

**MBAUSP**  
**ESALQ**

# Árvores, Redes e Ensemble Models IV

João Fernando Serrajordia Rocha de Mello

# MBAUSP ESALA

A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

**Proibida a reprodução**, total ou parcial, sem autorização.

Lei nº 9610/98

# Você vai precisar de...



## Preparativos

- Abrir o Spyder
- Rodar o script 00
- Algo para fazer suas anotações

# Agenda

Revisão

*Histórico*

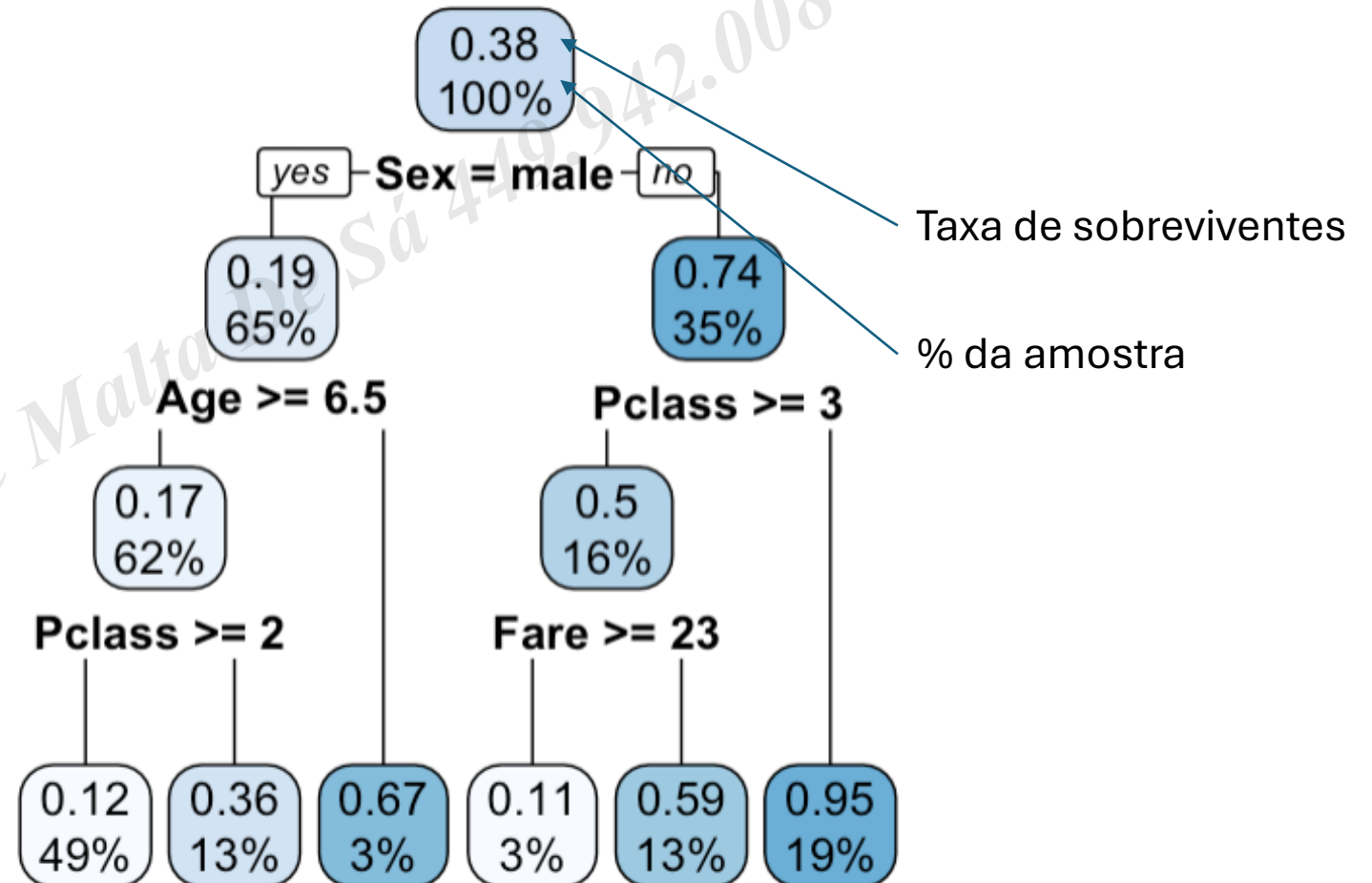
*Ideias básicas*

*Usos*

# O que é uma árvore de decisão?

A árvore de decisão é:

Uma sequência de segmentações binárias  
Que visa homogeneidade da variável resposta



# Índice de Gini

$$I_g(p) = 1 - \sum_{i=1}^J p_i^2$$

- Impureza máxima com distribuição uniforme
- Impureza mínima na concentração total

Lucas Rezende Malta De Sá 449.942.008-31

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Criterion – é o critério. Pode ser GINI ou Entropia na árvore de classificação.



# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",
                        max_depth=None, min_samples_split=2,
                        min_samples_leaf=1,
                        min_weight_fraction_leaf=0.0,
                        max_features=None, random_state=None,
                        max_leaf_nodes=None,
                        min_impurity_decrease=0.0,
                        class_weight=None, ccp_alpha=0.0,
                        monotonic_cst=None)
```

Splitter – como é escolhida a quebra. Pode ser ‘best’ ou ‘Random’ – a melhor quebra ou aleatória.

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Max\_depth – a profundidade máxima da árvore (o número máximo de quebras para classificar um indivíduo)

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Min\_samples\_split – o número mínimo de observações para se procurar um split

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Min\_samples\_leaf – o número mínimo de observações permitido por folha

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Max\_features – sorteia as variáveis a serem consideradas a cada quebra.

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Max\_leaf\_nodes – O número máximo permitido de folhas na árvore final.

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Min\_impurity\_decrease – A cada quebra testa se o ganho é maior que este parâmetro.

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Class\_weight – controla a ponderação da target:

**None** – peso 1 para todos,

**'balanced'** – emula uma amostra estratificada.



# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Min\_weight\_fraction\_leaf– Semelhante a min\_samples\_leaf, mas ponderado pelos pesos das classes.

# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",  
                        max_depth=None, min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features=None, random_state=None,  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        class_weight=None, ccp_alpha=0.0,  
                        monotonic_cst=None)
```

Ccp\_alpha – Semelhante a min\_impurity\_decrease, mas elimina quebras após a árvore estar pronta.

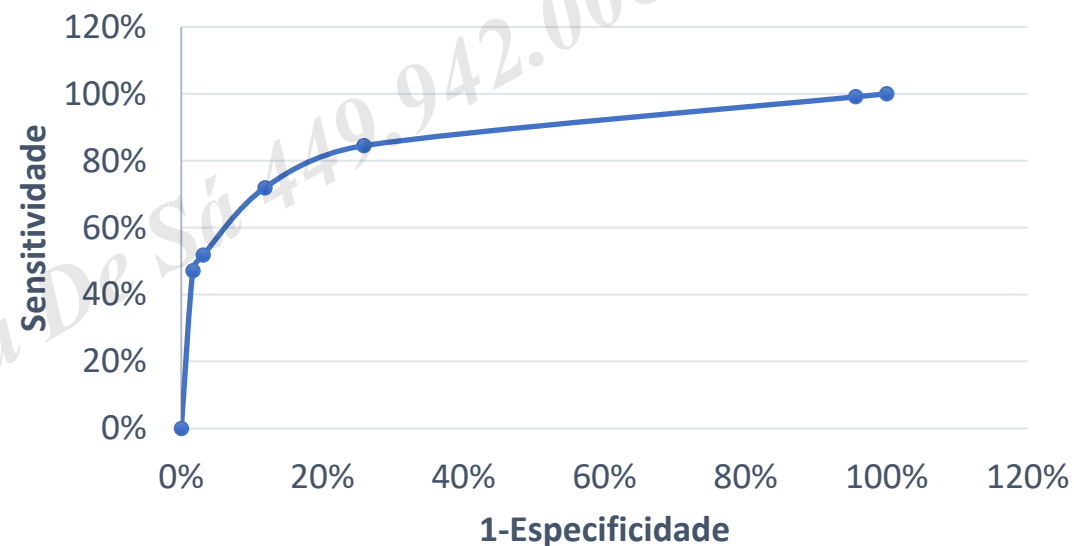
# Hiperparâmetros

```
DecisionTreeClassifier(*, criterion="gini", splitter="best",
                        max_depth=None, min_samples_split=2,
                        min_samples_leaf=1,
                        min_weight_fraction_leaf=0.0,
                        max_features=None, random_state=None,
                        max_leaf_nodes=None,
                        min_impurity_decrease=0.0,
                        class_weight=None, ccp_alpha=0.0,
                        monotonic_cst=None)
```

Monotonic\_cst – restrições monotônicas: define se a ordem das variáveis será importante. Se pode passar uma lista indicando quais variáveis são ordinais, quais não.

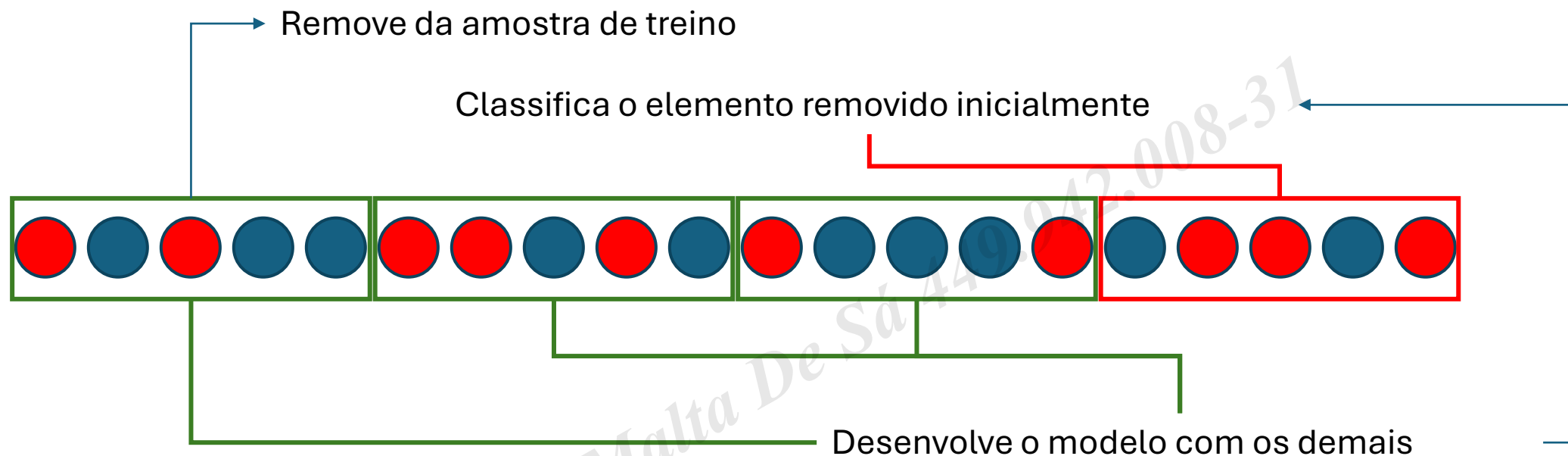
# Curva ROC

Corte	1-Especificidade	Sensibilidade
0% - 11,1%	100%	100%
11,1% - 11,5%	96%	99%
11,5% - 35,8%	26%	85%
35,8% - 58,9%	12%	72%
58,9% - 66,7%	3%	52%
66,7% - 94,7%	2%	47%
94,7% - 100%	0%	0%



A curva ROC é um gráfico de dispersão de 1-Especificidade no eixo x por Sensibilidade no eixo y, obtidos para cada possível ponto de corte do classificador.

# K-fold



- Dividimos a base em k subamostras
- Para cada subamostra:
  - Removemos a subamostra como validação
  - Treinamos o modelo com as observações restantes
  - Utilizamos este modelo para classificar a subamostra removida
  - Avaliamos a métrica de desempenho do modelo
- Calculamos a média das métricas de desempenho do modelo

# K-fold

Tipicamente, fazemos o mesmo para variações do modelo para otimizar hiperparâmetros.



	Acurácia 1	Acurácia 2	Acurácia 3	Acurácia 4	Acurácia Média
Modelo 1	62%	58%	61%	59%	<b>60%</b>
Modelo 2	50%	51%	49%	47%	<b>49%</b>
Modelo 3	72%	68%	71%	75%	<b>72%</b>

# Exemplo numérico



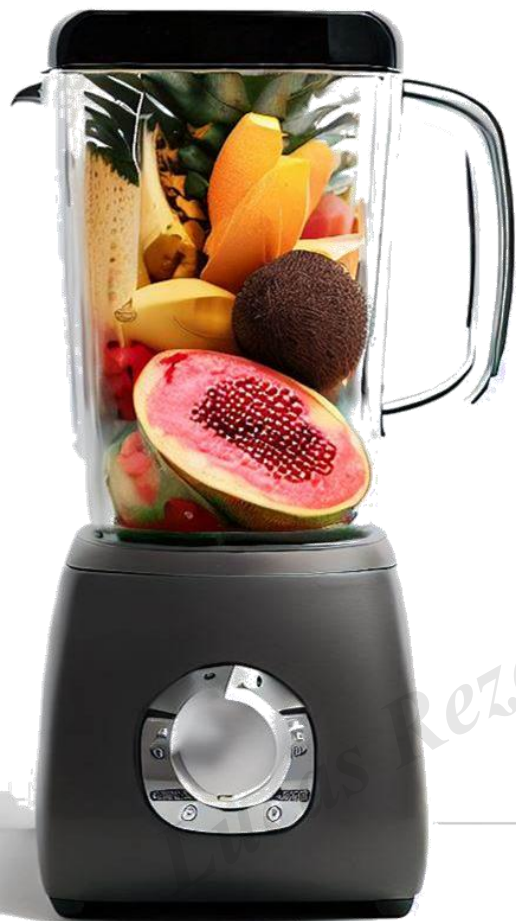


# Exemplo revisado no titanic

Vamos abrir o Spyder







# Ensemble

---

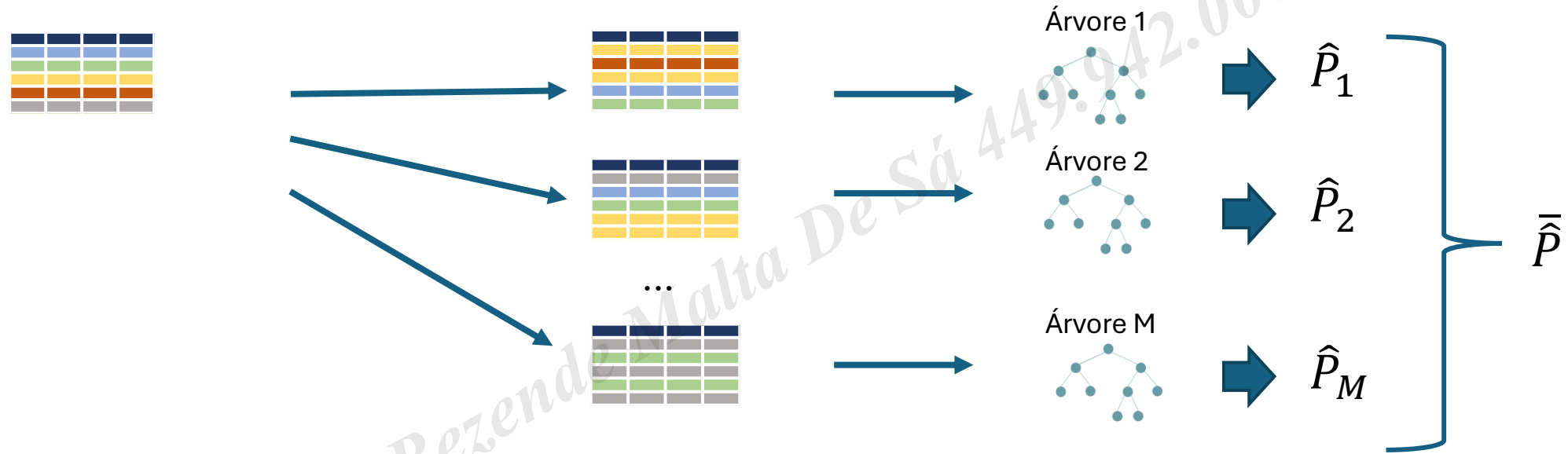
Um ensemble é qualquer mistura de modelos já existentes. Os principais tipos são:

*Bagging*

*Boosting*

*Stacking*

# Bootstrap – aggregation (bagging)



O *bagging* com árvores é o famoso *Random Forest*

# *Bootstrap – aggregation (bagging)*

```
RandomForestClassifier(n_estimators=100, *,  
                        criterion="gini", max_depth=None,  
                        min_samples_split=2,  
                        min_samples_leaf=1,  
                        min_weight_fraction_leaf=0.0,  
                        max_features="sqrt",  
                        max_leaf_nodes=None,  
                        min_impurity_decrease=0.0,  
                        bootstrap=True, oob_score=False,  
                        n_jobs=None, random_state=None,  
                        verbose=0, warm_start=False,  
                        class_weight=None, ccp_alpha=0.0,  
                        max_samples=None, monotonic_cst=None)
```

Bootstrap = True – amostragem com reposição  
Bootstrap = False – amostragem sem reposição

# Bootstrap – aggregation (bagging)

```
RandomForestClassifier(n_estimators=100, *,
                       criterion="gini", max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.0,
                       max_features="sqrt",
```

id	x1	x2	x3	x4	x5
1	0,7	0,1	0,8	0,1	0,1
2	0,8	0,8	0,8	0,9	0,8
3	0	0,8	0	0,7	0,3
4	0,6	0,1	1	0	0,6
5	0,2	0,3	0,6	0,5	0,6
6	0,5	0,2	0,7	0,4	0,7



id	x1	x2	x3	x4	x5
1	0,7	0,1	0,8	0,1	0,1
3	0	0,8	0	0,7	0,3
6	0,5	0,2	0,7	0,4	0,7

Max\_features – número de variáveis sorteadas por árvore treinada.

# *Bootstrap – aggregation (bagging)*

```
RandomForestClassifier(n_estimators=100, *,
                       criterion="gini", max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.0,
                       max_features="sqrt",
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.0,
                       bootstrap=True, oob_score=False,
                       n_jobs=None, random_state=None,
                       verbose=0, warm_start=False,
                       class_weight=None, ccp_alpha=0.0,
                       max_samples=None, monotonic_cst=None)
```

OOB – out of bag. Usa as observações não selecionadas da amostragem como validação cruzada.

# *Bootstrap – aggregation (bagging)*

```
RandomForestClassifier(n_estimators=100, *,
                       criterion="gini", max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.0,
                       max_features="sqrt",
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.0,
                       bootstrap=True, oob_score=False,
                       n_jobs=None, random_state=None,
                       verbose=0, warm_start=False,
                       class_weight=None, ccp_alpha=0.0,
                       max_samples=None, monotonic_cst=None)
```

Warm\_start – se True, não inicia o treinamento do zero, mas sim adiciona árvores ao modelo já treinado.



# Exemplo revisado no titanic

Vamos abrir o Spyder



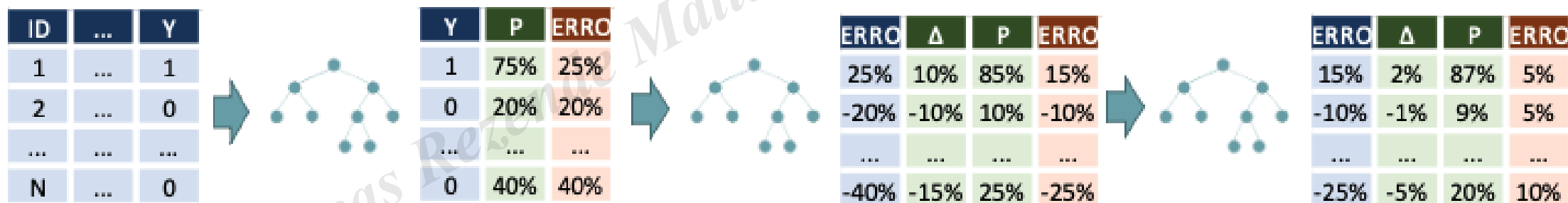
# *Boosting*

Correção sequencial de erros



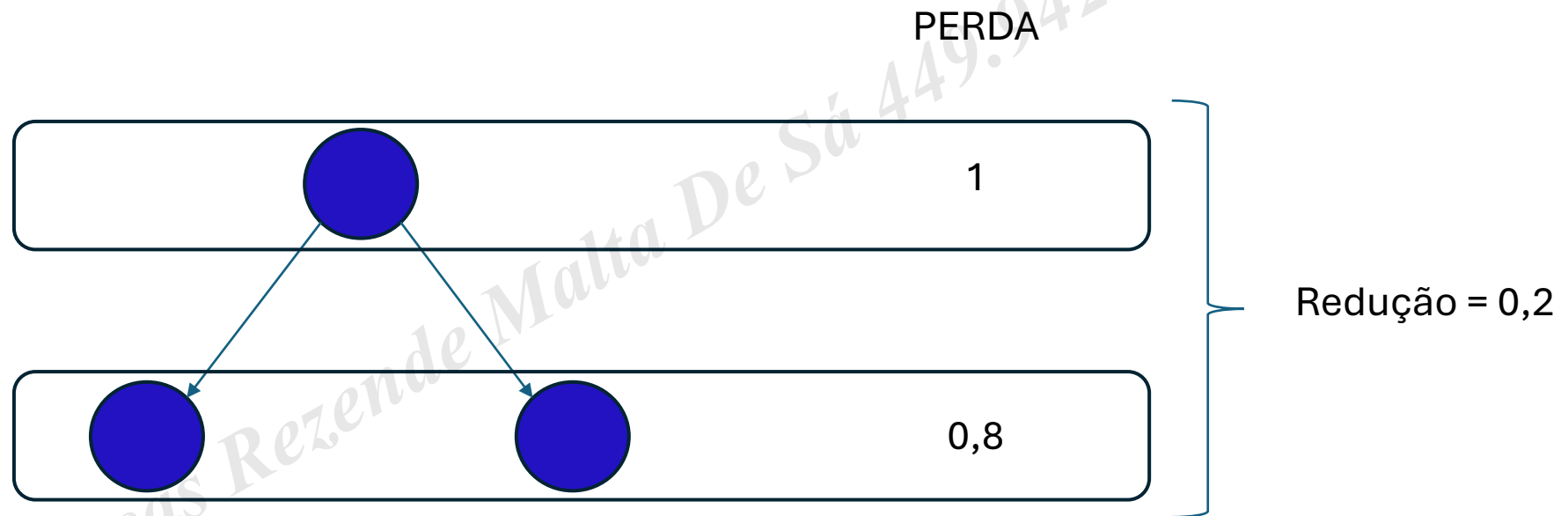
# Gradient Boosting

- O *Gradiente Boosting* um boosting baseado em árvores com alguns hiperparâmetros que controlam o algoritmo



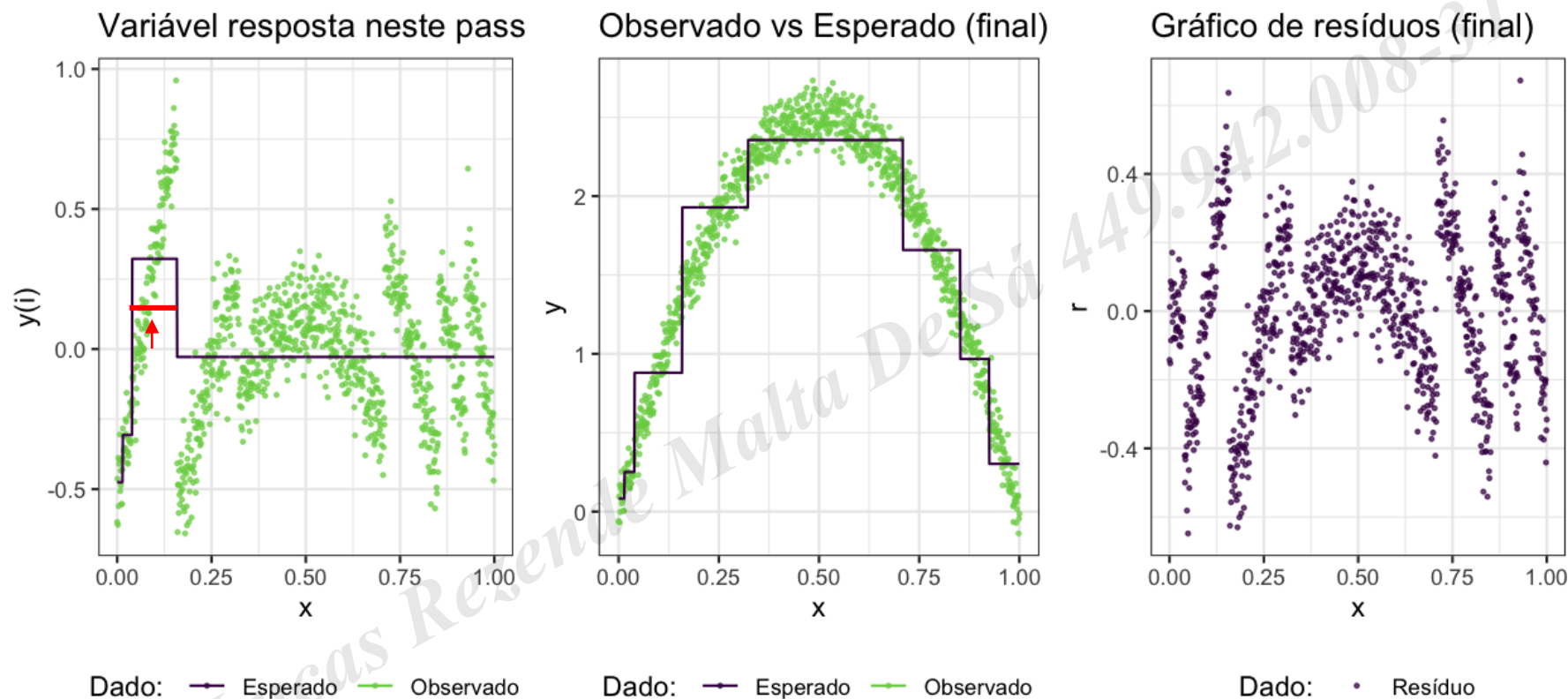
# XGBoost – hiperparâmetros

- Gamma – redução mínima de perda



Se usa em geral valores de gamma entre 0 e 10.  
O valor ótimo pode variar muito de um problema para outro.

# XGBoost - eta



O Learning Rate diminui o impacto de cada iteração  
costuma demandar mais iterações,  
mas ajuda a alcançar melhores resultados

# XGBoost – hiperparâmetros

- Colsample\_bytree – o número de variáveis sorteadas a cada passo
  - Controla o número de colunas sorteadas a cada passo.
- Subsample (variação) – o percentual da amostra no bootstrap
  - Controla o número de linhas sorteadas a cada passo.

id	x1	x2	x3	x4	x5
1	0,9	0,9	0,3	0,1	0
2	0,3	0,7	0,3	0,3	0,8
3	0,5	0,3	0,1	0	0,3
4	0,6	0,4	0,8	0,6	0,9
5	0,5	0,1	0,1	0,7	0,1
6	0,8	0,4	0,1	0,7	0,9



id	x1	x3	x5
1	0,6	0,8	0,5
2	0,7	0,4	0,1
3	0,4	0	0,6
4	0,6	0,9	0,1
5	1	0,4	0,3
6	0,9	0,4	0,9

id	x1	x2	x3	x4	x5
1	0,7	0,1	0,8	0,1	0,1
2	0,8	0,8	0,8	0,9	0,8
3	0	0,8	0	0,7	0,3
4	0,6	0,1	1	0	0,6
5	0,2	0,3	0,6	0,5	0,6
6	0,5	0,2	0,7	0,4	0,7



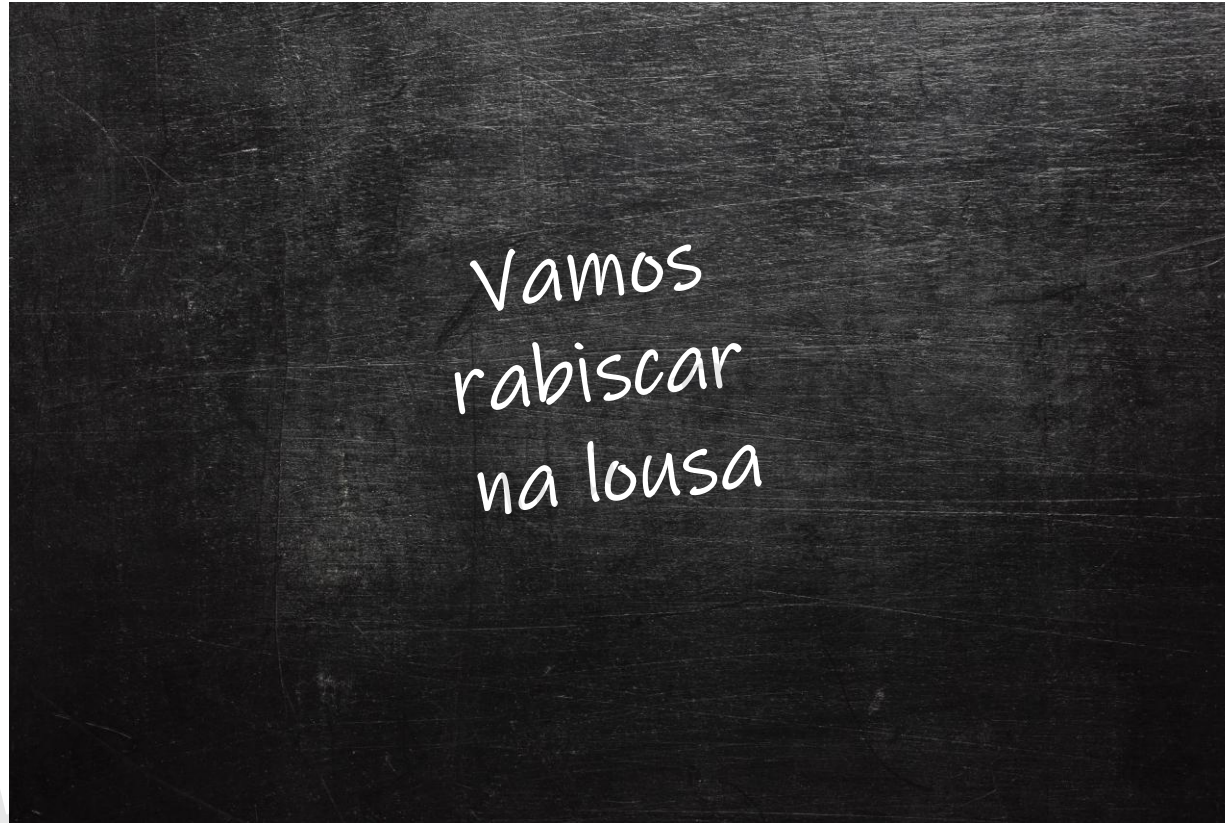
id	x1	x2	x3	x4	x5
1	0,7	0,1	0,8	0,1	0,1
3	0	0,8	0	0,7	0,3
6	0,5	0,2	0,7	0,4	0,7

# XGBoost - hiperparâmetros

- Lambda - Regularização

$$Loss = \frac{\sum_j R^2}{Num.R + \lambda}$$

# XGBoost - hiperparâmetros



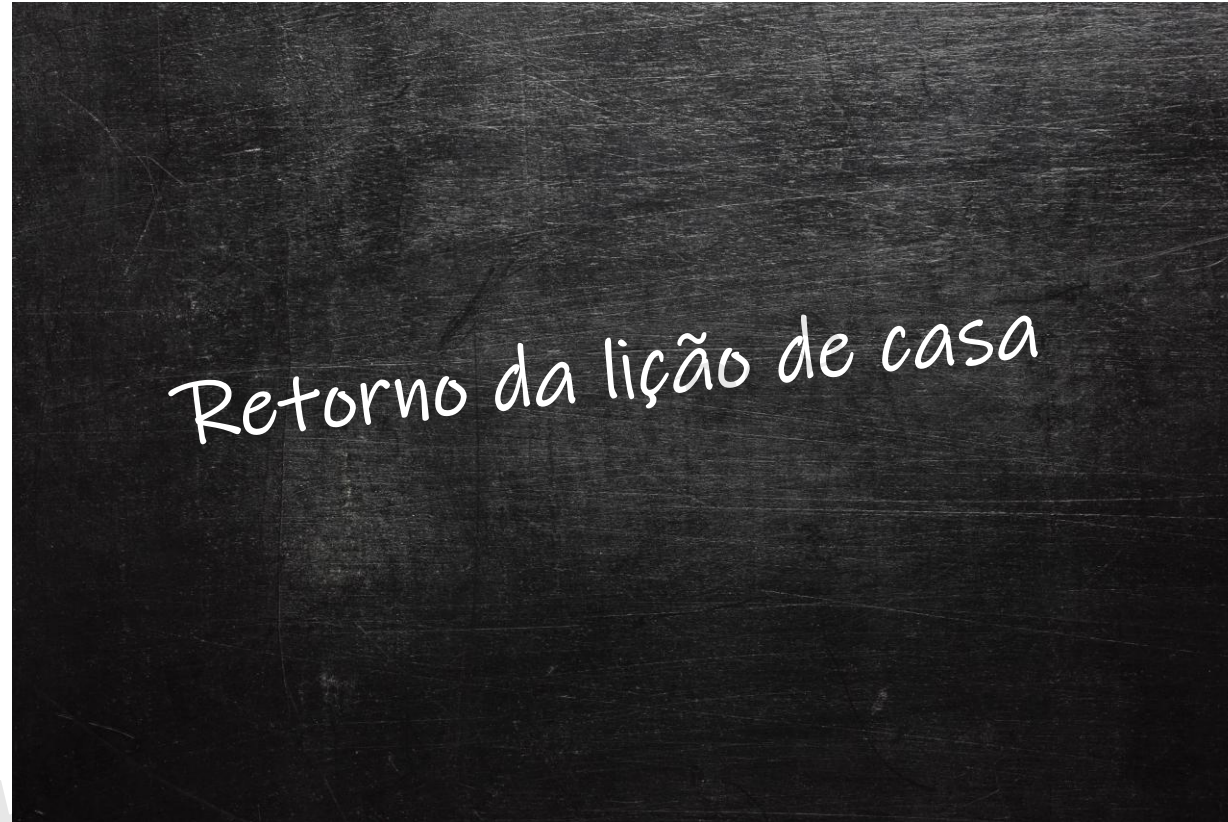


# Exemplo revisado no titanic

Vamos abrir o Spyder

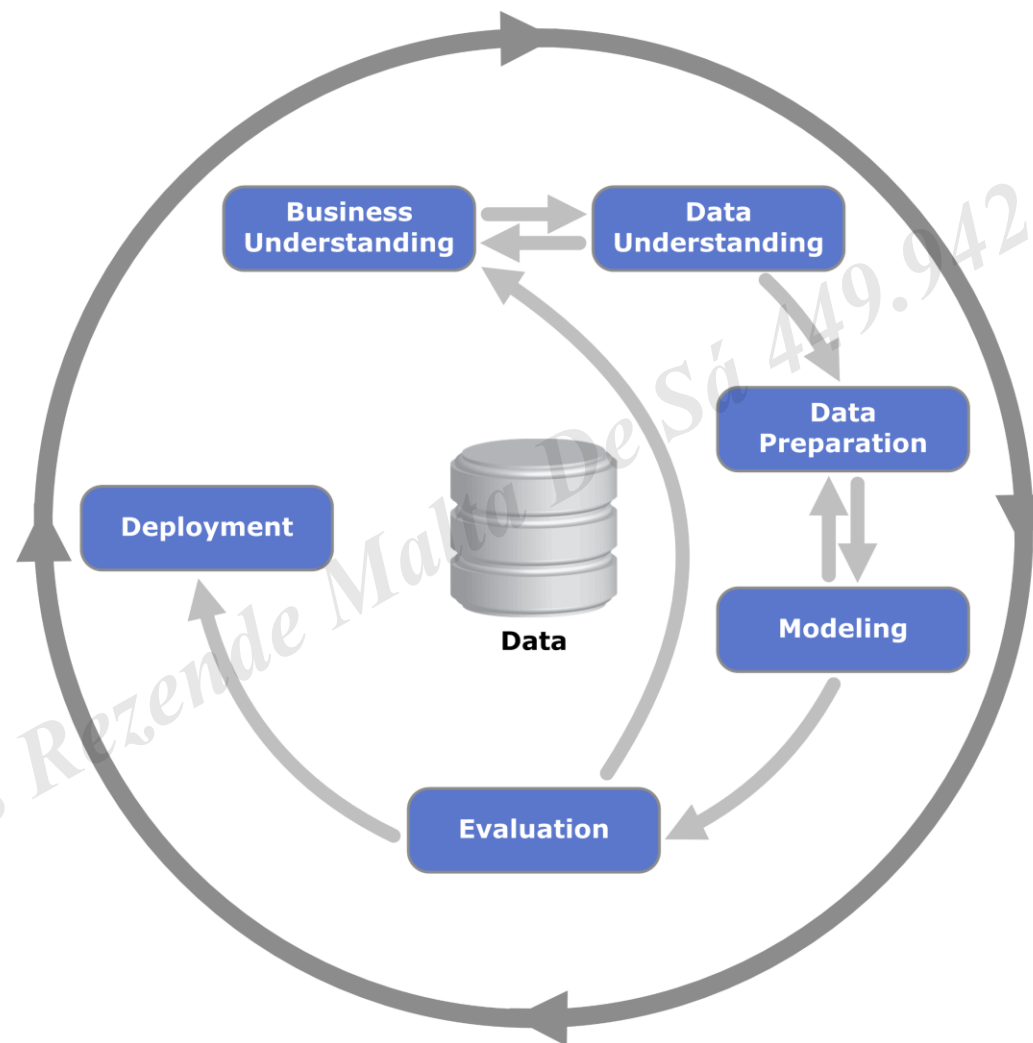


# XGBoost – hiperparâmetros





# CRISP-DM



Fonte: [https://commons.wikimedia.org/wiki/File:CRISP-DM\\_Process\\_Diagram.png](https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png)

\*A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98

# Classificação

Lucas Rezende Malta De Sá 449.942.008-31

# Classificação dos algoritmos



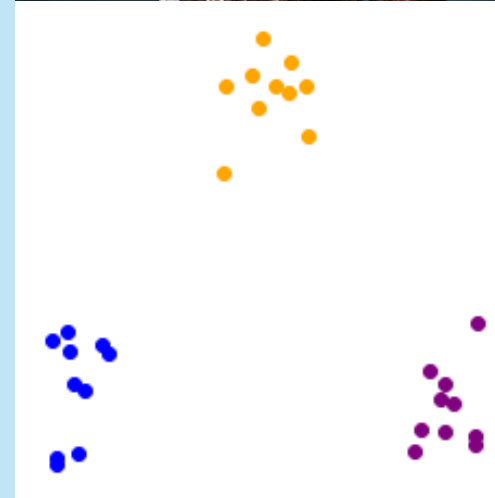
## Supervisionados

- Regressão
- GLM
- GLMM
- Support vector machines
- Naive Bayes
- K-nearest neighbors
- Redes Neurais
- Decision Trees



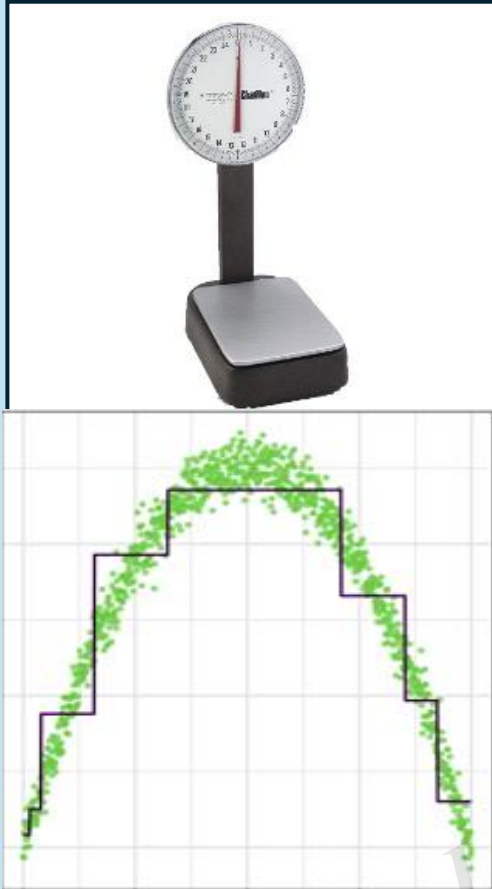
## Não supervisionados

- K-Means
- Métodos hierárquicos
- Mistura Gaussiana
- DBScan
- Mini-Batch-K-Means



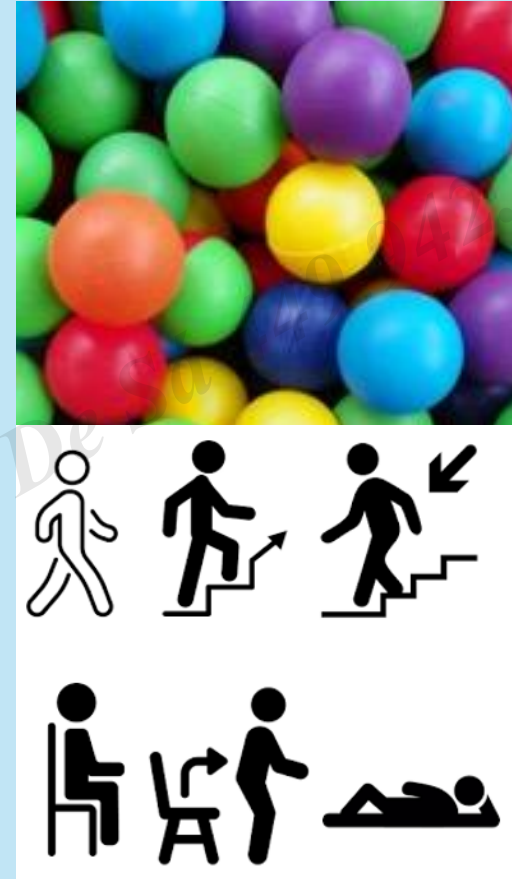
**Estamos aqui!**

# Classificação dos algoritmos



## Resposta contínua

- Regressão
- GLM
- GLMM
- Support vector machines
- K-nearest neighbors
- Redes Neurais
- Regression Trees



## Resposta discreta

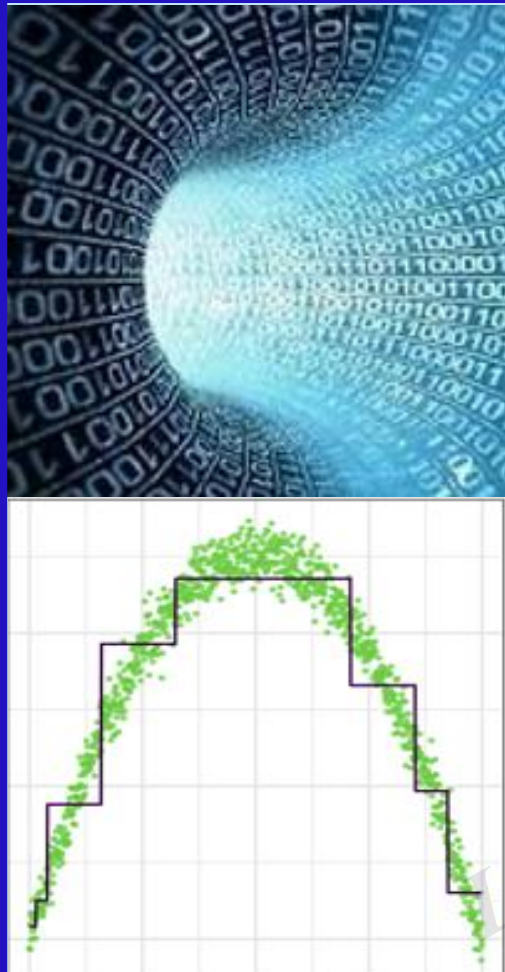
- Regressão logística
- Classification trees
- Redes Neurais
- GLM
- GLMM

**Estamos aqui!**

# Classificação dos algoritmos

## Métodos Machinelârnicos

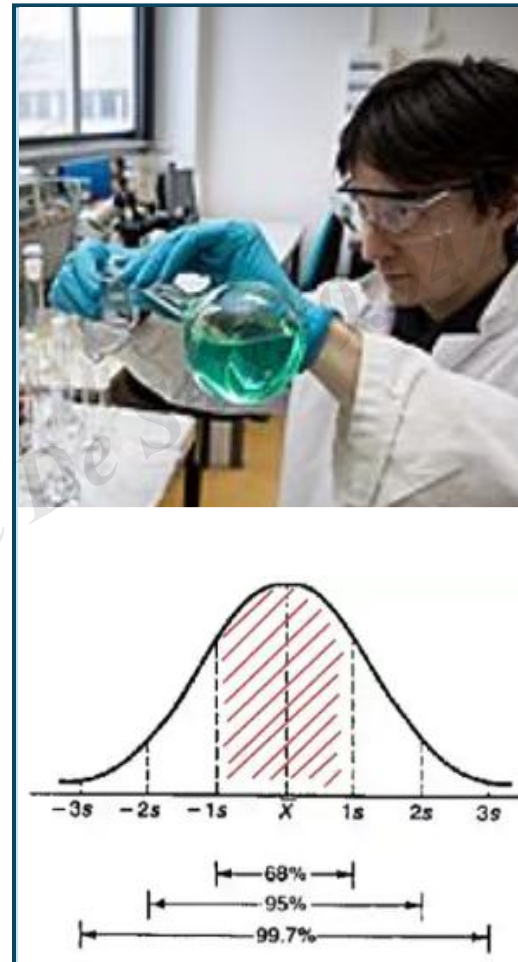
- Árvores de decisão
- Bagging
- Boosting
- K-NN
- Redes Neurais
- Support Vector Machines



**Estamos aqui!**

## Métodos Machinelârnico- estatísticos

- Regressão
- GLM
- GLMM
- ANOVA

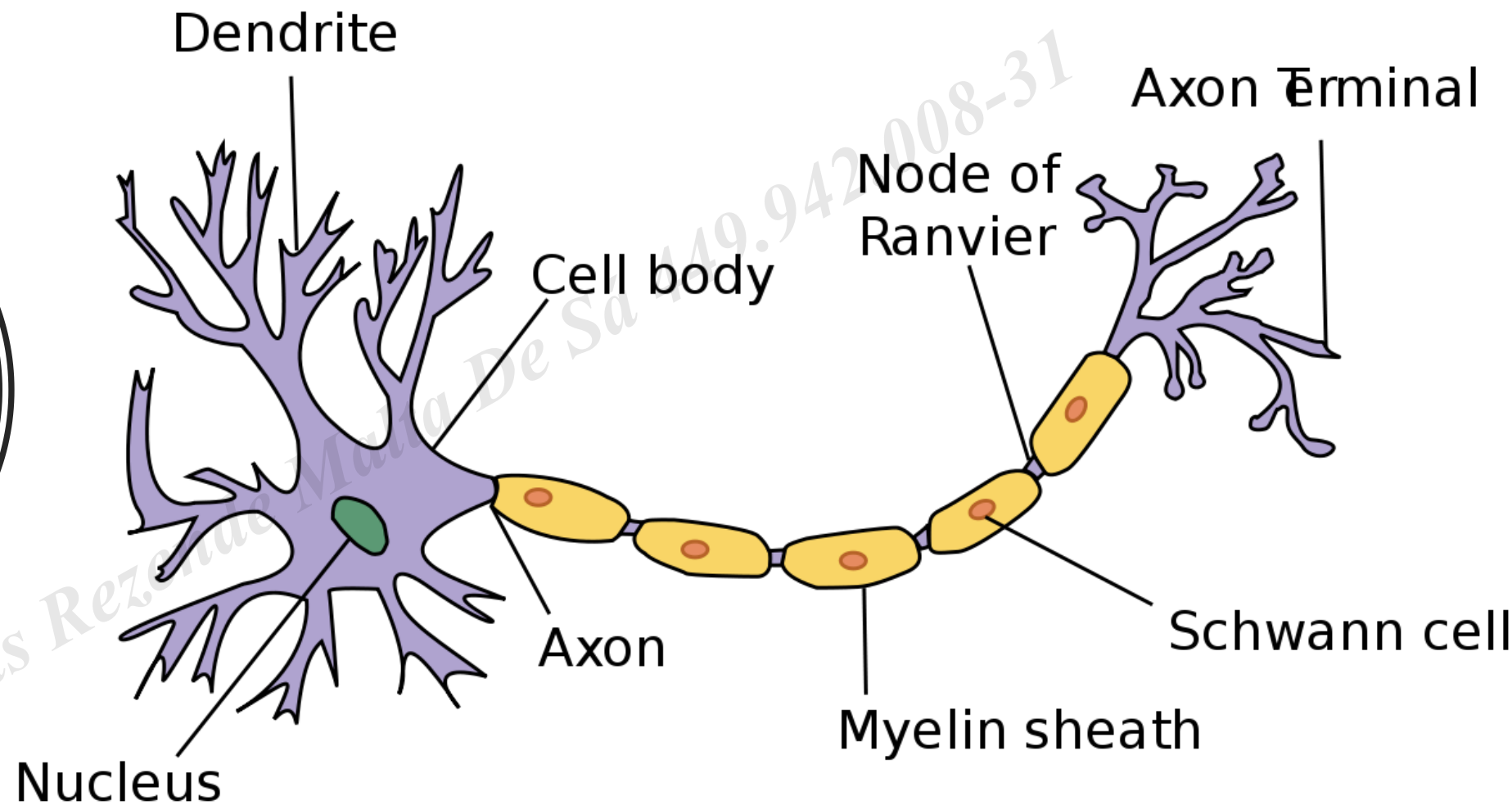






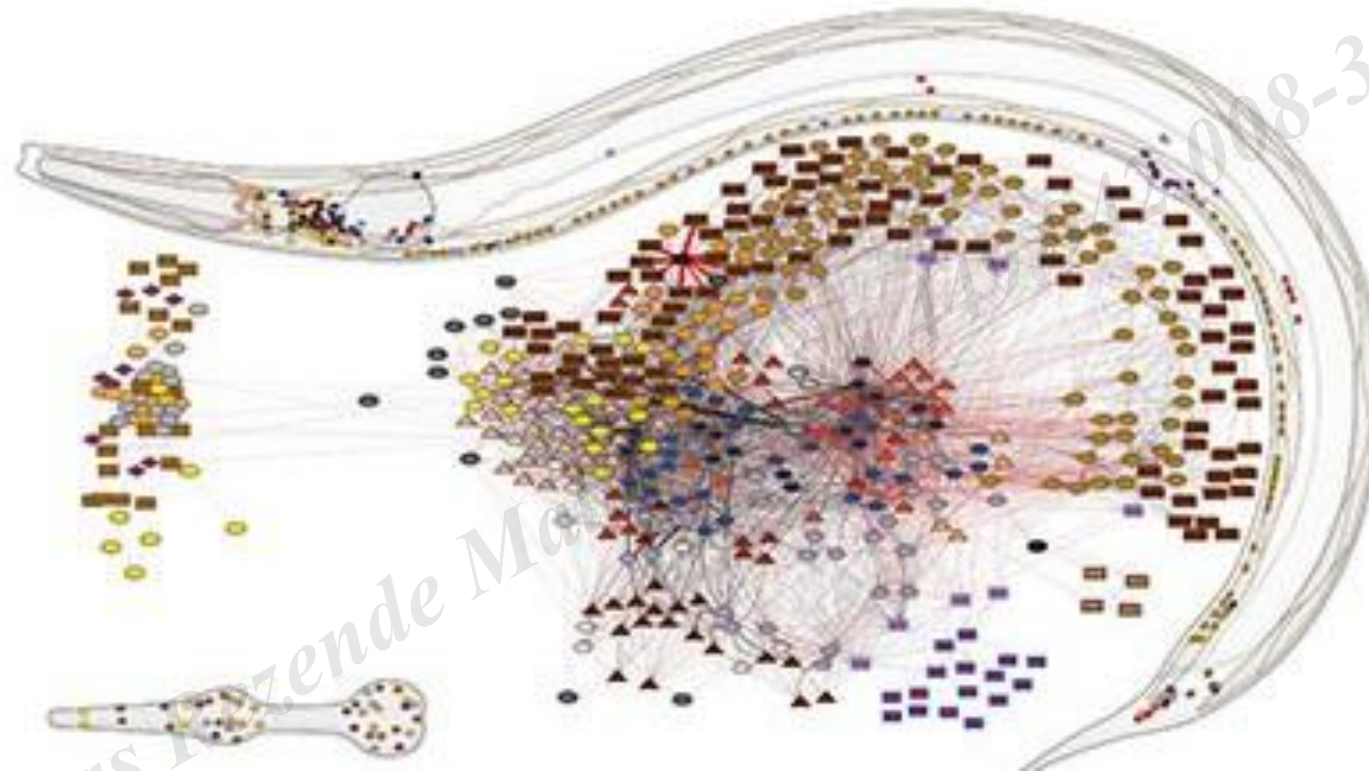
# Redes Neurais Artificiais

# Metáfora



<https://en.wikipedia.org/wiki/Myelin>

# Exemplo biológico

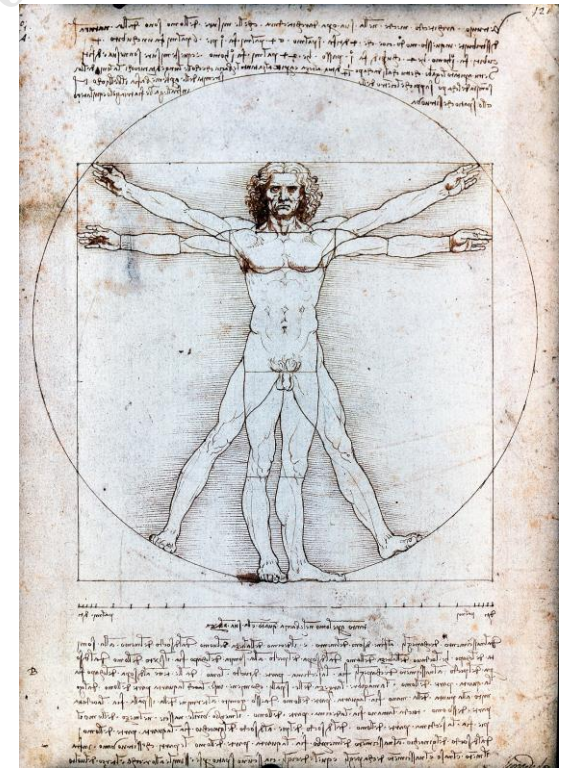
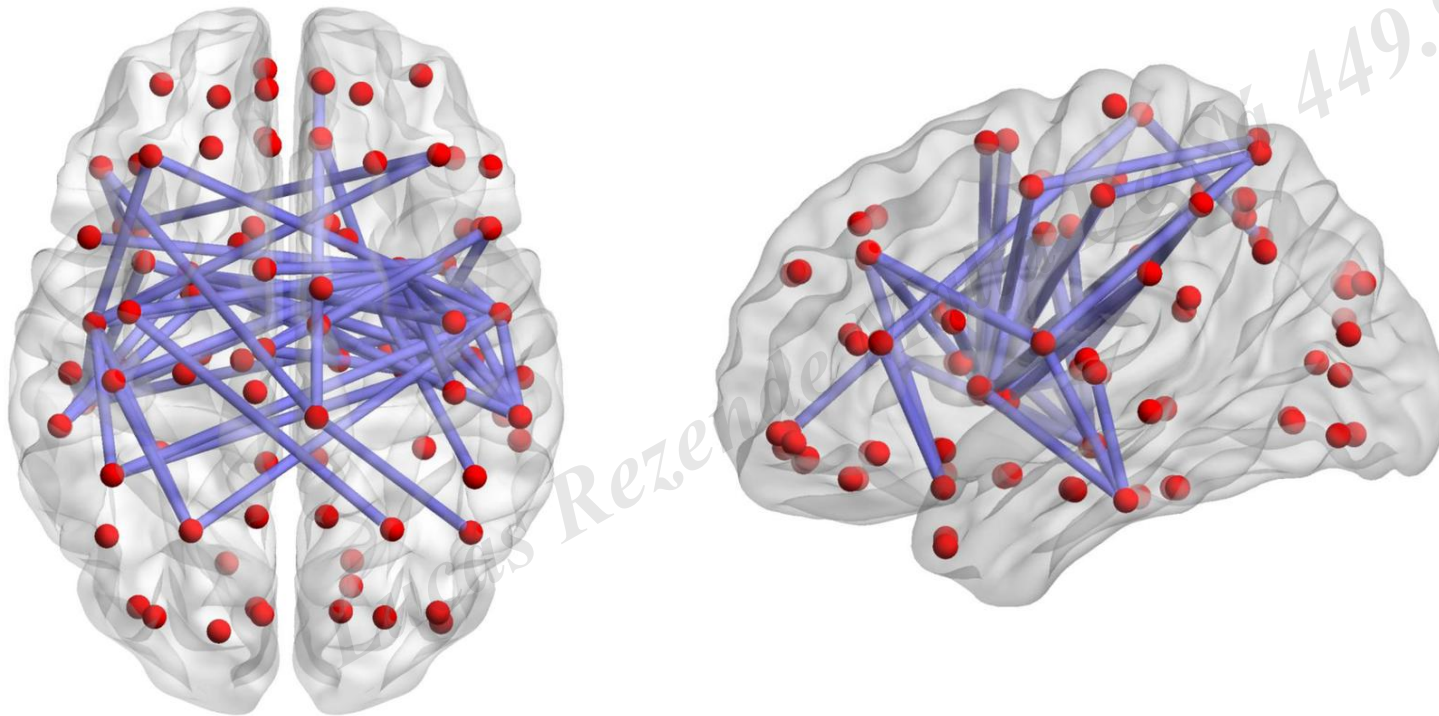


- *Nematelminto*: 302 neurônios

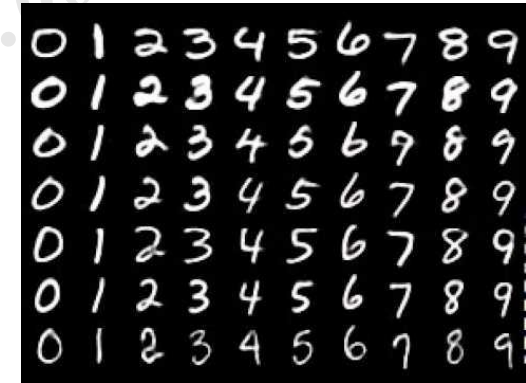


# Rede Neural Humana

- *Homo sapiens*: 100.000.000.000 de neurônios



# Onde vivem?



Redes Neurais Artificiais têm tido muito sucesso em problemas com dados pouco estruturados como imagens, áudios, textos e vídeos.

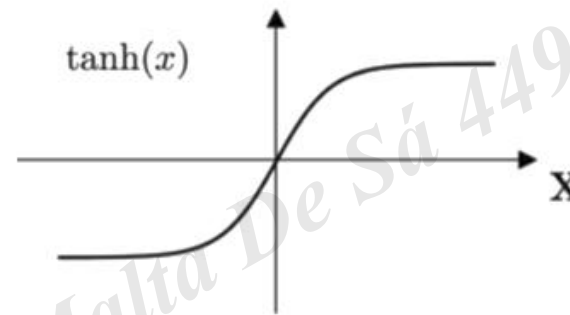
# Funções de ativação

Cada neurônio da RNA é uma função não linear de uma combinação linear dos elementos da camada anterior.

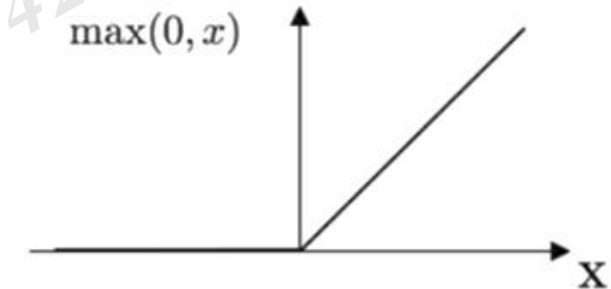
Essa função é chamada de função de ativação.

Na analogia com a rede neural natural, ele visa identificar a presença ou não de um estímulo específico.

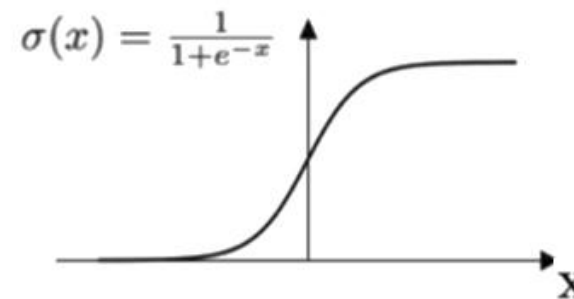
Tanh



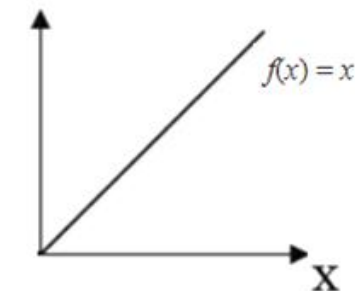
ReLU



Sigmoid



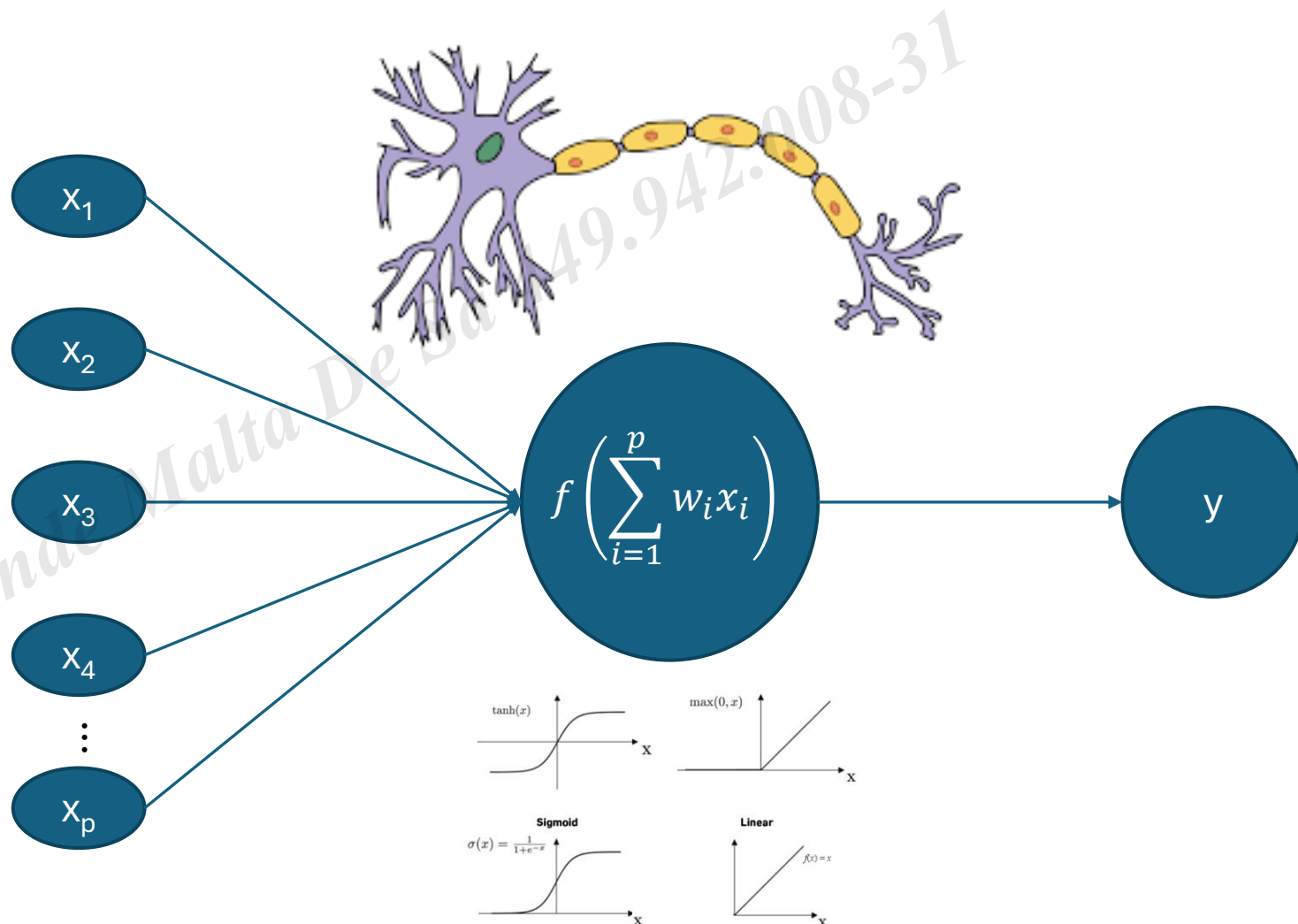
Linear



# Perceptron

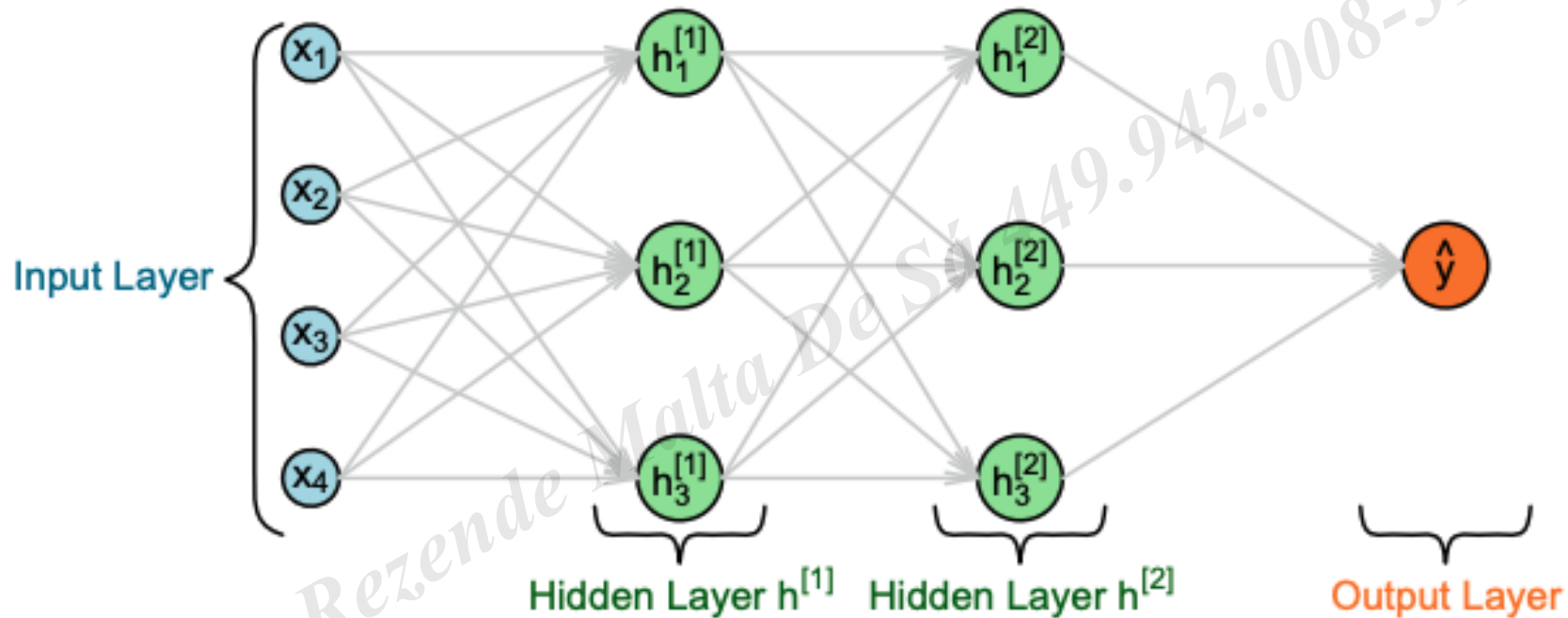
O perceptron é um tipo de neurônio bastante comum nas RNAs.

Eles podem ainda ser combinados de formas diferentes formando diferentes arquiteturas de RNAs.



# Redes Neurais Artificiais

## Combinações de perceptrons



**Fig. 2.3** A representation of a neural network with four input features, two hidden layers with three nodes each, and an output layer

Deep learning with R - Abhijit Ghatak, ed. Springer, 2019

# RNA no scikit

- Há várias implementações de RNA disponíveis.
- O Scikit Learn já tem diversas funções implementadas, inclusive RNA.
- Pacotes bem populares e poderosos são o PyTorch do facebook, e o Tensorflow/Keras, do Google.

Lucas Rezende Malta De Sá 449.942.008-31

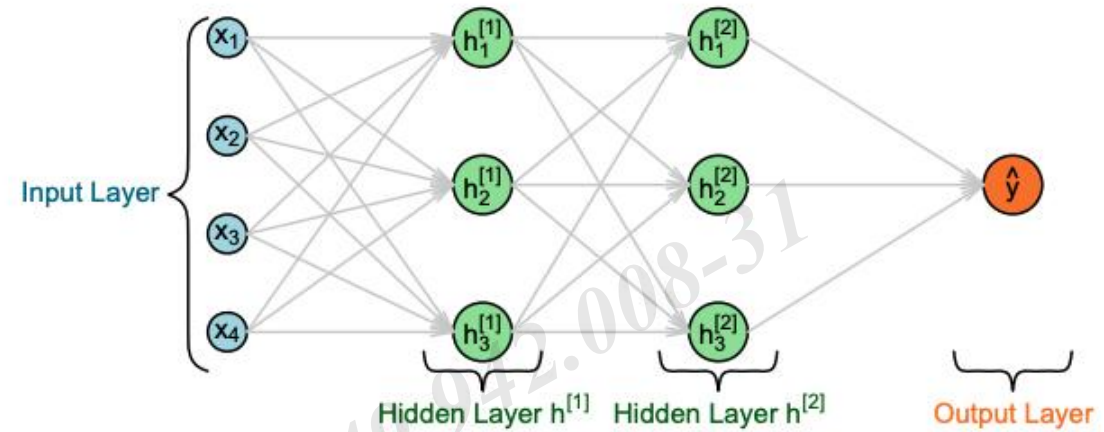


# RNA no scikit

# Primeira rede neural

```
params = { 'hidden_layer_sizes' : [3, 3, 1],  
          'activation' : 'logistic',  
          'solver' : 'adam',  
          'alpha' : 0.0,  
          'batch_size' : 10,  
          'learning_rate' : 'constant',  
          'learning_rate_init' : 0.01,  
          'random_state' : 1729,  
          'tol' : 0.0001,  
          'max_iter' : 10000,  
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)  
nn0.fit(df1[['x']], df1['y'])  
df1['pred0'] = nn0.predict(df1[['x']])
```



Podemos definir um dicionário com os parâmetros a passar para a função.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

Aqui definimos a função de ativação como 'logística'.

$$\ln\left(\frac{p}{1-p}\right)$$



# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

Aqui definimos o otimizados. 'adam' é um popular otimizador estocástico que costuma ser eficiente.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

Aqui definimos a taxa de regularização L2. No caso, 0 indica que não há penalização. Quanto maior este número, maior a ‘parcimônia’ da rede.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1['y'])
df1['pred0'] = nn0.predict(df1[['x']])
```

“batch” é a  
quantidade de linhas  
que serão  
processadas a cada  
momento.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1['y'])
df1['pred0'] = nn0.predict(df1[['x']])
```

O learning rate é semelhante ao do XGBoost. Ele pode diminuir com o tempo ou ser constante.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

Chave aleatória.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

A tolerância é um critério de parada. Quando a melhoria é menor que este valor o algoritmo para.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

O algoritmo é iterativo.  
Aqui definimos o  
número máximo de  
iterações.



# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

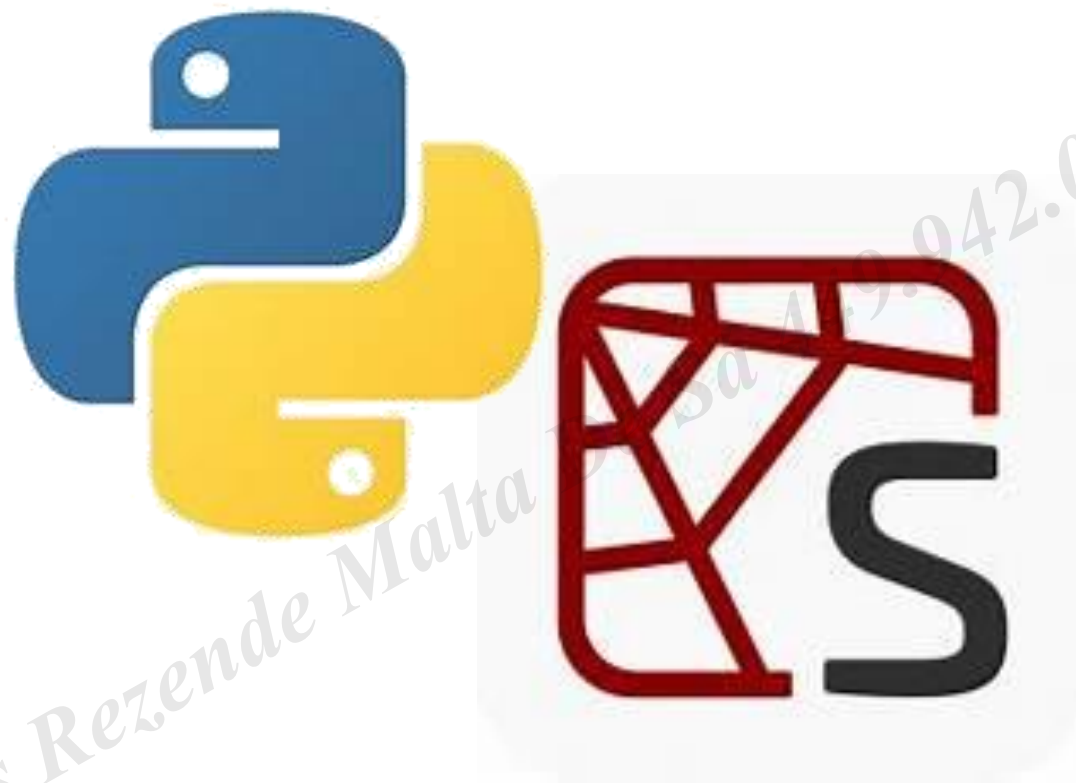
Este parâmetro é para ter um output menos extenso.

# RNA no scikit

```
# Primeira rede neural
params = { 'hidden_layer_sizes' : [1],
          'activation' : 'logistic',
          'solver' : 'adam',
          'alpha' : 0.0,
          'batch_size' : 10,
          'learning_rate' : 'constant',
          'learning_rate_init' : 0.01,
          'random_state' : 1729,
          'tol' : 0.0001,
          'max_iter' : 10000,
          'verbose' : False }
```

```
nn0 = MLPRegressor(**params)
nn0.fit(df1[['x']], df1[['y']])
df1['pred0'] = nn0.predict(df1[['x']])
```

Inserimos os dados  
com o método fit(),  
semelhante aos  
outros algoritmos do  
scikit learn.



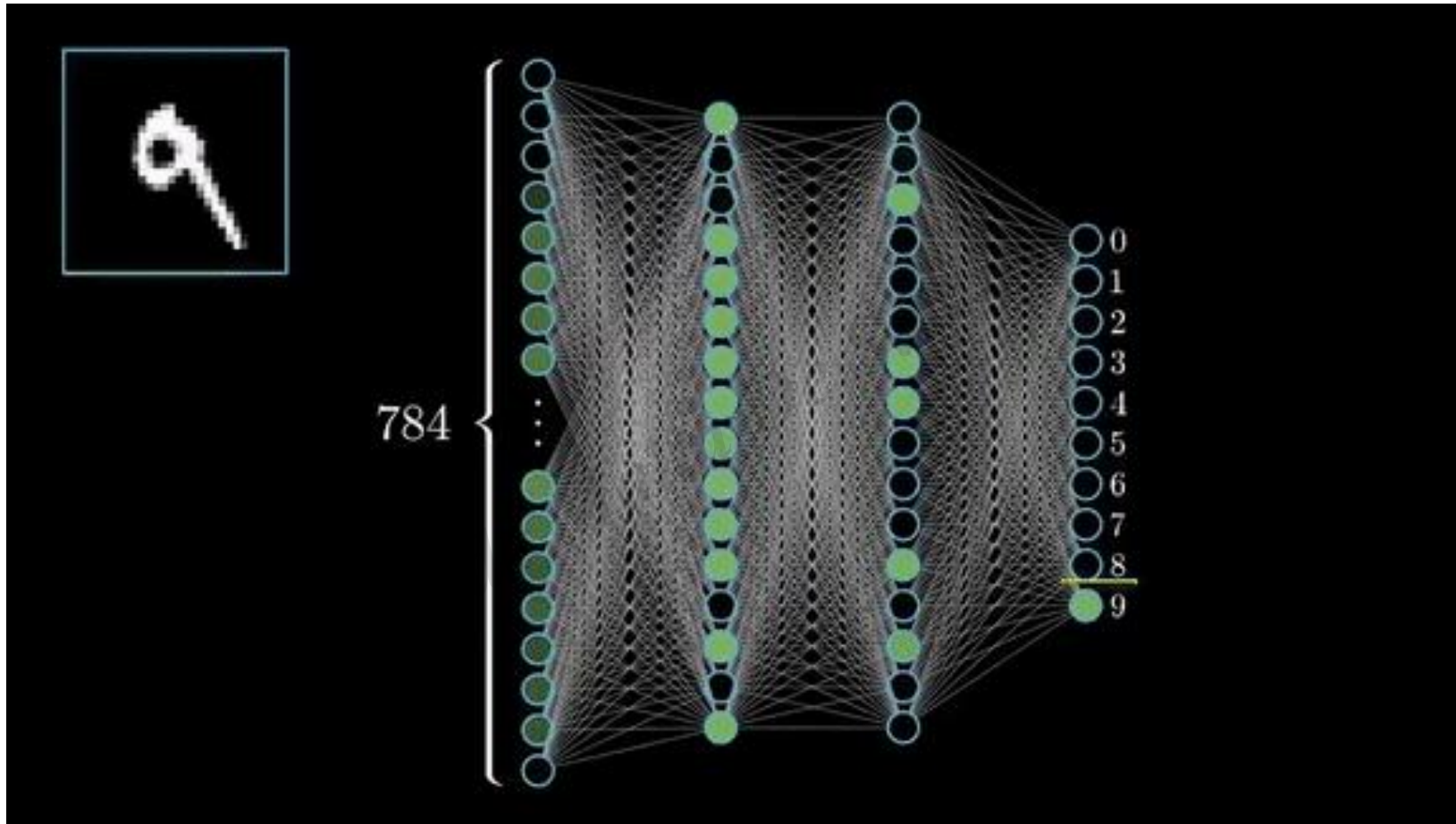
RNA\_ilustracao.py

# Exemplo revisado no titanic

Vamos abrir o Spyder







3blue1brown - <https://www.youtube.com/watch?v=aircAruvnKk>

# Demora pra rodar?

Uma breve história

Lucas Rezende Mendes De Sá 449.942.008-31



# game

## A contribuição da indústria do Game nas Redes Neurais



# Processadores

- Distância entre transístores: 14 nm
- Fio de cabelo humano: 80.000 nm
- Diâmetro do átomo de ouro: 0,3 nm

A close-up, macro photograph of a GPU (Graphics Processing Unit) chip mounted on a printed circuit board (PCB). The chip is a dark, square component with numerous gold-plated pins along its edges. The surrounding PCB is populated with various electronic components, including capacitors and other integrated circuits, all set against a blue-tinted background.

# GPU

Lucas Rezen



# Processamento com GPU



AMD  
ROCm



# TPU



# Regularização L2

Variáveis Contínuas  
SQE

$$SQE = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum \beta_i^2$$

Variáveis binárias  
Cross-Entropy

$$L = \sum y_i \log(\hat{y}_i) + \lambda \sum \beta_i^2$$

# Conclusões

- Redes Neurais são a introdução ao Deep Learning (que é um ramo muito promissor)
- São poderosas e flexíveis
- Requerem poder computacional especial (GPU / TPU)
- São famosas em dados menos estruturados (ex: imagens, áudio)







“Se você estica a corda demais, ela arrebenta, se deixa frouxa, o instrumento não toca”

Sidarta Gautama





Por hoje é só (-;



[linkedin.com/in/joao-serrajordia](https://www.linkedin.com/in/joao-serrajordia)

# MBAUSP ESALQ

## Obrigado!

João Fernando Serrajordia Rocha de Mello  
[linkedin.com/in/joao-serrajordia](https://www.linkedin.com/in/joao-serrajordia)