Introdução a Machine Learning

Conceitos, algoritmos e exemplos práticos

Cristopher Freitas Douglas Moura Eduardo Gomes

Laboratório de Computação Científica e Análise Numérica (LaCCAN)
Universidade Federal de Alagoas (UFAL)

I Curso de Capacitação do LaCCAN

Conteúdo Programático

Introdução

Os seguintes pré-requisitos são necessários:

- *Python* v3.6 ou similar.
- Instalar a biblioteca *scikit-learn*.

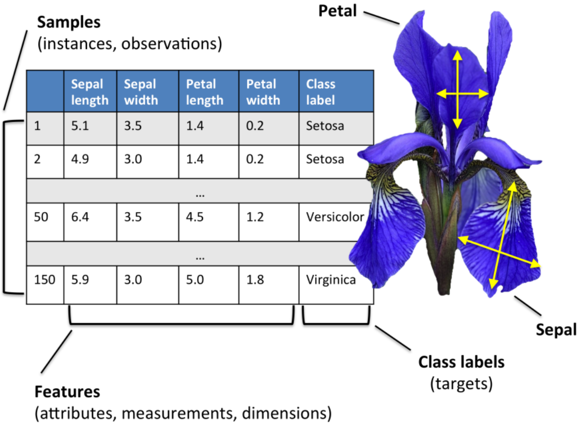
Configurando um ambiente virtual:

```
$ mkdir machine-learning && cd machine-learning
$ virtualenv -p python3 .env
$ source .env/bin/activate
$ pip install sklearn
```

É possível utilizar um interpretador *online*:
`https://repl.it/languages/python3`

Introdução

Conhecendo o primeiro dataset



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Figura: Conjunto de dados Iris.

Introdução

```python
from sklearn import datasets

iris = datasets.load_iris()

print(list(iris.keys()))
print(iris.data)
print(iris.data.shape)
print(iris.feature_names)
print(iris.target_names)
```

Algoritmo 1: Primeiros passos.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

Objetivo desta aula é fornecer o conhecimento teórico e prático para trabalhar com PCA no contexto de *machine learning*.

Ao final desta aula, você será capaz de:

- Realizar uma redução de dimensionalidade com PCA.
- Comprimir e recuperar dados com pouca perda de informações.
- Visualizar dados multivariados.

- Muitos problemas de machine learning envolvem instâncias contendo milhares ou até milhões de atributos.
- Isso não apenas torna o treinamento extremamente lento, como também torna muito mais difícil encontrar uma boa solução.
- *Principal Component Analysis* (PCA) é de longe o algoritmo de redução de dimensionalidade mais popular.

- Criada em 1901 pelo famoso estatístico Karl Pearson.
- Simplificação dos dados com pouca perda de informações.
- Informação $\rightarrow$ Variância

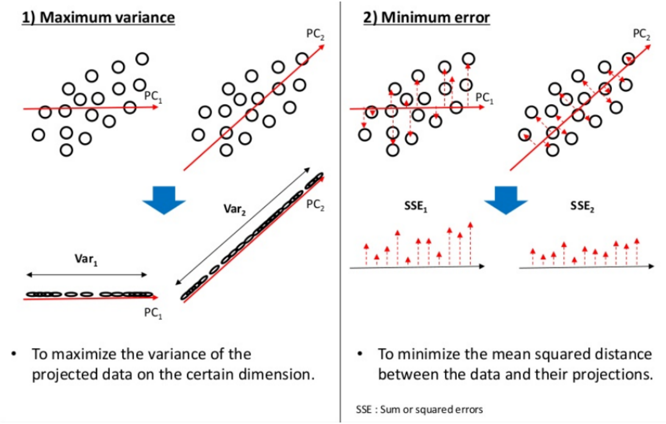


Figura: Princípios do PCA.

- Calcula-se a média (μ_j) de cada atributo x_j:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \tag{1}$$

- Centraliza-se os dados na origem[1]:

$$x_j = x_j - \mu_j \tag{2}$$

[1] Atributos com escalas diferentes necessitam ser padronizados.

Principal Component Analysis (PCA)

Como calcular o PCA

```python
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils.extmath import svd_flip

iris = datasets.load_iris()

# matriz (150 x 4).
X = iris.data

# centraliza os dados na origem.
X_centered = (X - X.mean(axis=0))
```

Algoritmo 2: Redução de dimensionalidade.

- Após a normalização dos dados, calcula-se a matriz de covariância.

$$\Sigma = \begin{bmatrix} var(x_1) & cov(x_1, x_2) & \dots & cov(x_1, x_n) \\ cov(x_2, x_1) & var(x_2) & \dots & cov(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(x_n, x_1) & cov(x_n, x_2) & \dots & cov(x_n, x_n) \end{bmatrix} \qquad (3)$$

- Covariância (ou variância conjunta) fornece um indicativo de inter-relação entre duas variáveis.

- A covariância é calculada da seguinte forma:

$$cov(x_i, x_j) = \frac{(x_i - \mu_i) \times (x_j - \mu_j)}{m - 1} \tag{4}$$

- A matriz pode ser calculada de forma matricial:

$$\Sigma = \frac{(X^T \cdot X)}{m - 1} \tag{5}$$

```
# numero de linhas - 1.
m = X_centered.shape[0] - 1

# calcula a matriz de covariancia.
sigma = np.dot(X_centered.T, X_centered) / m

# calculo da matriz de covariancia com numpy.
sigma = np.cov(X_centered.T, rowvar=False)
```

Algoritmo 3: Redução de dimensionalidade.

- Decomposição da matriz de covariância:

$$[U, S, V] = SDV(\Sigma) \tag{6}$$

- SDV (Singular Value Decomposition) é uma técnica de fatoração que decompõe a matriz em um produto de três matrizes.
- A matriz V^T contém todas as componentes principais.
- Utilizaremos as k primeiras componentes principais.
- Os dados serão projetados no novo espaço $\mathbb{R}^k$: $Z = X \cdot V^T$

```
# decomposicao em valores singulares.
U, s, V = np.linalg.svd(sigma)

# correcao da saida do svd.
U, V = svd_flip(U, V)

# projecao dos dados utilizando os autovetores.
Z = np.dot(X_centered, V.T[:, 0:2])

# visualizacao em 2D.
plt.scatter(Z[:,0], Z[:,1], c=iris.target)
plt.show() # ou plt.savefig("pca.png")
```

Algoritmo 4: Redução de dimensionalidade.

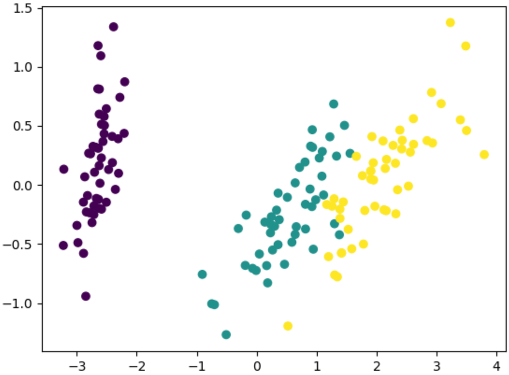


Figura: Visualização em 2 dimensões.

Como escolher o valor de k?

- No geral, 2 ou 3 dimensões em problemas de visualização.
- Em problemas de compressão gostaríamos preservar a variância dos dados.
- O valor é escolhido de acordo com o percentual de variância a ser preservado.

Exemplo: Um valor de k suficiente para preservar 95% da variância do *training set*.

```python
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

digits = load_digits()
X = digits.data

pca = PCA(n_components = .99)

Z = pca.fit_transform(X)
```

Algoritmo 5: Comprimindo dados com o PCA.

```
digits_2 = pca.inverse_transform(Z)

plt.subplot(121)
plt.imshow(digits.images[0], cmap=plt.cm.gray_r)

plt.subplot(122)
restaurado = digits_2[0].reshape(8, 8)
plt.imshow(restaurado, cmap=plt.cm.gray_r)

plt.show()
```

Algoritmo 6: Comprimindo dados com o PCA.

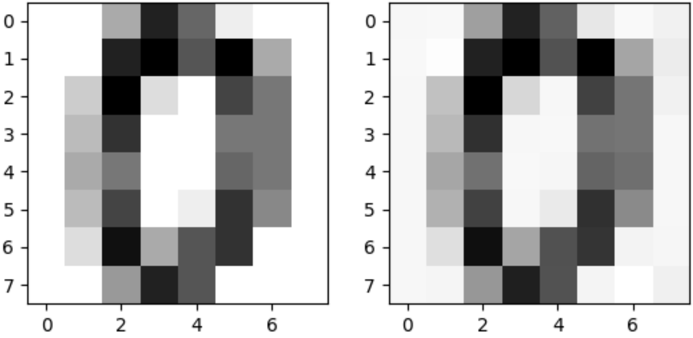


Figura: Voilá!

Observe o valor dos seguintes atributos:

- pca.components_.shape
- pca.explained_variance_ratio_
- Tente somar a taxa de variância

Foi possível reduzir 23 atributos e preservar 99% da variância!

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Numero de componentes')
plt.ylabel('Variancia preservada')
plt.ylim(0,1)

plt.show()
```

Algoritmo 7: Variância explicada.

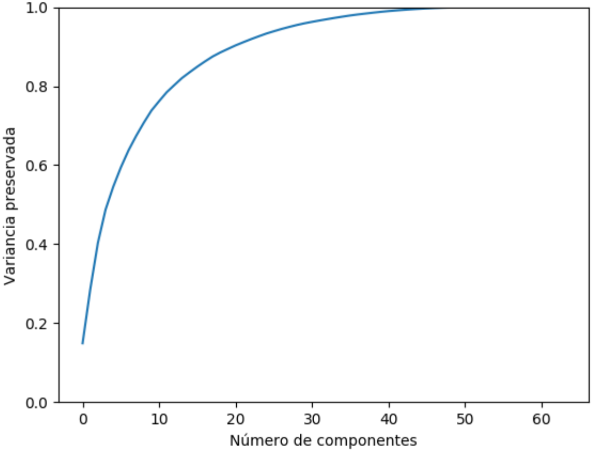


Figura: Variância em função do número de componentes.

Exercitando a mente:

1. Quais são as principais motivações para reduzir a dimensionalidade de um conjunto de dados? Quais são as principais desvantagens?

2. Como você pode avaliar o desempenho de um algoritmo de redução de dimensionalidade no seu conjunto de dados?

3. Faz algum sentido encadear dois algoritmos de redução de dimensionalidade diferentes?

Support Vector Machine (SVM)

Objetivo desta aula é explicar o conceito de *Support Vector Machines* e como utilizá-lo para realizar classificações.

Ao final desta aula, você será capaz de:

- Realizar classificações lineares e não-lineares com SVM.
- Ter uma noção básica de como trabalhar com *kernels*.

- *Support Vector Machines* (SVM) é um método de aprendizado supervisionado.
- Capaz de executar classificação linear ou não-linear, regressão e detecção de *outliers*.
- Muito poderoso e amplamente utilizado tanto na indústria quanto na academia:
- Detecção de face, categorização de textos e aplicações em Bioinformática.

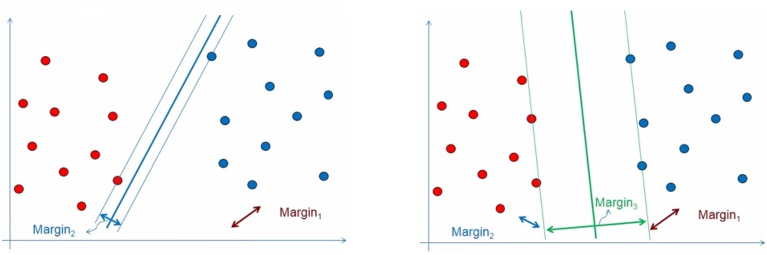


Figura: Ideia por trás do *Support Vector Machine*.

A margem de um classificador é definida como a menor distância entre os exemplos do conjunto de treinamento e o hiperplano utilizado na separação desses dados em classes.

- Qual margem é melhor? Por quê?

O modelo de classificação irá predizer a classe de uma nova instância $x^{(i)}$ computando uma função de decisão:

$$\hat{y}^{(i)} = \begin{cases} +1, & \text{se } w^T \cdot x^{(i)} + b \geq +1 \\ -1, & \text{se } w^T \cdot x^{(i)} + b \leq -1 \end{cases} \tag{7}$$

Que pode ser reescrita como:

$$\hat{y}^{(i)}(w^T \cdot x^{(i)} + b) \geq +1 \tag{8}$$

Support Vector Machine (SVM)
Classificação de margem rígida

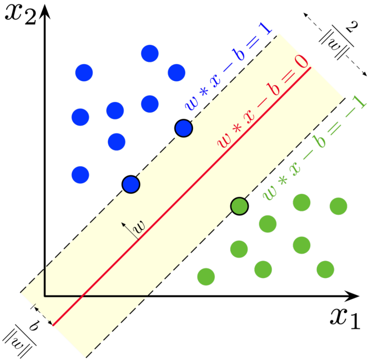


Figura: O tamanho da margem é $2 \cdot \frac{1}{||w||}$.

Função Objetivo:

$$\underset{w,b}{\text{Minimize}} \qquad \frac{1}{2} \cdot ||w||^2 \tag{9}$$

Sujeito a

$$\hat{y}(w^T \cdot x^{(i)} + b) \geq 1 \qquad \forall i = 1, ..., m \tag{10}$$

Se impusermos estritamente que todas as instâncias estejam fora da margem, isso é chamado de classificação de **margem rígida**.

```python
from sklearn import datasets
from sklearn.svm import SVC

iris = datasets.load_iris()

X = iris.data[:, (2, 3)]
y = iris.target

setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

svm_clf = SVC(kernel="linear", C=float("inf"))
svm_clf.fit(X, y)
```

Algoritmo 8: SVM com margem rígida.

```python
# largura e comprimento
x_test = [[2.5, 0.8]]

# vamos predizer o valor de y
h = svm_clf.predict(x_test)

# qual e a classe de x_test
print("Qual e a classe", iris.target\_names[h])
```

Algoritmo 9: SVM com margem rígida.

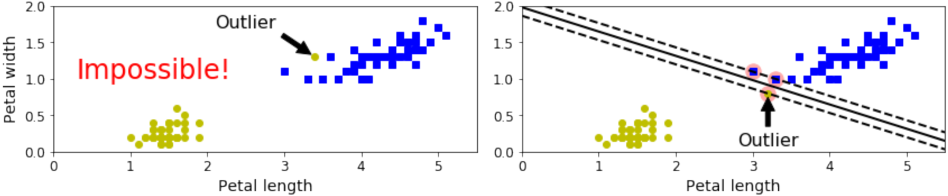


Figura: SVMs são sensíveis a *outliers*.

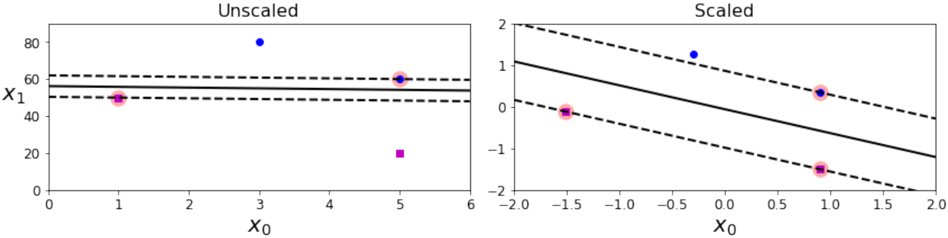


Figura: SVMs são sensíveis a *feature scales*.

Na classificação de **margem suave** o objetivo é encontrar um *trade-off* entre o tamanho da margem e o limite de violações.

Função Objetivo:

$$\underset{w,b,\zeta}{\text{Minimize}} \qquad \frac{1}{2} \cdot ||w||^2 + C \cdot \sum_{j=1}^{m} \zeta^{(i)} \tag{11}$$

Sujeito a

$$\hat{y}(w^T \cdot x^{(i)} + b) \geq 1 - \zeta^{(i)} \qquad \forall i = 1, ..., m \tag{12}$$

$$\zeta^{(i)} \geq 0 \tag{13}$$

```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt

iris = datasets.load_iris()

X = iris.data[:, (2, 3)]
y = iris.target
```

Algoritmo 10: SVM com margem suave.

```python
virginica_or_versicolor = (y == 2) | (y == 1)
X = X[virginica_or_versicolor]
y = y[virginica_or_versicolor]

svm_clf = SVC(kernel="linear", C=100)

X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size=0.3, random_state=0)

svm_clf.fit(X_train, y_train)

h = svm_clf.predict(X_test)
```

Algoritmo 11: SVM com margem suave.

Support Vector Machine (SVM)

Classificação de margem suave

```
cm = confusion_matrix(y_test, h)

plot_confusion_matrix(svm_clf, X_test, y_test,
    display_labels=iris.target_names[1:],
                                        cmap=plt.cm.Blues)

# caso queira salvar em png
plt.savefig("matrix_conf.png")
plt.show()
```

Algoritmo 12: SVM com margem suave.

Exemplos de funções Kernels comumente utilizadas:

Linear	$K(a, b) = aT \cdot b$
Polinomial	$K(a, b) = (\gamma a^T \cdot b + r)^d$
Gaussiano	$K(a, b) = exp(-\gamma\|\|a - b\|\|^2)$
Sigmoid	$K(a, b) = tanh(\gamma a^T \cdot b + r)$

Outras como string, chi-quadrado, intersecção de histograma etc.

SVC(kernel="poly", degree=3, coef0=1, C=5)

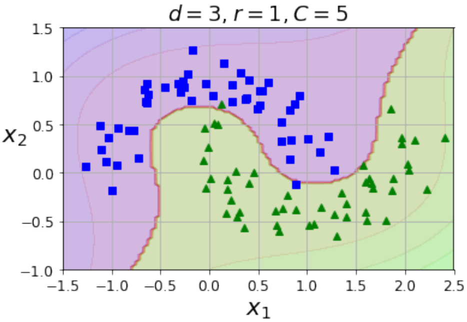


Figura: Funções mais complexas para realizar separações.

- O SVM é aplicável diretamente somente para problemas binários (duas classes).
- Estratégias de redução: *one-versus-all* e *one-versus-one*.
- Por sorte, bibliotecas, como *sklearn*, já implementam a classificação multiclasse.

Exercitando a mente:

1. Qual é a ideia fundamental por trás do SVM?
2. Por quê é importante que os atributos estejam na mesma escala?
3. Em quais destes datasets posso utilizar SVM?
   - ( ) Linearmente separáveis.
   - ( ) Não-linearmente separáveis.
   - ( ) *Dataset* composto de milhões de instâncias.

Objetivo desta aula é apresentar os conceitos fundamentais para trabalhar com o algoritmo *decision tree*.

Ao final desta aula, você será capaz de:

- Treinar, visualizar e realizar classificações utilizando árvores de decisão.
- Realizar tarefas de regressão.

- *Decision Tree* é um modelo hierárquico para aprendizado supervisionado, pode ser usado para classificação e regressão.
- Fornece uma interpretação fácil das regras que levaram à classificação.
- Cada nó folha possui um rótulo de saída, que no caso de classificação é o código da classe e em regressão é um valor númerico.

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz

iris = load_iris()

X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

Algoritmo 13: Criando um classificador.

```
export_graphviz(
    tree_clf,
    out_file="iris_tree.dot",
    feature_names=iris.feature_names[2:],
    class_names=iris.target_names,
    rounded=True,
    filled=True
)
```

Algoritmo 14: Visualizando a árvore.

- O algoritmo anterior exporta a árvore de decisão para o formato DOT.
- É possível converter para outros formados, como PNG ou PDF.
- Para converter em formato PNG:

Terminal de comandos

$ dot -Tpng iris_tree.dot -o iris_tree.png

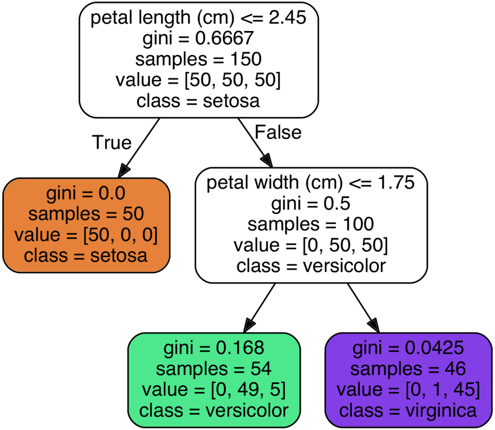


Figura: Árvore de decisão.

Vamos tentar classificar as seguintes flores:

(a) Petal Length = 4.50 Petal width = 1.50
(b) Petal Length = 1.50 Petal width = 0.43
(c) Petal Length = 6.20 Petal width = 2.24

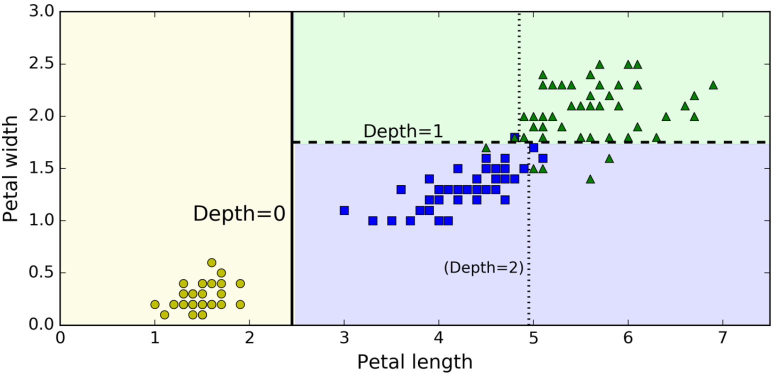


Figura: Limites de decisão da árvore de decisão.

```
print(tree_clf.predict_proba([[4.5, 1.5]]))
print(tree_clf.predict_proba([[1.5, 0.43]]))
print(tree_clf.predict_proba([[6.2, 2.24]]))
```

Algoritmo 15: Estimando probabilidades.

Calculando o índice Gini:

$$G_i = 1 - \sum_{k=1}^{n} P_{i,k}^2 \tag{14}$$

$p_{i,k}$ é a taxa de instâncias da classe k nas instâncias de treino do i-ésimo nó.

Por exemplo,

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168. \tag{15}$$

Calculando a entropia:

$$H_i = -\sum_{k=1}^{n} p_{i,k} log(P_{i,k}) \qquad \forall p_{i,k} \neq 0 \tag{16}$$

Por exemplo,

$$-\frac{49}{54} log(\frac{49}{54}) - \frac{5}{54} log(\frac{5}{54}) \approx 0.31$$

Qual medida de impureza utilizar?

- A impureza de Gini é um pouco mais rápida de se calcular.
- Tende a isolar a classe mais frequente em seu próprio ramo da árvore.
- Por outro lado, a entropia tende a produzir árvores um pouco mais balanceadas.

- Scikit-Learn implementa o algoritmo CART (*classification and regression tree*)
- Função de custo:

$$j(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{17}$$

Onde:

$G_{left/right}$ medida de impureza do subconjunto *left/right*.
$m_{left/right}$ número de instâncias no subconjunto *left/right*.

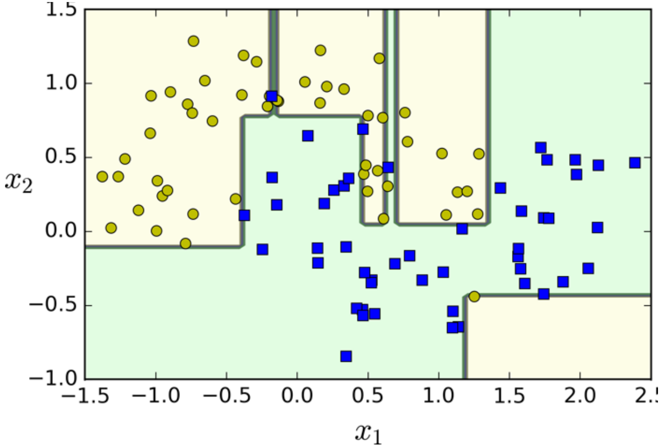


Figura: Overfitting.

Conclusão

Fim!

Conclusão

Material de referência:

- Leitura recomendada:
  Ethem Alpaydin. *Introduction to Machine Learning* (2014).
- Tutoriais:
  https://scikit-learn.org/stable/tutorial
  https://github.com/ageron/handson-ml
- Curso do Coursera:
  https://www.coursera.org/learn/machine-learning/