passo 3: Treinando o modelo

Para a classificação utilizaremos inicialmente o algoritmo 1R do pacote *RWeka* disponível na função *OneR ()*.

**1R classification rule syntax**

using the `OneR()` function in the `RWeka` package

**Building the classifier:**

```
m <- OneR(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a 1R model object that can be used to make predictions.

**Making predictions:**

```
p <- predict(m, test)
```

- `m` is a model trained by the `OneR()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

**Example:**

```
mushroom_classifier <- OneR(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

```
# train OneR() on the data
mushroom_1R <- OneR(type ~ ., data = mushrooms)
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

INSTITUTO FEDERAL
Santa Catarina
Câmpus
São José

# Passo 4: Avaliando o desempenho do modelo

```
## Step 4: Evaluating model performance ----
mushroom_1R
```

```
> mushroom_1R
odor:
        almond  -> edible
        anise   -> edible
        creosote        -> poisonous
        fishy   -> poisonous
        foul    -> poisonous
        musty   -> poisonous
        none    -> edible
        pungent -> poisonous
        spicy   -> poisonous
(8004/8124 instances correct)
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Passo 4: Avaliando o desempenho do modelo

```
## Step 4: Evaluating model performance ----
summary(mushroom_1R)
```

```
=== Summary ===

Correctly Classified Instances        8004               98.5229 %
Incorrectly Classified Instances       120                1.4771 %
Kappa statistic                          0.9704
Mean absolute error                      0.0148
Root mean squared error                  0.1215
Relative absolute error                  2.958  %
Root relative squared error             24.323  %
Total Number of Instances             8124

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = edible
  120 3796 |    b = poisonous
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Treinando e avaliando o desempenho do modelo com o algoritmo RIPPER

Nesta etapa utilizaremos a função *Jrip ()*, uma versão do algoritmo RIPPER baseada em Java, disponível no pacote *Rweka.*

**RIPPER classification rule syntax**

using the `JRip()` function in the `RWeka` package

**Building the classifier:**

```
m <- JRip(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a RIPPER model object that can be used to make predictions.

**Making predictions:**

```
p <- predict(m, test)
```

- `m` is a model trained by the `JRip()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

**Example:**

```
mushroom_classifier <- JRip(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Treinando e avaliando o desempenho do modelo com o algoritmo RIPPER

```
## Evaluating model performance ----
mushroom_JRip <- JRip(type ~ ., data = mushrooms)
mushroom_JRip
```

```
JRIP rules:
===========

(odor = foul) => type=poisonous (2160.0/0.0)
(gill_size = narrow) and (gill_color = buff) => type=poisonous (1152.0/0.0)
(gill_size = narrow) and (odor = pungent) => type=poisonous (256.0/0.0)
(odor = creosote) => type=poisonous (192.0/0.0)
(spore_print_color = green) => type=poisonous (72.0/0.0)
(stalk_surface_below_ring = scaly) and (stalk_surface_above_ring = silky) => type=poisonous
 (68.0/0.0)
(habitat = leaves) and (cap_color = white) => type=poisonous (8.0/0.0)
(stalk_color_above_ring = yellow) => type=poisonous (8.0/0.0)
 => type=edible (4208.0/0.0)

Number of Rules : 9
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Treinando e avaliando o desempenho do modelo com o algoritmo RIPPER

```
## Evaluating model performance ----
summary(mushroom_JRip)
```

```
=== Summary ===

Correctly Classified Instances        8124               100       %
Incorrectly Classified Instances         0                 0       %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0       %
Root relative squared error              0       %
Total Number of Instances             8124

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = edible
    0 3916 |    b = poisonous
```

Fonte: Brett Lantz – Machine Learning with R – Second Edition

# Desempenho utilizando a técnica de Árvore de Decisão

- Utilize a técnica de árvore de decisão para o *dataset* deste exemplo.