

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

- 1 Introdução
- 2 *Principal Component Analysis (PCA)*
- 3 *Support Vector Machine (SVM)*
- 4 *Decision Tree*
- 5 Conclusão

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

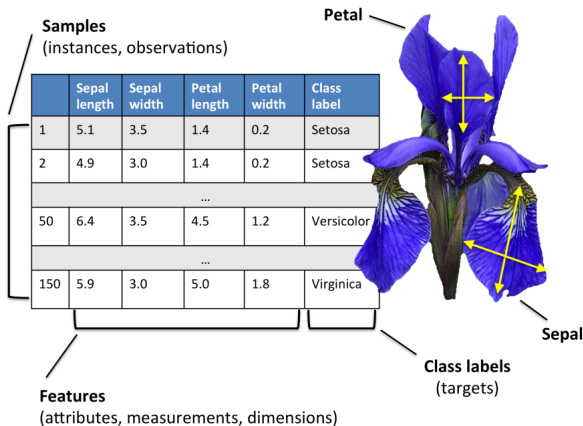


Figura: Conjunto de dados Iris.

```
1 from sklearn.datasets import load_iris
2
3 iris = datasets.load_iris()
4
5 print(list(iris.keys()))
6 print(iris.data)
7 print(iris.data.shape)
8 print(iris.feature_names)
9 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.

Principal Component Analysis (PCA)

Motivação

- 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.

Principal Component Analysis (PCA)

Motivação

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



Principal Component Analysis (PCA)

Motivação

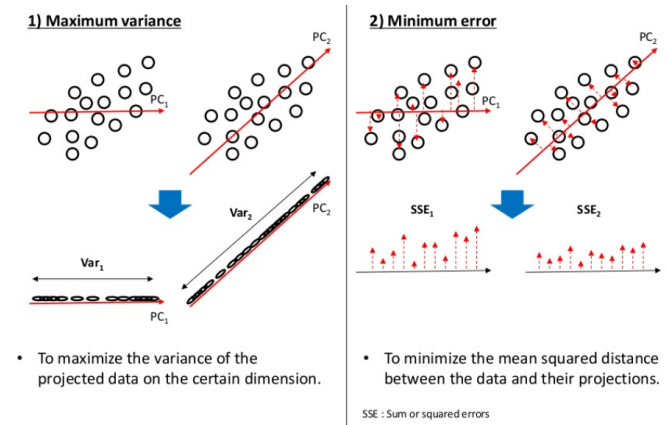


Figura: Princípios do PCA.

Principal Component Analysis (PCA)

Como calcular o PCA - Pré-processamento dos dados

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

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

- Centraliza-se os dados na origem¹:

$$x_j = x_j - \mu_j \quad (2)$$

¹Atributos com escalas diferentes necessitam ser padronizados.

Principal Component Analysis (PCA)

Como calcular o PCA

```
1 from sklearn import datasets
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.utils.extmath import svd_flip
5
6 iris = datasets.load_iris()
7
8 # matriz (150 x 4).
9 X = iris.data
10
11 # centraliza os dados na origem.
12 X_centered = (X - X.mean(axis=0))
```

Algoritmo 2: Redução de dimensionalidade.

Principal Component Analysis (PCA)

Como calcular o PCA - Algoritmo

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

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

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

Principal Component Analysis (PCA)

Como calcular o PCA - Algoritmo

- A covariância é calculada da seguinte forma:

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

- A matriz pode ser calculada de forma matricial:

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

Principal Component Analysis (PCA)

Como calcular o PCA

```
13 # numero de linhas - 1.  
14 m = X_centered.shape[0] - 1  
15  
16 # calcula a matriz de covariancia.  
17 sigma = np.dot(X_centered.T, X_centered) / m  
18  
19 # calculo da matriz de covariancia com numpy.  
20 sigma = np.cov(X_centered.T, rowvar=False)
```

Algoritmo 3: Redução de dimensionalidade.

Principal Component Analysis (PCA)

Como calcular o PCA - Algoritmo

- Decomposição da matriz de covariância:

$$[U, S, V] = SDV(\Sigma) \quad (6)$$

- 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 $\mathbb{R}^k$: $Z = X \cdot V^T$

Principal Component Analysis (PCA)

Como calcular o PCA

```
21 # decomposicao em valores singulares.  
22 U, s, V = np.linalg.svd(sigma)  
23  
24 # correcao da saida do svd.  
25 U, V = svd_flip(U, V)  
26  
27 # projecao dos dados utilizando os autovetores.  
28 Z = np.dot(X_centered, V.T[:, 0:2])  
29  
30 # visualizacao em 2D.  
31 plt.scatter(Z[:,0], Z[:,1], c=iris.target)  
32 plt.show()
```

Algoritmo 4: Redução de dimensionalidade.

Principal Component Analysis (PCA)

Como calcular o PCA - Algoritmo

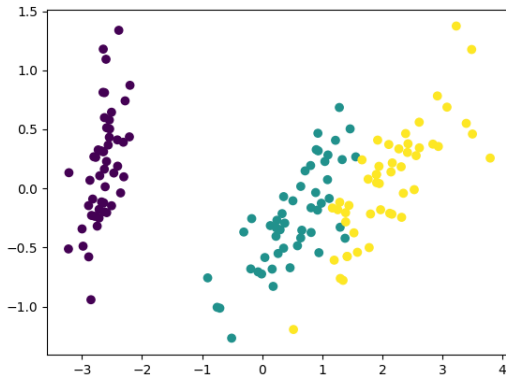


Figura: Visualização em 2 dimensões.

Principal Component Analysis (PCA)

Como calcular o PCA - Número de componentes principais

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*.

Principal Component Analysis (PCA)

PCA para compressão de dados

```
1 from sklearn.decomposition import PCA
2 from sklearn.datasets import load_digits
3 import matplotlib.pyplot as plt
4
5 digits = load_digits()
6 X = digits.data
7
8 pca = PCA(n_components = .99)
9
10 Z = pca.fit_transform(X)
```

Algoritmo 5: Comprimindo dados com o PCA.

Principal Component Analysis (PCA)

PCA para compressão de dados

```
11 digits_2 = pca.inverse_transform(Z)
12
13 plt.subplot(121)
14 plt.imshow(digits.images[0], cmap=plt.cm.gray_r)
15
16 plt.subplot(122)
17 restaurado = digits_2[0].reshape(8, 8)
18 plt.imshow(restaurado, cmap=plt.cm.gray_r)
19
20 plt.show()
```

Algoritmo 6: Comprimindo dados com o PCA.

Principal Component Analysis (PCA)

PCA para compressão de dados

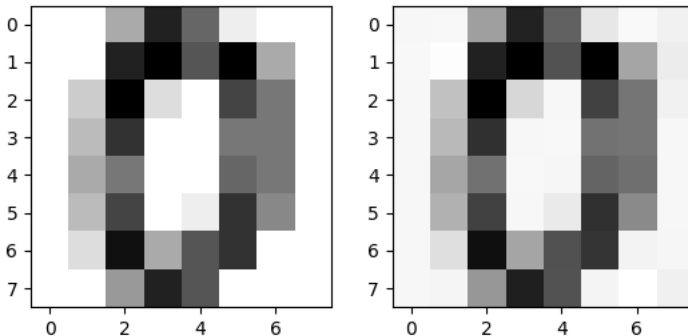


Figura: Voilà!

Principal Component Analysis (PCA)

PCA para compressão de dados

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!

Principal Component Analysis (PCA)

PCA para compressão de dados

```
1 plt.plot(np.cumsum(pca.explained_variance_ratio_))
2 plt.xlabel('Numero de componentes')
3 plt.ylabel('Variância preservada')
4 plt.ylim(0,1)
5
6 plt.show()
```

Algoritmo 7: Variância explicada.

Principal Component Analysis (PCA)

PCA para compressão de dados

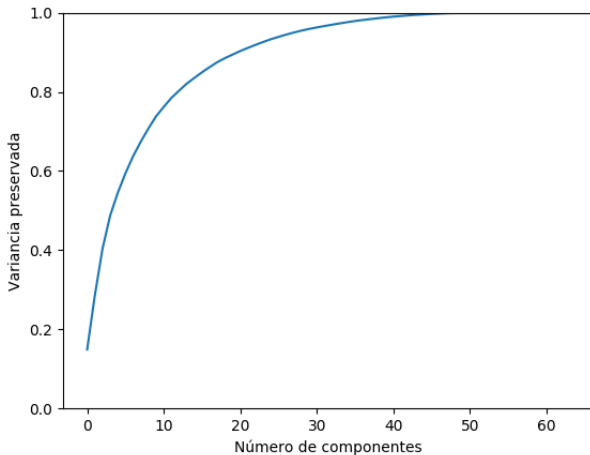


Figura: Variância em função do número de componentes.

Principal Component Analysis (PCA)

PCA para compressão de dados- Exercícios

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)

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 Machine (SVM)

Motivação

- *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.

Support Vector Machine (SVM)

Motivação

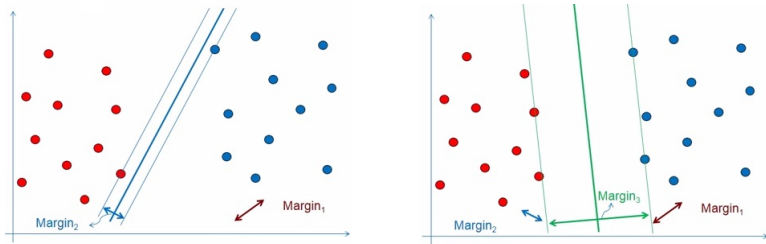


Figura: Ideia por trás do *Support Vector Machine*.

Support Vector Machine (SVM)

Motivação

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?

Support Vector Machine (SVM)

Formulação

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

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

Que pode ser reescrita como:

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

Support Vector Machine (SVM)

Formulação

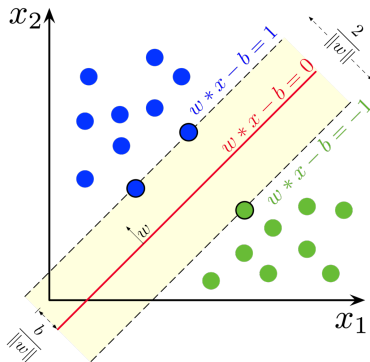


Figura: O tamanho da margem é $2 \cdot \frac{1}{\|w\|}$.

Support Vector Machine (SVM)

Formulação - Classificação de margem rígida

Função Objetivo:

$$\underset{w,b}{\text{Minimize}} \quad \frac{1}{2} \cdot \|w\|^2 \quad (9)$$

Sujeito a

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

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

Support Vector Machine (SVM)

Formulação

```
1 from sklearn import datasets
2 from sklearn.svm import SVC
3
4 iris = datasets.load_iris()
5
6 X = iris.data[:, (2, 3)]
7 y = iris.target
8
9 setosa_or_versicolor = (y == 0) | (y == 1)
10 X = X[setosa_or_versicolor]
11 y = y[setosa_or_versicolor]
12
13 svm_clf = SVC(kernel="linear", C=float("inf"))
14 svm_clf.fit(X, y)
```

Algoritmo 8: SVM com margem rígida.

Support Vector Machine (SVM)

Formulação

```
15 # largura e comprimento
16 x_test = [[2.5, 0.8]]
17
18 # vamos prever o valor de y
19 h = svm_clf.predict(x_test)
20
21 # qual e a classe de x_test
22 print("Qual e a classe", iris.target_names[h])
```

Algoritmo 9: SVM com margem rígida.

Support Vector Machine (SVM)

Formulação - Classificação de margem rígida

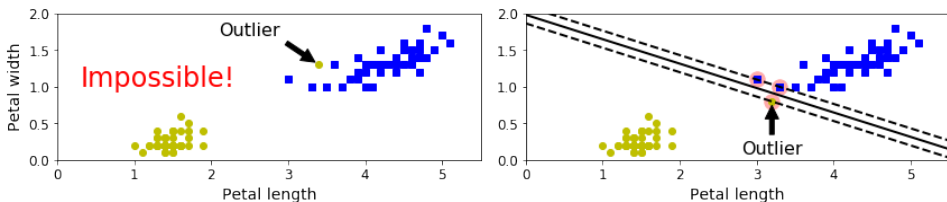


Figura: SVMs são sensíveis a *outliers*.

Support Vector Machine (SVM)

Formulação - Classificação de margem rígida

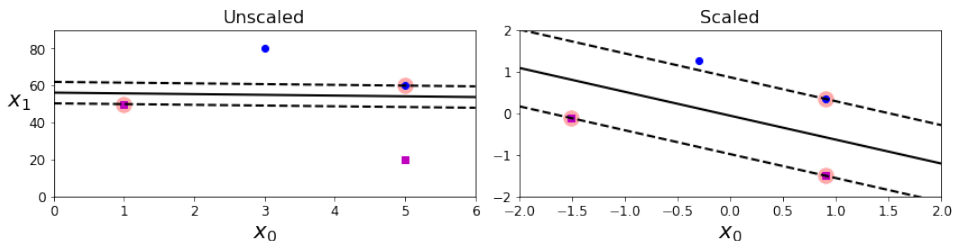


Figura: SVMs são sensíveis a *feature scales*.

Support Vector Machine (SVM)

Formulação - Classificação de margem suave

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

Função Objetivo:

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

Sujeito a

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

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

Support Vector Machine (SVM)

Formulação

```
1 from sklearn import datasets
2 from sklearn.model_selection import
   train_test_split
3 from sklearn.svm import SVC
4 from sklearn.metrics import confusion_matrix
5 from sklearn.metrics import plot_confusion_matrix
6
7 iris = datasets.load_iris()
8
9 X = iris.data[:, (2, 3)]
```

Algoritmo 10: SVM com margem suave.

Support Vector Machine (SVM)

Formulação

```
1 cm = confusion_matrix(y_test, h)
2
3 plot_confusion_matrix(svm_clf, X_test, y_test,
4                       display_labels=iris.target_names[1:],
5                           cmap=plt.cm.Blues)
6 plt.show()
```

Algoritmo 11: SVM com margem suave.

Support Vector Machine (SVM)

SVM com Kernel

Exemplos de funções Kernels comumente utilizadas:

Linear $K(a, b) = a^T \cdot b$

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)`

Support Vector Machine (SVM)

SVM com Kernel

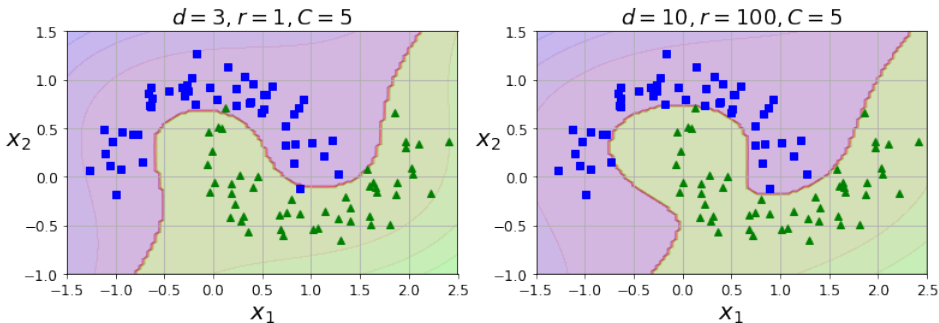


Figura: Funções mais complexas para realizar separações.

Support Vector Machine (SVM)

Classificação Multiclasse

- 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.

Support Vector Machine (SVM)

Classificação Multiclasse- Exercícios

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 numérico.

Decision Tree

Treinando e visualizando uma árvore

```
1 from sklearn.datasets import load_iris
2 from sklearn.tree import DecisionTreeClassifier
3
4 iris = load_iris()
5
6 X = iris.data[:, 2:] # petal length and width
7 y = iris.target
8
9 tree_clf = DecisionTreeClassifier(max_depth=2)
10 tree_clf.fit(X, y)
```

Algoritmo 12: Criando um classificador.

Decision Tree

Treinando e visualizando uma árvore

```
1 from sklearn.tree import export_graphviz
2
3 export_graphviz(
4     tree_clf,
5     out_file="iris_tree.dot",
6     feature_names=iris.feature_names[2:],
7     class_names=iris.target_names,
8     rounded=True,
9     filled=True
10 )
```

Algoritmo 13: Visualizando a árvore.

Decision Tree

Treinando e visualizando uma árvore

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

Terminal de comandos

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

Decision Tree

Treinando e visualizando uma árvore

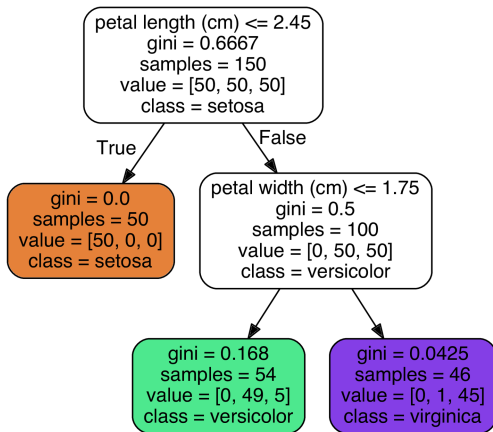


Figura: Árvore de decisão.

Decision Tree

Treinando e visualizando uma árvore

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

Decision Tree

Treinando e visualizando uma árvore

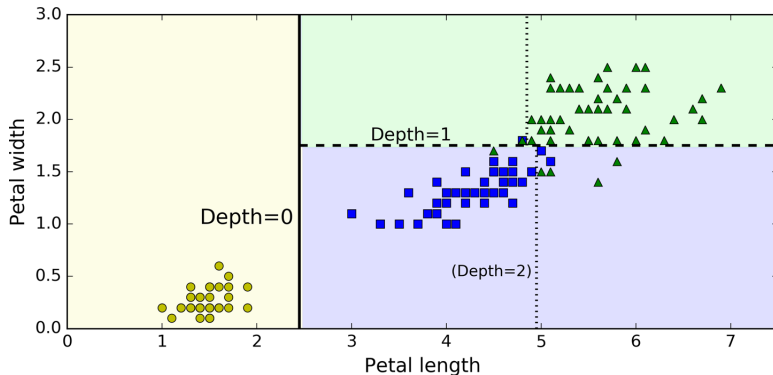


Figura: Limites de decisão da árvore de decisão.

Decision Tree

Treinando e visualizando uma árvore

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

Algoritmo 14: Estimando probabilidades.

Calculando o índice Gini:

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

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

Por exemplo,

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

Calculando a entropia:

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

Por exemplo,

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

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.

Decision Tree

Treinando e visualizando uma árvore

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

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

Onde:

$G_{\text{left/right}}$ medida de impureza do subconjunto *left/right*.

$m_{\text{left/right}}$ número de instâncias no subconjunto *left/right*.

Decision Tree

Treinando e visualizando uma árvore

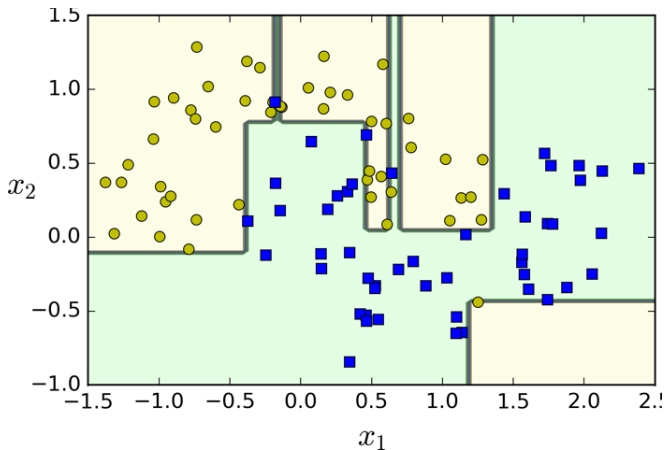


Figura: Overfitting.

Fim!

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/>