Aula 4 - Machine Learning

Introdução ao aprendizado de máquina - UEMA 2025

Thiago S. F. Silva

2025-12-15

Parte I - Introdução

Revisitando: *machine learning* vs. estatística

- Focado em **predição**
- Sem valor-p, intervalo de confiança, pressuposições, etc.
- Pouca ou nenhuma **interpretabilidade** do modelo.
- Sem preocupação com o 'mínimo modelo' ou parsimônia.
- Os melhores algoritmos de ML não requerem linearidade, normalidade, independência, etc.
- Bons métodos estatísticos (ex. modelos lineares) são inferiores para ML.

Exemplo: Boston Housing Price

```
'data.frame':   506 obs. of  14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ..
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Regressão Linear vs Random Forests

```
# A tibble: 2 x 4
  model .metric .estimator .estimate
  <chr> <chr>   <chr>          <dbl>
1 OLS   rmse    standard        6.03
2 RF    rmse    standard        3.15
```

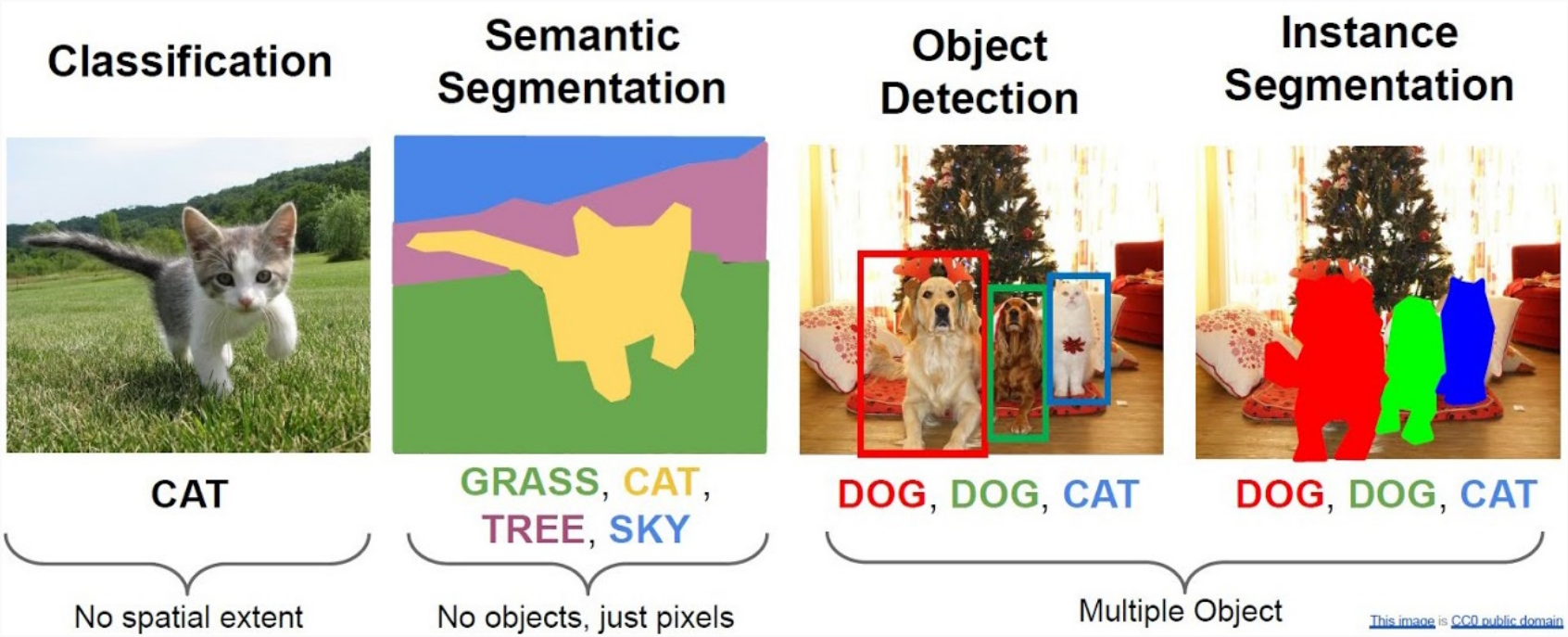
Terminologia

- **Treinamento:** cálculo dos parâmetros do modelo a partir dos dados.
- **Tuning (otimização):** otimização dos *hyperparâmetros* do modelo.
- **Validação:** avaliação do modelo durante o *tuning*.
- **Teste:** avaliação final do modelo usando dados de teste.
- **Target (Alvo):** variável dependente / alvo da predição.
- **Feature (feição):** variáveis preditoras/variáveis independentes.

Terminologia

- **Classificação:** qualquer problema/modelo que produz resultados *categóricos*.
- **Regressão:** qualquer problema/modelo que produz resultados *contínuos*.

No caso específico de visão computacional:



Parte II - Aplicando Machine Learning

Machine learning no R

- O pacote `tidymodels`
- O livro 'Tidy Models with R'.
- O livro mais antigo `Applied Predictive Modeling with R` ainda é bom para *entender* ML.
- O website do pacote Python Scikit-learn é excelente para explicações sobre algoritmos específicos, independente da linguagem.

Os passos de uma análise de ML

1. ~~Preparação dos dados~~
2. ~~Análise descritiva~~
3. ~~Visualização de dados~~
4. **Divisão dos dados em *treinamento* e *teste***
5. **Escolha do algoritmo**
6. **Otimização e validação do modelo**
7. **Teste do modelo**

Algoritmos de Machine Learning

Existem literalmente dezenas de algoritmos de machine learning.

Vamos focar nos mais comuns:

- Árvores de decisão (*decision trees*)
- *Support Vector Machine* (SVM)
- Métodos de *Ensemble*:
  - Random Forests (*bagging*)
  - LightGBM (*boosting*)

Árvores de Decisão

Um dos métodos mais 'básicos' de ML. Suas vantagens são:

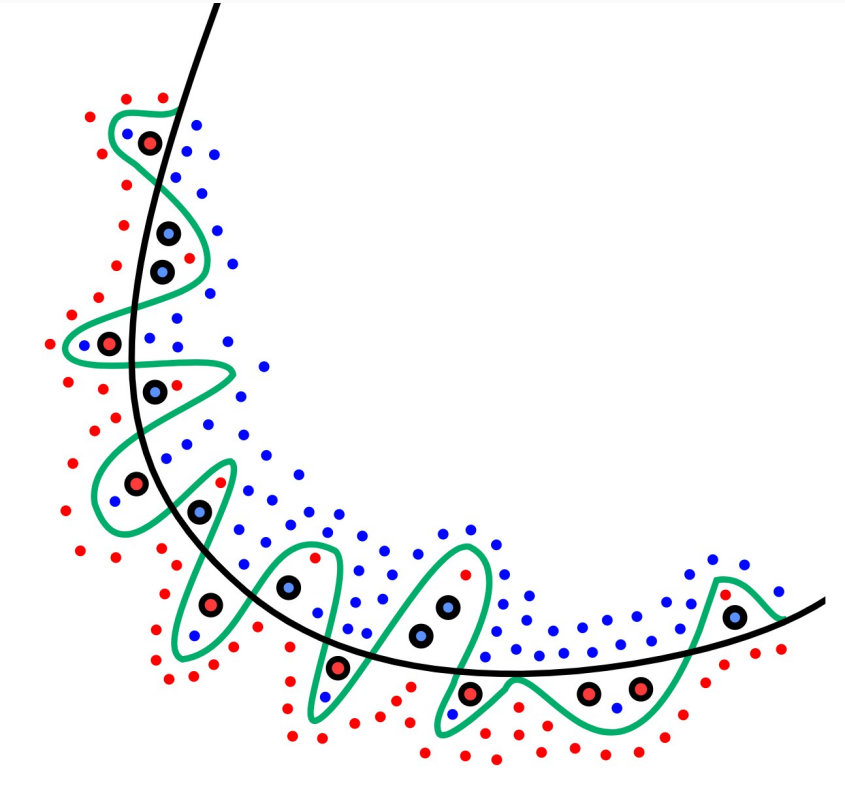
- Fáceis de compreender
- Robustos com relação aos dados
- Não-Paramétricos
- Funcionam pra Classificação e Regressão.

Desvantagens:

- Árvores de decisão são propensas ao *overfitting*, e requerem métodos de "poda" (*pruning*) para manter a generalidade.
- As predições não são contínuas.
- Sensíveis a dados desbalanceados.

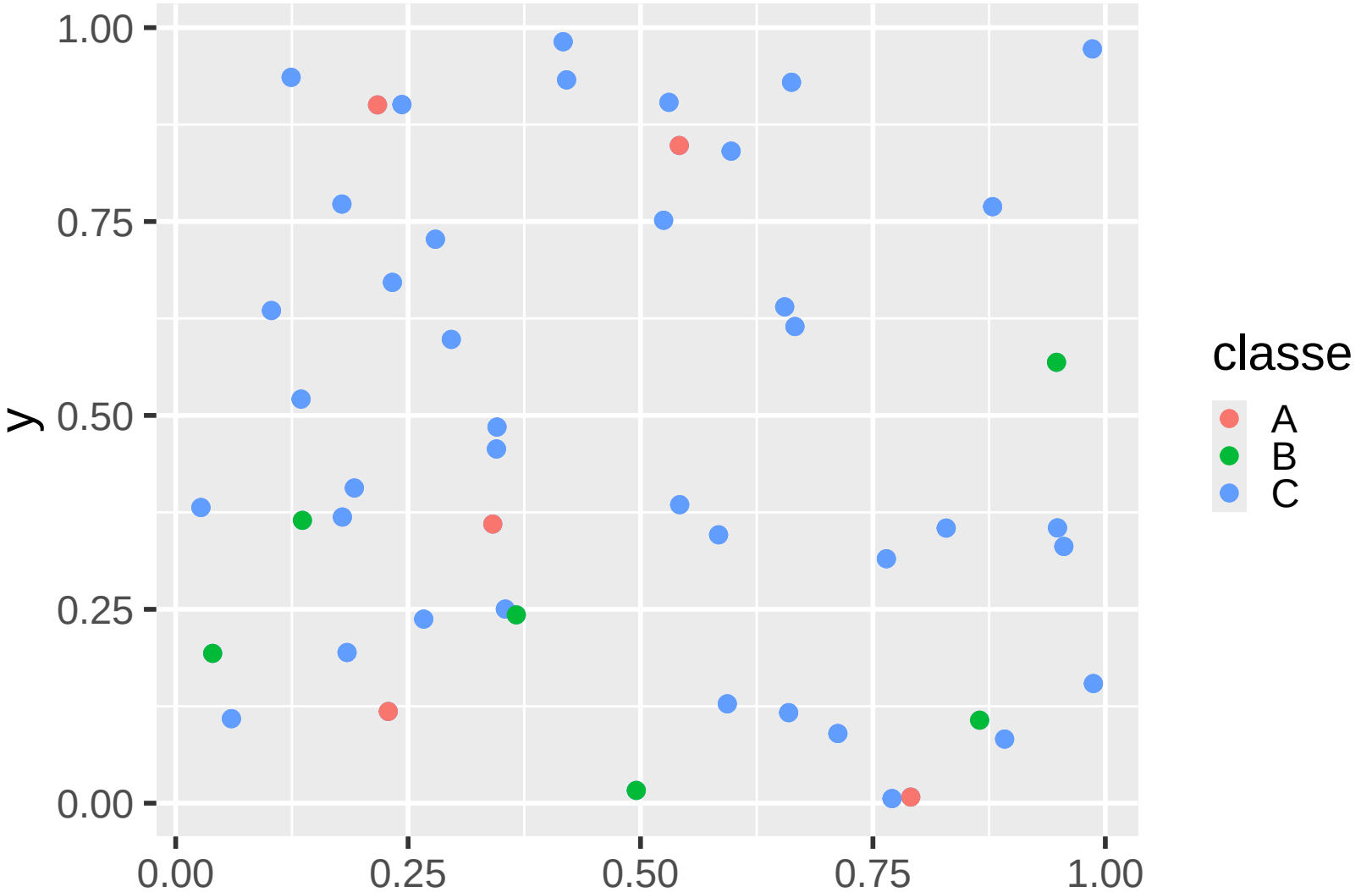
Overfitting

Algoritmos de ML podem aprender "bem demais". Isso é chamado de *overfitting* (ajuste excessivo). Normalmente detectado por uma acurácia alta durante a otimização, e baixa durante o teste.

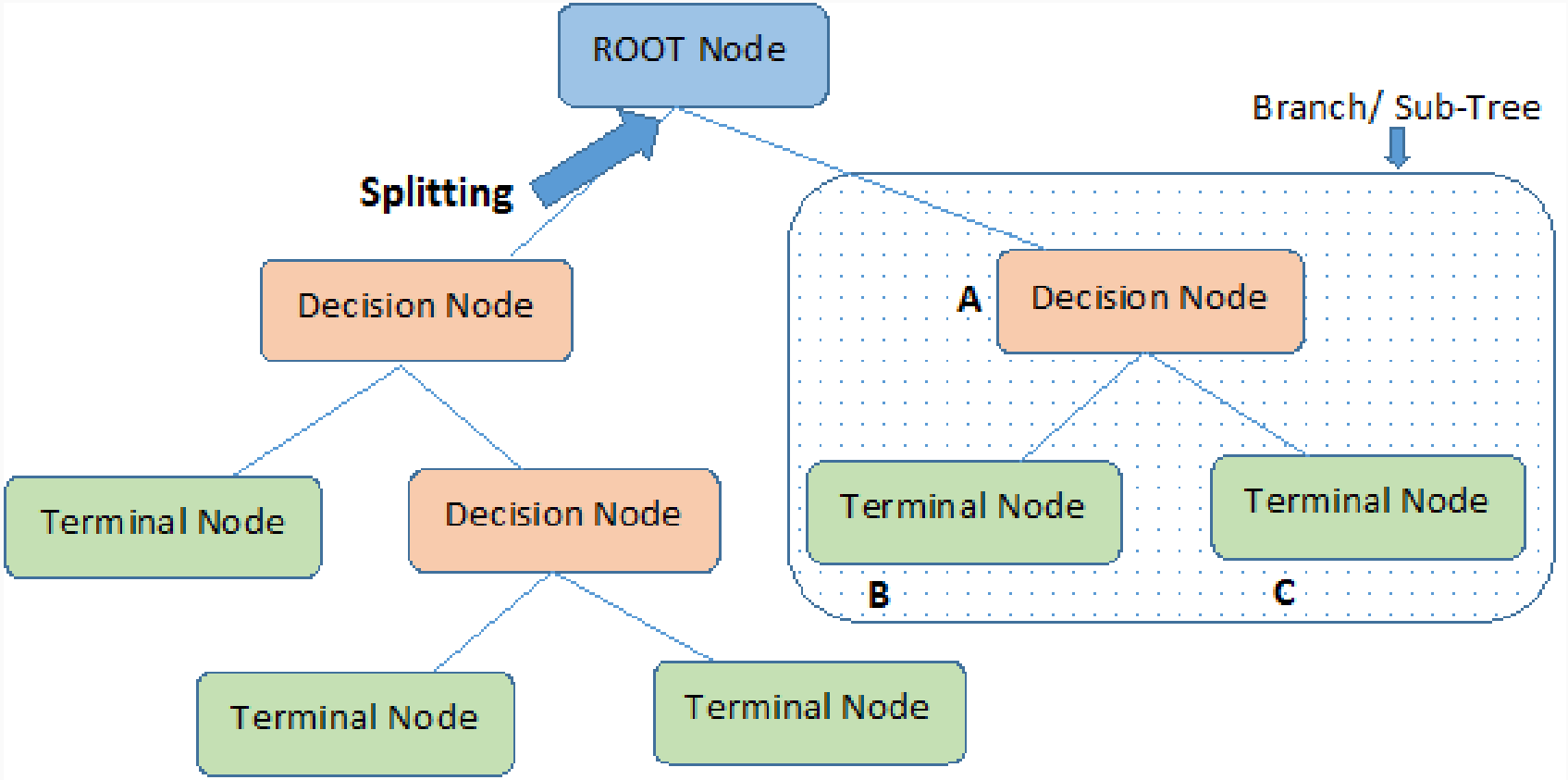


Dados desbalanceados

Quando uma ou mais classes dominam a quantidade de amostras sobre as outras.



Árvores de Decisão



https://www.youtube.com/watch?v=ZVR2Way4nwQ

Classificando o dataset *iris*

`Petal.Width`, `Petal.Lenght`, `Sepal.Width`, `Sepal.Length` são as nossas *features*.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Classificando o dataset *iris*

Nosso nó inicial contém todas as amostras. Queremos achar um limiar para um dos quatro *features* que maximiza a separação entre os grupos.

Como 'medir' essa separação?

Pureza do Nó (Node Purity): quão 'misturado' é o dataset após a separação?

Classificando o dataset *iris*

Exemplos (C = número de classes):

Índice de Pureza de Gini

$Gini = 1 - \sum_{i=1}^{C} p_i * (1 - p_i)$

- Máximo de Pureza: 0
- Mínimo: $(C-1)/C$

Índice de Entropia de Shannon

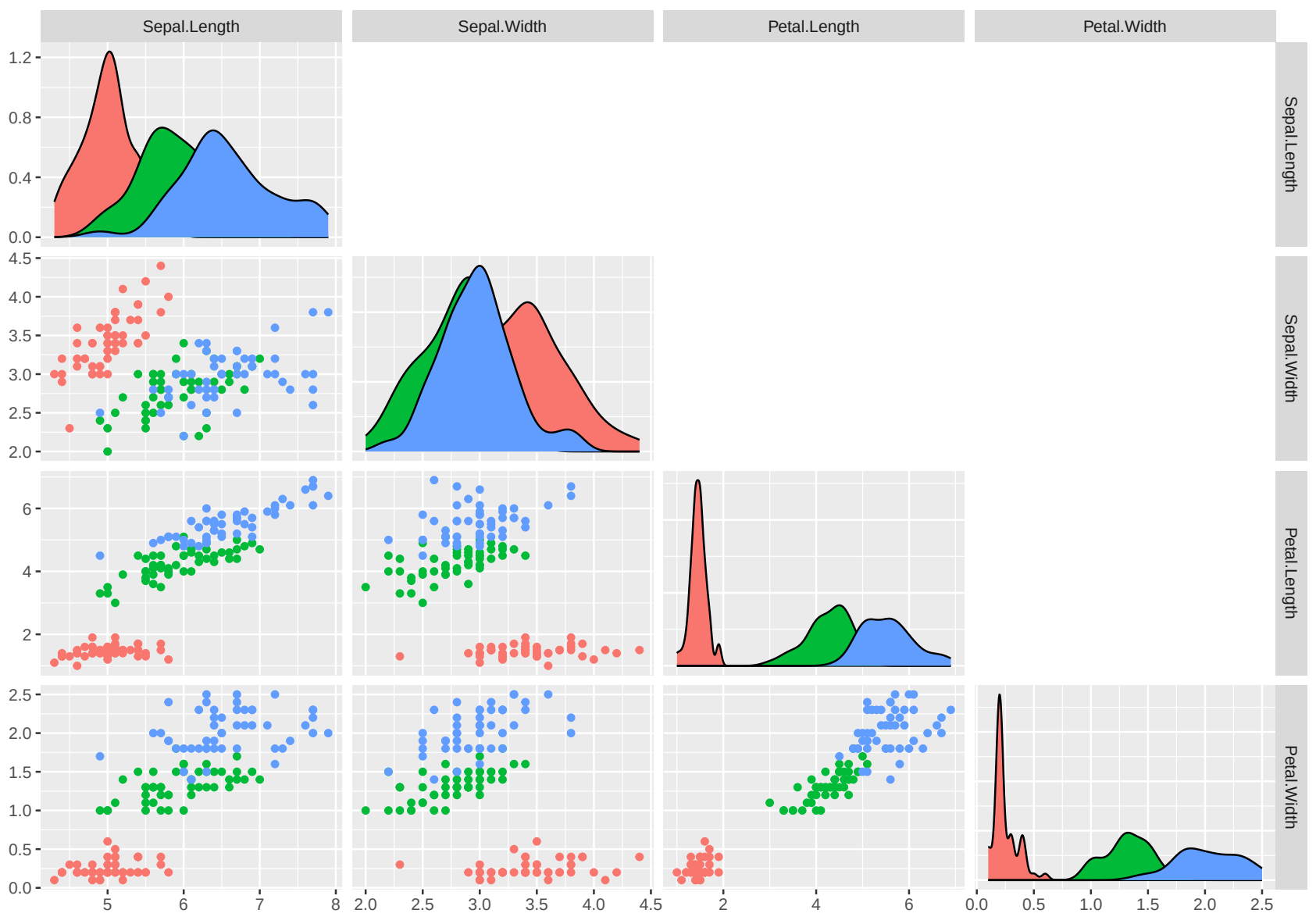
$Entropia = -\sum_{i=1}^{C} p_i * log(p_i)$

- Mínimo de Entropia: 0
- Máximo: $log(C)$

Classificando o dataset *iris*

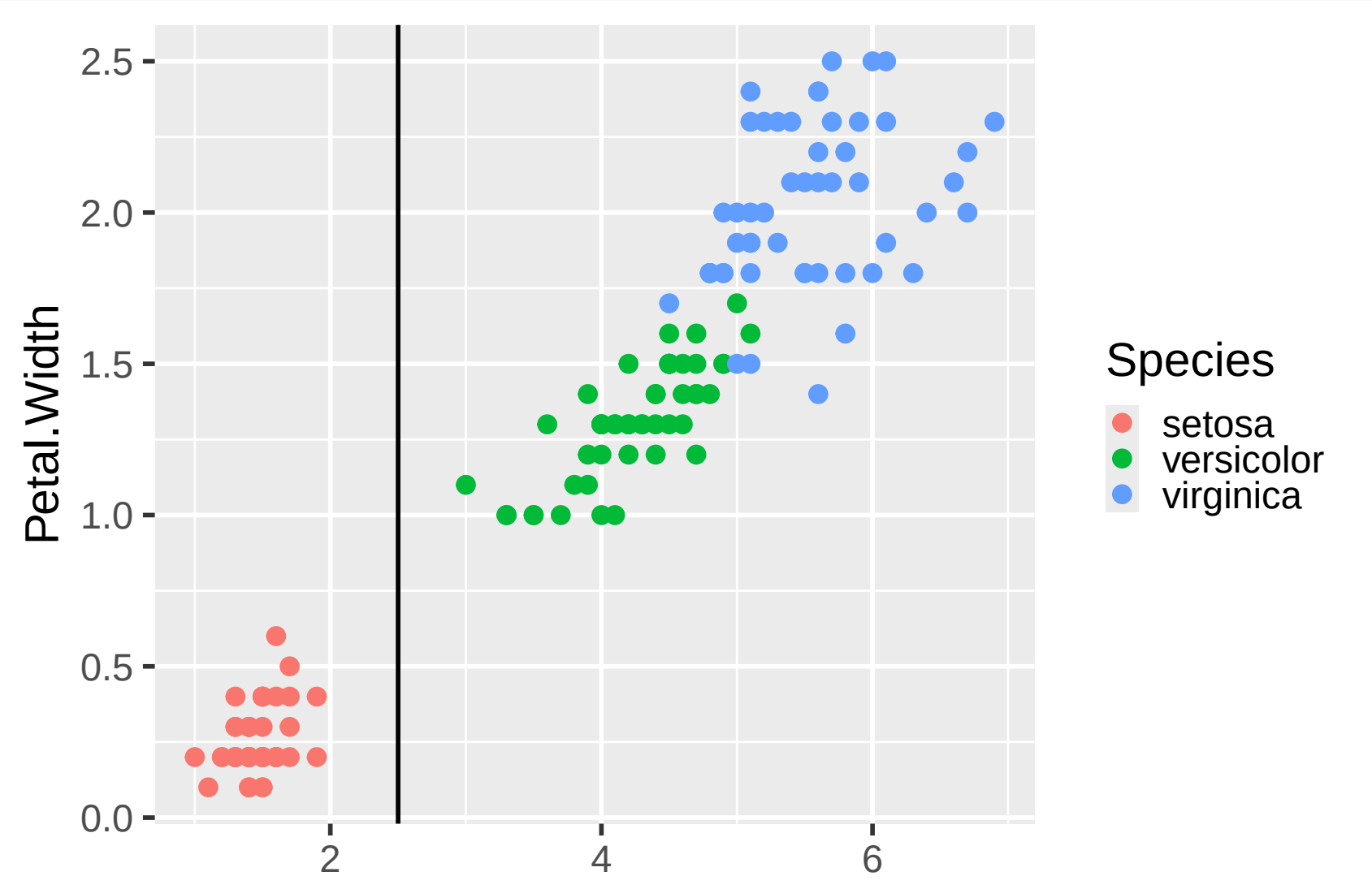
Então podemos testar diferentes limiares de separação para diferentes *features*, e escolher aquele que maximiza a pureza (minimiza o índice Gini).

Classificando o dataset *iris*



Classificando o dataset *iris*

Para separar a espécie *setosa*, basta um limiar de `Petal.Length > 2.5`.



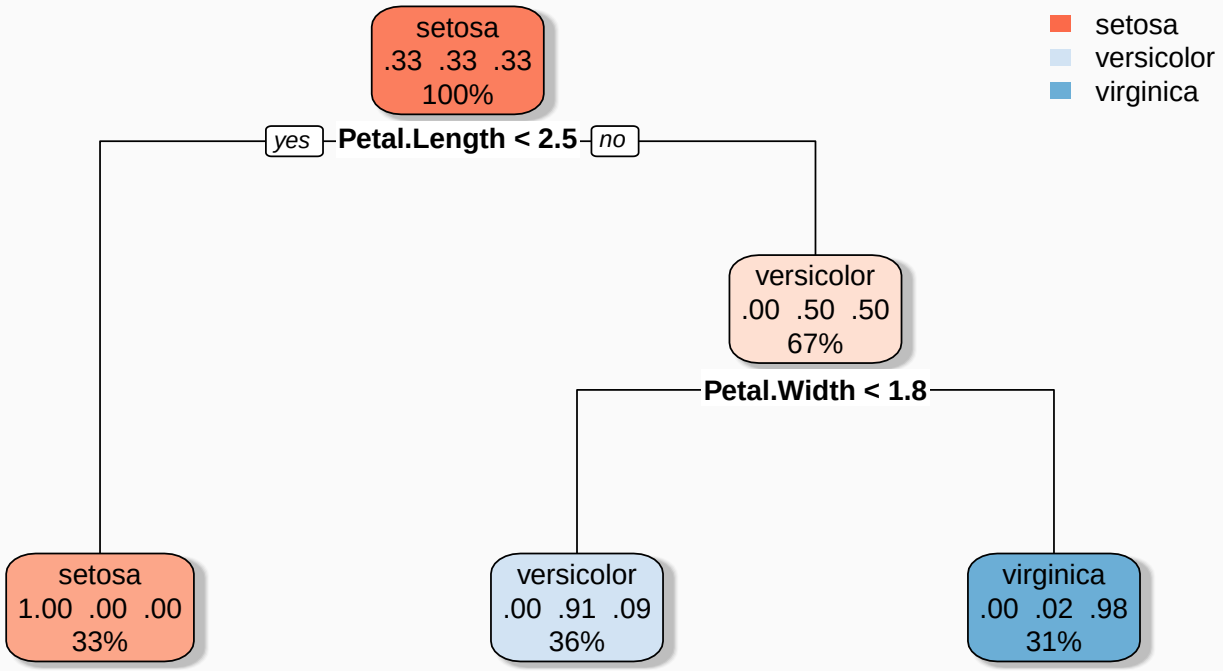
Classificando o dataset *iris*

Diferenciação entre *versicolor* e *virginica* é mais complexa. Pra isso serve o computador!

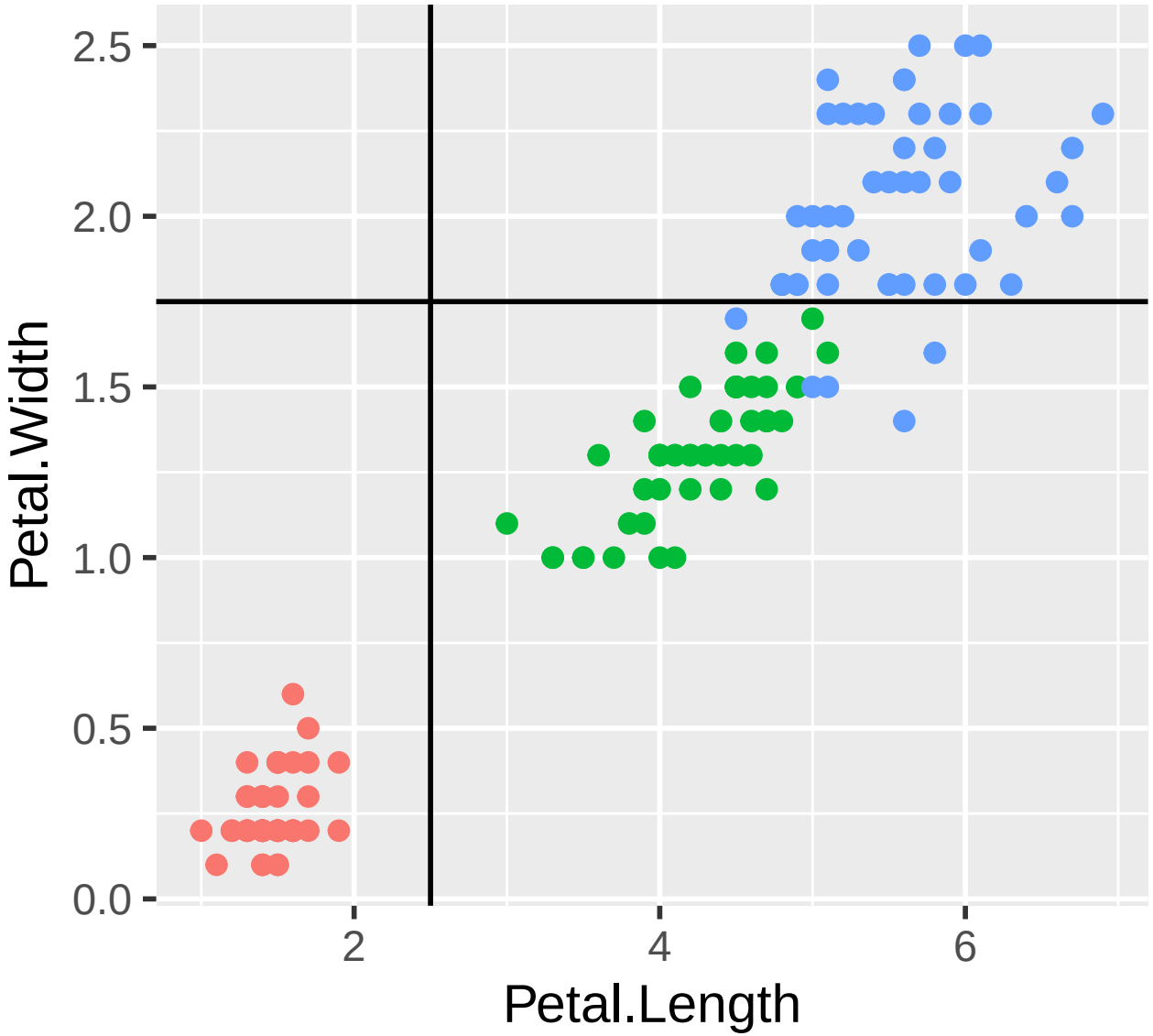
```
library(rpart)
library(rpart.plot)
model1 <- rpart(Species ~ ., data = iris,
                method = "class")
```

Classificando o dataset *iris*

```
rpart.plot(model1, box.palette = "RdBu", shadow.col = "gray",
           fallen.leaves = TRUE)
```



Classificando o dataset *iris*

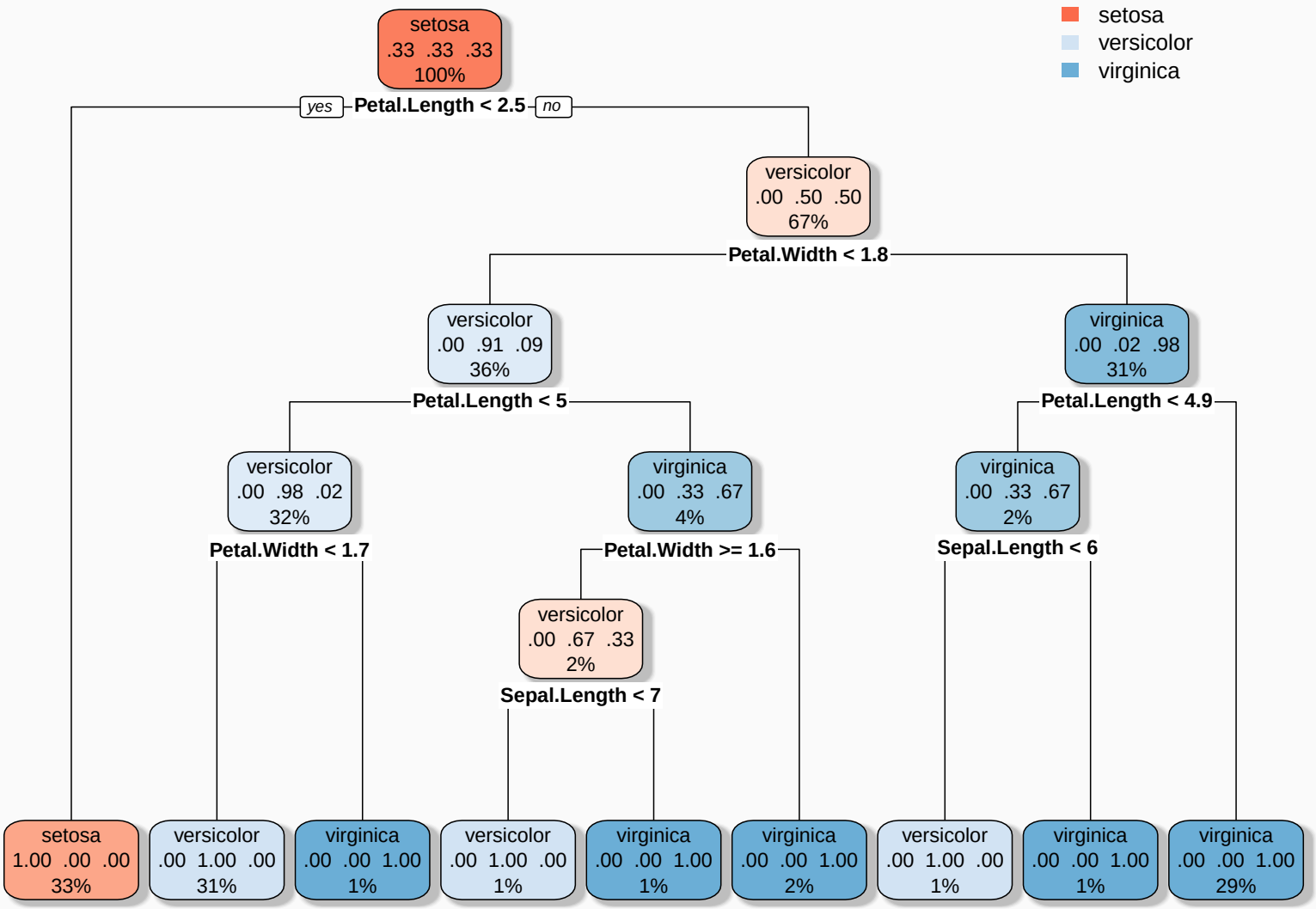


Classificando o dataset *iris*

Poderíamos ir além?

```
model2 <- rpart(Species ~ .,
                data = iris,
                method = "class",
                control = rpart.control(cp = 0,
                                        minsplit = 2))
```

Classificando o dataset *iris*



Classificando o dataset *iris*

Nesse novo modelo, mudamos dois *hiperparâmetros*:

- `cp` : controla o grau de complexidade da árvore
- `minsplit` : numero mínimo de observações após uma separação.

```
[1] "Acurácia M1 (cp = 0.01, minsplit = 20): 96%"
```

```
[1] "Acurácia M2 (cp = 0, minsplit = 2 ): 100%"
```

Qual o melhor modelo?

De árvores para florestas

As árvores de decisào funcionam bem com problemas simples, mas:

- Perdem eficiência em problemas complexos
- Tem uma grande tendência ao overfitting

Random Forests

Random Forests (Florestas Aleatórias): um modo de deixar o uso de árvores de decisão mais robusto e poderoso.

Criado por Leo Breiman em 2001:
https://doi.org/10.1023/A:1010933404324

Random Forests

A ideia é incrivelmente simples:

- Ao invés de uma árvore gigante, crie diversas pequenas árvores
- Cada árvore prevê corretamente só uma pequena parte dos dados (*weak learner*)
- A classificação/regressão final é o *consenso* entre todas as árvores.
- Em ML, essa técnica é chamada de *bagging* (bootstrap + aggregation).

Random Forests

Esse 'truque' de usar vários modelos para gerar uma predição final é chamado de *ensemble* (conjunto).

Ele pode ser também utilizado para combinar resultados de diferentes algoritmos, buscando assim maior robustez.

https://www.youtube.com/watch?v=v6VJ2RO66Ag&t=61s

Random Forests

O algoritmo *Random Forest* é extremamente robusto e poderoso, e fácil de treinar e otimizar. Os principais *hiperparâmetros* a ser otimizados são:

- `n_estimators` : quantas árvores? Mais árvores = melhores modelos = maior tempo de computação.
- `max_features` : o número máximo de features considerado a cada separação, O padrão é `sqrt(total_features)` . Serve para criar árvores mais diferentes.
- `max_depth` :** profundidade máxima de cada árvore (reduz overfitting)
- `min_samples_split` : número mínimo de amostras que um nó deve ter pra ser considerado pra mais uma separação.
- `min_samples_leaf` : número mínimo de amostras nas folhas (nós terminais).

Um exemplo usando o pacote `ranger`

Existem mais de um pacote no `R` que implementam o Random Forest:

- `ranger`
- `randomForest`
- `partykit`
- `aorsf`

Vamos usar o pacote `ranger` para um exemplo.

Quais dados vamos modelar?

Breast Cancer Winsconsin: um dataset com dados reais derivados de imagens de células cancerígenas, utilizados para diagnosticar entre maligno/benigno.

Disponível no R no pacote `mlbench`, um pacote com vários datasets clássicos usados como *benchmark* no mundo do ML.

Breast Cancer Wisconsin (Diagnostic)

Quais dados vamos modelar?

```
data("BreastCancer", package = "mlbench")
head(BreastCancer)
str(BreastCancer)
```

Primeiro Passo - Inspecionando os dados

```
library(GGAlly)
summary(BreastCancer)
ggpairs(BreastCancer, columns = c(2:5,11))
ggpairs(BreastCancer, columns = c(6:10,11))
```

Segundo Passo - Separando os dados em treino e teste

```r
set.seed(42)
nrow(BreastCancer) * 0.7
row_inds <- c(1:nrow(BreastCancer))
train_inds <- sample(row_inds,490, replace = FALSE)

train_df <- BreastCancer[train_inds,-1]
test_df <- BreastCancer[-train_inds,-1]
```

Terceiro Passo - Ajuste do modelo

```
library(ranger)

m1 <- ranger(Class ~ ., data = train_df)

m1$prediction.error
```

Quarto passo - otimização do modelo

```
tune_df <- data.frame(numtree_vals = c(10, 50,100,500,1000),
                      error = NA)
tune_df

for(n in c(1:nrow(tune_df))){
    temp_mod <- ranger(Class ~ ., data = train_df,
                       num.trees = tune_df$numtree_vals[n])
    tune_df$error[n] <- temp_mod$prediction.error
}

tune_df
```

Quarto passo - otimização do modelo

```
tune_df <- data.frame(
    numtree_vals = rep(c(10, 50,100,500,1000),3),
    mtry_vals = rep(c(2,4,6),3, each=5),
    error = NA
    )
tune_df

for(n in c(1:nrow(tune_df))){
    temp_mod <- ranger(Class ~ ., data = train_df,
                       num.trees = tune_df$numtree_vals[n],
                       mtry = tune_df$mtry_vals[n])
    tune_df$error[n] <- temp_mod$prediction.error
}

tune_df

which(tune_df$error == min(tune_df$error))
```

Quinto Passo - Teste do modelo

```
final_mod <- ranger(Class ~ ., data = train_df,
                    num.trees = 1000,
                    mtry = 2)
pred <- predict(final_mod,test_df)

conf_mat <- table(test_df$Class, pred$predictions)

acuracia <- sum(diag(conf_mat)) / sum(conf_mat)

erro <- 1-acuracia
```

Parte III - Conceitos Adicionais

`set.seed()` ?

O comando `set.seed()` fica a 'semente'do gerador de numeros aleatorios.

- É muito dificuil criar um mecanismo computacional para gerar numeros realmente aleatórios
- Computadores usam algortimos que geram números pseudo-aleatórios
- A partir de um valor inicial, aplicam uma série de cálculos instáveis, que resultam em numeros completamente diferentes mesmo se a semente é bem similar.
- Sem uma especificação de semente, o computador usa o horário.

`set.seed()` ?

```
set.seed(42)
runif(5,0,1)
```

```
[1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455
```

```
set.seed(43)
runif(5,0,1)
```

```
[1] 0.48503768 0.91076604 0.05767389 0.70539841 0.31349510
```

`set.seed()` ?

Especificar o `set.seed()` garante a reprodutibilidade dos resultados.

Engenharia de feições

Pré-processamento das feições para melhorar a performance do modelo.

- Normalização/Padronização: equaliza a escala entre as feições contínuas.
- Transformações não-lineares: logaritmos, raízes, potências.
- Combinações de variáveis: multiplicações, razões. Ex. NDVI.
- Desmembramento de variáveis: separa informações complexas, como decompor datas em dia, mês, ano, semestre, trimestre, etc.
- *One-Hot Encoding*: converter variáveis categóricas em variáveis *dummy* numéricas.
- *Label Encoding*: substuir categorias por números inteiros (menos impprtante no R).
- Imputação: preenchimento de valores vazios usando alguma regra.

Otimizando o modelo - validação cruzada

- Sempre queremos testar o modelo com dados 'novos'.
- A separação entre treino e teste já sacrifica parte dos dados.
- Se precisarmos separar a amostra de treino de novo entre treino e validação, perdemos ainda mais.

Otimizando o modelo - validação cruzada

k- (ou v-) fold cross validation: separação dos dados de treino em 'blocos' (*folds*).

Ex: 5-fold cross-validation

1. Divida os dados de treino aleatoriamente em 5 blocos iguais.
2. Junte quatro dos blocos em uma tablela e ajuste o modelo.
3. Use o quinto bloco para validar o modelo, calculando uma métrica de erro.
4. Repita mais 4 vezes, cada vez deixando um bloco de fora.
5. Faça a média dos 5 erros calculados.

Vantagens:

- Todos os dados de treino são usados para treino e para validação.
- A cada combinação de hiperparametros testada, você estima

Métricas de Erro/Acurácia

- Quantificar o erro de predição é um dos aspectos essencias do processo de ML.
- Como se o modelo fosse uma prova, e a métrica fosse a nota da prova.
- Quando melhor a nota, maior o aprendizado (em teoria)

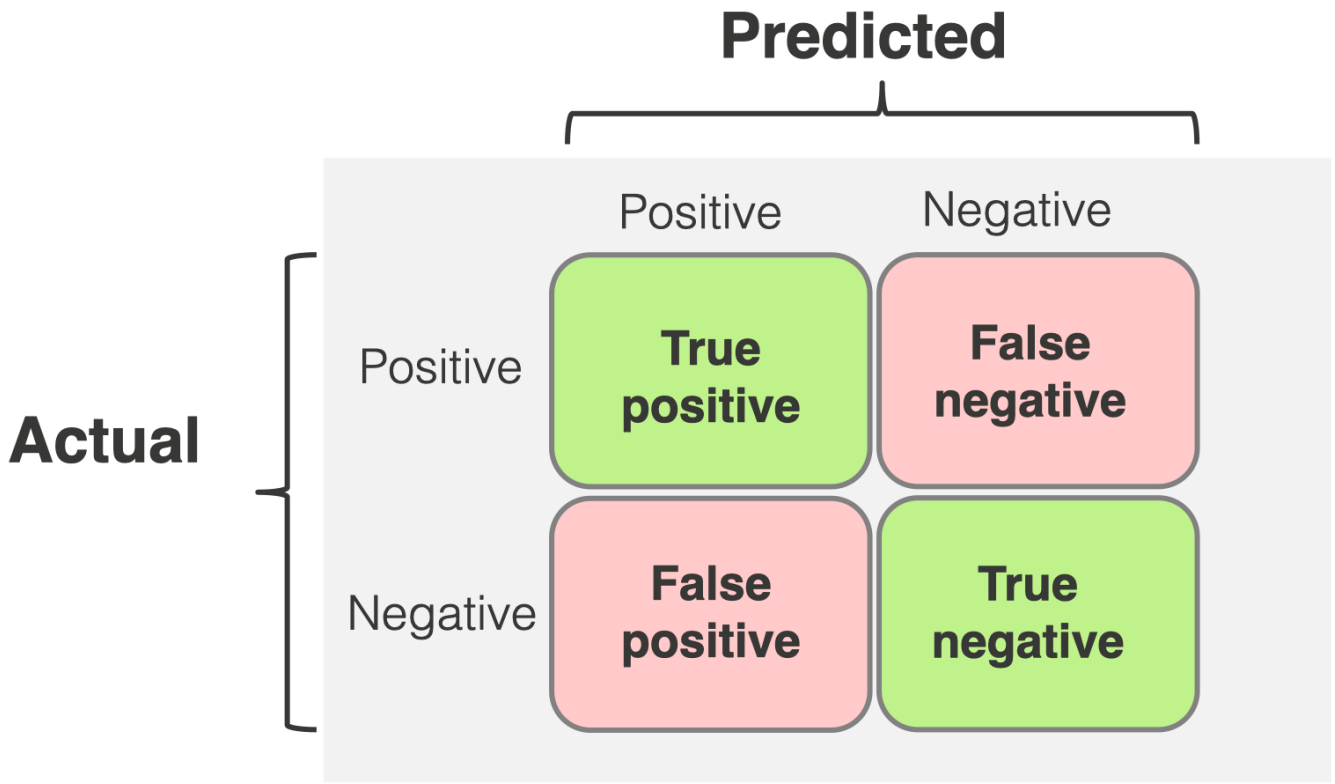
Métricas de Erro/Acurácia

Existem dezenas de métricas de erro, vamos focar apenas em algumas. Também são chamadas de *loss functions* ou *scoring functions*.

- Acurácia Simples
- Índice Kappa
- AUC/ROC
- Precision/Recall/F1 score
- RMSE e MAE

A matriz de confusão

Todas as métricas de classificação são de alguma maneira relacionadas à matriz de confusão:



A matriz de confusão

Para um problema multi-classe, temos que considerar cada classe:

	Classe A	Classe B	Classe C	Classe D
Classe A	*TP*	*FN*	*FN*	*FN*
Classe B	TN	TN	TN	TN
Classe C	TN	TN	TN	TN
Classe D	TN	TN	TN	TN

A matriz de confusão

Para um problema multi-classe, temos que considerar cada classe:

	Classe A	**Classe B**	Classe C	Classe D
Classe A	TN	TN	TN	TN
Classe B	*FN*	*TP*	*FN*	*FN*
Classe C	TN	TN	TN	TN
Classe D	TN	TN	TN	TN

A matriz de confusão

Dependendo da métrica, podemos generalizar para:

	Classe A	Classe B	Classe C	Classe D
Classe A	Correto	Errado	Errado	Errado
Classe B	Errado	Correto	Errado	Errado
Classe C	Errado	Errado	Correto	Errado
Classe D	Errado	Errado	Errado	Correto

Acurácia Simples ou Global

Simplesmente a proporção de acertos (ou erros):

$$Acuracia = \frac{TP+TN}{TP+TN+FP+FN}$$

Acurácia Simples ou Global

	Classe A	Classe B
Classe A	26	3
Classe B	6	30

- Ac. Global = (26 + 30)/(26+3+6+30) = 0.862
- Ac. Classe A = 26/(26+3) = 0.897
- Ac. Classe B = 30/(30+6) = 0.833

Índice Kappa

Muito usado em sensoriamento remoto, é uma modificação da acurácia que 'desconta' a probabilidade de você acertar por acaso.

$$\kappa = \frac{2\times(TP\times TN - FN\times FP)}{(TP+FP)\times(FP+TN)+(TP+FN)\times(FN+TN)}$$

Índice Kappa

	Classe A	Classe B
Classe A	26	3
Classe B	6	30

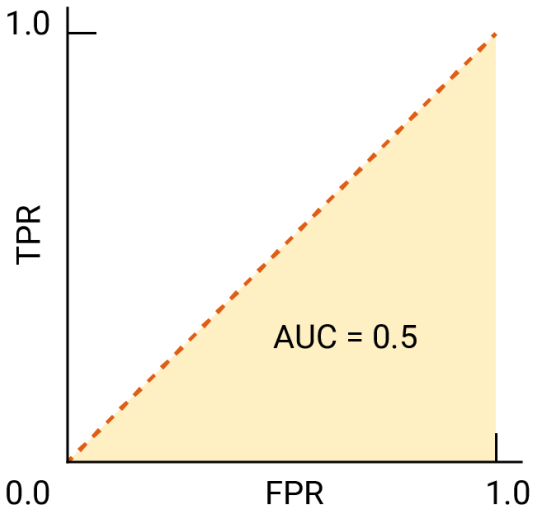
Acurácia Global $= 0.862$ Kappa $= 0.723$.

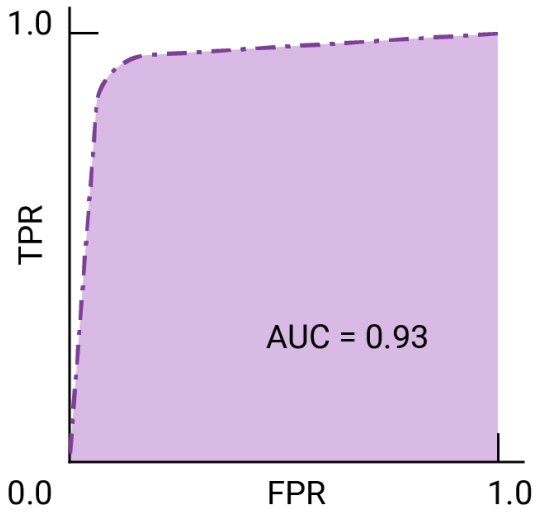
ROC/AUC

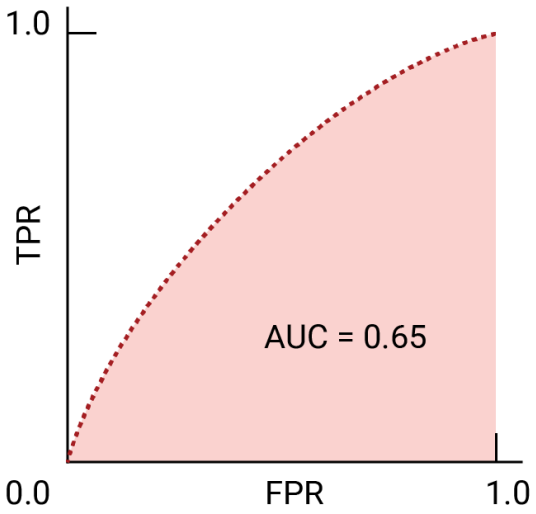
ROC significa *Receiver Operator Curve*: representação visual da performance do modelo.

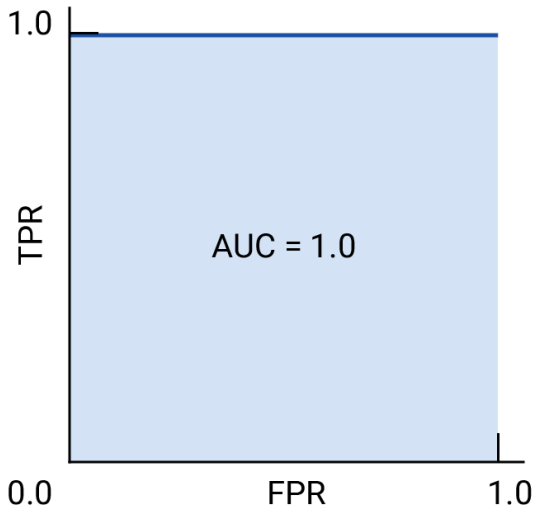
AUC significa *Area Under the Curve*: área abaixo da curva ROC.

ROC/AUC


1.0
TPR
AUC = 0.5
0.0
FPR
1.0

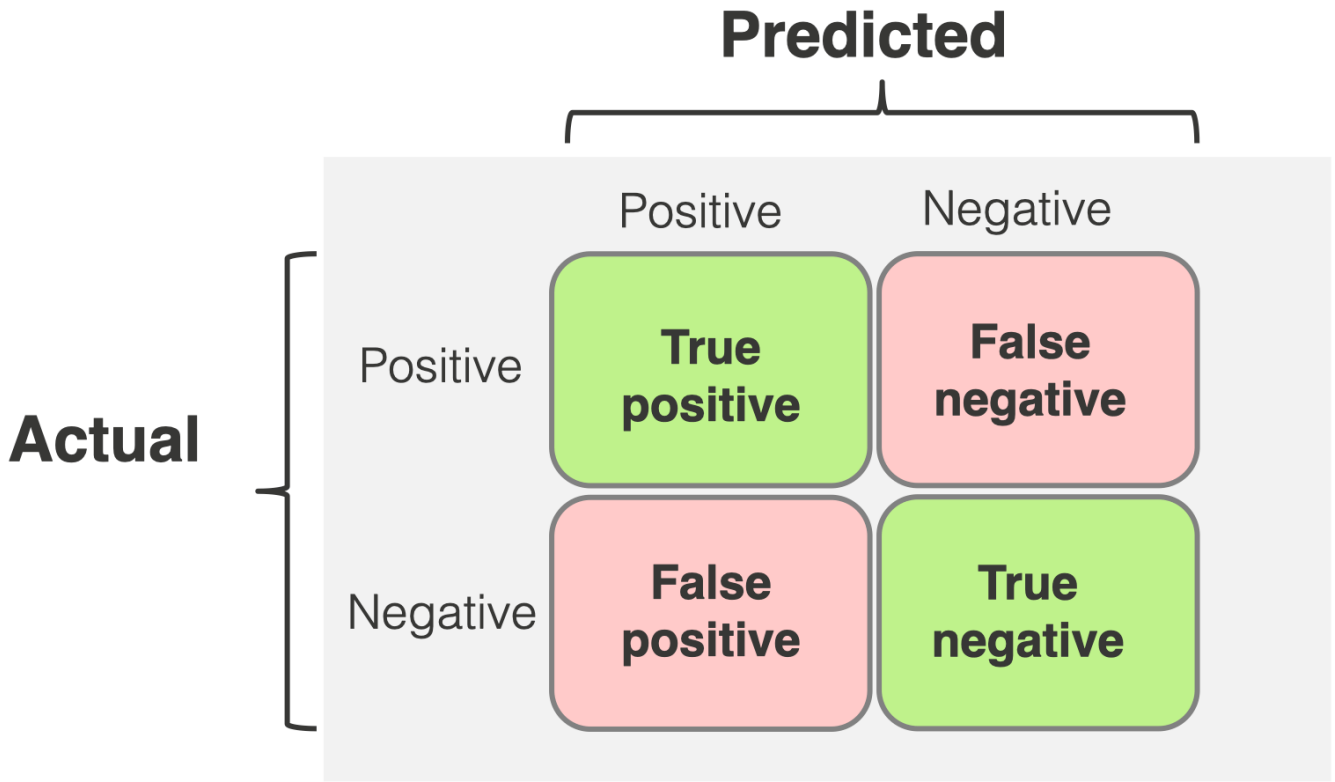

1.0
TPR
AUC = 0.93
0.0
FPR
1.0


1.0
TPR
AUC = 0.65
0.0
FPR
1.0


1.0
TPR
AUC = 1.0
0.0
FPR
1.0

Precision, Recall, Sensitivity, Specificty, F1 Score

Métricas mais apropriadas para o *TESTE* do modelo, pois podem ser interpretadas mais sutilmente. Relembrando:



Precision, Recall, Sensitivity, Specificty, F1 Score

Precision (TP/TP+FP): porcentagem de positivos corretos dentre todos os positivos *detectados*. Em SR, a *'acurácia do produtor'*. Maior precisão = menor chance de falsos positivos.

	Classe A	Classe B
Classe A	26	3
Classe B	6	30

- Precisão Classe A = 26/(26+6) = 0.897
- Precisão Classe B = 30/(30+3) = 0.833

Precision, Recall, F1 Score

Recall (TP/TP+FN): porcentagem de positivos verdadeiros que foram efetivamente identificados. O mesmo que acurácia por classe ou *'acurácia do usuário*. Maior recall = maior chance de detecção de positivos.

	Classe A	Classe B
Classe A	26	3
Classe B	6	30

- Precisão Classe A = 26/(26+3) = 0.897
- Precisão Classe B = 30/(30+6) = 0.833

Exemplos de aplicação

Caso 1: Deteção de Spam - é melhor deixar um e-mail de Spam cair no seu inbox do que jogar um e-mail real na caixa de Spam.

- A Precisão foca em minimizar Falsos Positivos (FP). Maior precisão significa menos emails de SPAM são marcados como reais, à custa de marcar algus emails reais como spam.

Exemplos de aplicação

Caso 2: Diagnostico de uma doença - é melhor diagnosticar um paciente sadio como doente do que deixar de diagnosticar um paciente doente.

- O Recall foca em minimizar Falsos Negativos. Maior recall significa que mais pacientes doentes são diagnosticados, à custa de diagnosticar alguns sadios.

Precision, Recall, F1 Score

Exemplo interativo:

https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

F1 Score

F1 Score: uma métrica única que combina Precision e Recall:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Essa formula prioriza valores balanceados das duas métricas.

Erros contínuos

Para problemas de *regressão*, os erros são mais simples: quão perto/longe você chegou do valor real?

- Root Mean Squared Error (RMSE): $\sqrt{\frac{\sum_{i=1}^{N}(pred_i - obs_i)^2}{N}}$
- Mean Absolute Error (MAE): $\frac{\sum_{i=1}^{N}|(pred_i - obs_i)|}{N}$