

INTELIGÊNCIA ARTIFICIAL  
E COMPUTACIONAL

# REPRESENTAÇÃO DA INCERTEZA

ERICK GALANI MAZIERO



6

## LISTA DE FIGURAS

Figura 6.1. Exemplo de interpretador interativo do Python instalado em um sistema Linux (distribuição Ubuntu).....	9
Figura 6.2. Tela inicial do Jupyter. Por exemplo, é listado um notebook criado, com nome "Experiment.ipynb" .....	11
Figura 6.3. Iniciando um novo código Python.....	12
Figura 6.4. Exemplo de edição de código e execução bloco a bloco .....	12
Figura 6.5. Exemplo de execução de dois blocos de códigos e suas respectivas saídas.....	15
Figura 6.6. Gráfico gerado com os dados da avaliação do SVM com três <i>kernels</i> ...	17
Figura 6.7. Gráfico gerado com os dados da avaliação das árvores de decisão com variações da profundidade máxima.....	19
Figura 6.8. Exemplo de gráfico de avaliação do modelo de regressão linear .....	20

**LISTA DE CÓDIGOS-FONTE**

Código-Fonte 6.1. Importação dos módulos Python para os experimentos com aprendizado automático .....	14
Código-Fonte 6.2. Carregamento do <i>dataset</i> de câncer de mama e exibição de seus atributos e classes (discreta) .....	14
Código-Fonte 6.3. Carregamento do <i>dataset</i> de diabetes e exibição de seus atributos e classes (contínua) .....	14
Código-Fonte 6.4. Separação dos datasets nos conjuntos de treino e test.....	16
Código-Fonte 6.5. Treinamento e avaliação do SVM .....	16
Código-Fonte 6.6. Treinamento e avaliação de árvores de decisão .....	18
Código-Fonte 6.7. Treinamento e avaliação do algoritmo de regressão linear com o <i>dataset</i> de diabetes .....	20

**SUMÁRIO**

6 REPRESENTAÇÃO DA INCERTEZA .....	5
6.1 Incerteza.....	5
6.2 Lidando com a incerteza: Probabilidades.....	6
6.3 O teorema de Bayes.....	7
6.3 Hands On! Aplicando o aprendizado automático! .....	8
6.3.1 Ajustando o ambiente .....	8
6.3.2 Dados do experimento .....	10
6.3.3 Executando e interpretando o experimento .....	11
REFERÊNCIAS .....	22
GLOSSÁRIO .....	23

## 6 REPRESENTAÇÃO DA INCERTEZA

Neste capítulo veremos como lidar com a incerteza em aplicações de aprendizado de máquina. Mesmo as decisões humanas lidam, muitas das vezes implicitamente, com o incerto, pois não se pode prever todas as variáveis que levam a uma decisão.

Além da representação da incerteza, aproveitaremos o capítulo para uma abordagem prática ao aprendizado automático com o uso da linguagem *Python* e um módulo de *machine learning* amplamente utilizado, o *Scikit-Learn*. Os experimentos serão realizados, inclusive, no ambiente do *Jupyter Notebook*, que permite, dentre outras funcionalidades, visualização gráfica do *dataset* e execução do código, bloco a bloco.

### 6.1 Incerteza

Nem sempre temos acesso a exatamente todos os fatos de um determinado ambiente e evento. Se o tivéssemos, poderíamos prever tudo o que aconteceria com certa exatidão. Em ambientes reais, dificilmente conseguiremos prever com exatidão tudo que acontecerá. Isso se reflete nas interpretações e decisões de agentes inteligentes, inclusive os gerados com aprendizado de máquina.

Diversas variáveis não podem ser observadas, por estarem além do escopo do problema que estamos tentando resolver com inteligência artificial. Assim, temos de recorrer a métodos que levem em consideração as incertezas que permeiam uma predição automática.

Um agente inteligente poderia ter planos de contingência para cada situação incerta, devido aos dados parciais do ambiente a que tem acesso, mas essa solução pode não ser viável, pois a quantidade de situações não previstas pode crescer exponencialmente. Inclusive, pode-se não conseguir, a priori, um plano para cada imprevisto.

Imagine um sistema especialista que faça a triagem de paciente em um pronto-socorro. Sua base de conhecimento é composta de conhecimentos extraídos de especialistas em triagem e consiste basicamente de conjuntos de sintomas que

indicam possíveis “quadros” dos pacientes. Um paciente que chegue ao pronto-socorro com um conjunto de sintomas C1 pode encaixar-se nos “quadros” Q1, Q2 e Q3. Por se tratar de uma triagem automática, o sistema especialista, mesmo não tendo certeza do quadro em que o paciente está, deve atribuir o paciente a um dos quadros. Então, como lidar com a incerteza? Qual o mecanismo que o sistema especialista deve adotar para escolher o mais promissor?

## 6.2 Lidando com a incerteza: Probabilidades

Uma forma de lidar com a incerteza é com o uso das *probabilidades*. Considere, por exemplo, um sistema especialista que auxilie um dentista em identificar problemas dentários. Um sintoma *dor\_de\_dente* pode indicar *cáries*.

*dor\_de\_dente* => *cárie*

Mas nem todos os pacientes estão com *dor\_de\_dente* por causa da *cárie*, pois pode ser por outras causas:

*dor\_de\_dente* => *cárie* ou *periodontite* ou *abscesso*

Portanto, quanto menos características a serem observadas temos, menor a certeza que temos em uma previsão, ou classificação. No entanto, no exemplo anterior, suponhamos que 80% dos pacientes avaliados tinham o sintoma *dor\_de\_dente* devido a alguma *cárie*, então, lidamos com a incerteza de uma previsão baseada apenas nesse atributo com a afirmação: “O paciente com *dor\_de\_dente* tem *cárie*, com 80% de chances”.

É óbvio que em problemas reais não baseamos uma afirmação em apenas uma característica, dessa forma, lançamos mão da **teoria da probabilidade** para atribuir um grau de confiança a cada afirmação feita. Esse grau de confiança, geralmente, está entre os valores 0 e 1.

### 6.3 O teorema de Bayes

Em teoria da probabilidade, o teorema de Bayes (ou lei, ou regra de Bayes) descreve a probabilidade de um evento baseado na observação e quantização de conhecimentos anteriores relacionados ao evento. Em outras palavras, o teorema de Bayes computa a probabilidade (ou grau de confiança) de uma afirmação dado um conjunto de observações.

O teorema de Bayes define o seguinte:

$P(H|E)$  - a probabilidade de que a hipótese, ou proposição  $H$ , seja verdadeira dada a evidência  $E$ .

$P(E|H)$  - a probabilidade de que a evidência  $E$  será observada se a hipótese, ou proposição  $H$ , for verdadeira.

$P(H)$  - a probabilidade “a priori” de que a hipótese, ou proposição  $H$ , é verdadeira na ausência de qualquer evidência específica.

A probabilidade  $P(A|B)$  é dada por (lei da probabilidade total):

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Considere o seguinte exemplo: em um dado bairro, no caso do disparo de um alarme, deseja-se quantificar a chance de uma tentativa de roubo a uma casa. Para isso, tem-se as seguintes observações:

- quando há uma tentativa de roubo, em 95% dos casos, o alarme dispara;
- em apenas 1% dos casos, o alarme dispara por outros motivos;
- no bairro em questão, há uma chance de 1 em 10.000 (0.0001) de uma casa ser assaltada em determinado dia.

Podemos formular, então:

$P(\text{alarme}|\text{roubo}) = 0.95$ , que é a probabilidade de um alarme disparar na tentativa de um roubo.

$P(\text{roubo}) = 0.0001$ , probabilidade de um roubo.

$P(\text{alarme}|\text{não_roubo}) = 0.01$ , probabilidade de um alarme por outros motivos.

$$\text{Então } P(\text{roubo}|\text{alarme}) = \frac{P(\text{alarme}|\text{roubo}) * P(\text{roubo})}{P(\text{alarme}|\text{não_roubo})} = \frac{0.95 * 0.0001}{0.01} = 0.0095$$

Dessa forma, podemos dizer que as chances de uma tentativa de roubo, quando um alarme dispara, são de 0,95%, ou seja, menos que 1%. Isto ocorreu, pois as chances de ocorrer uma tentativa de roubo no determinado bairro são muito pequenas (0.0001) em comparação com a probabilidade do alarme soar por qualquer outro motivo que não seja o roubo (0.01).

## 6.3 Hands On! Aplicando o aprendizado automático!

### 6.3.1 Ajustando o ambiente

Para preparar o ambiente dos experimentos, precisamos garantir que o Python esteja instalado. O restante dessa seção será conduzido considerando a versão 2.7 do Python. Também serão fornecidos comandos para os sistemas operacionais Windows e Linux (Ubuntu).

#### PYTHON

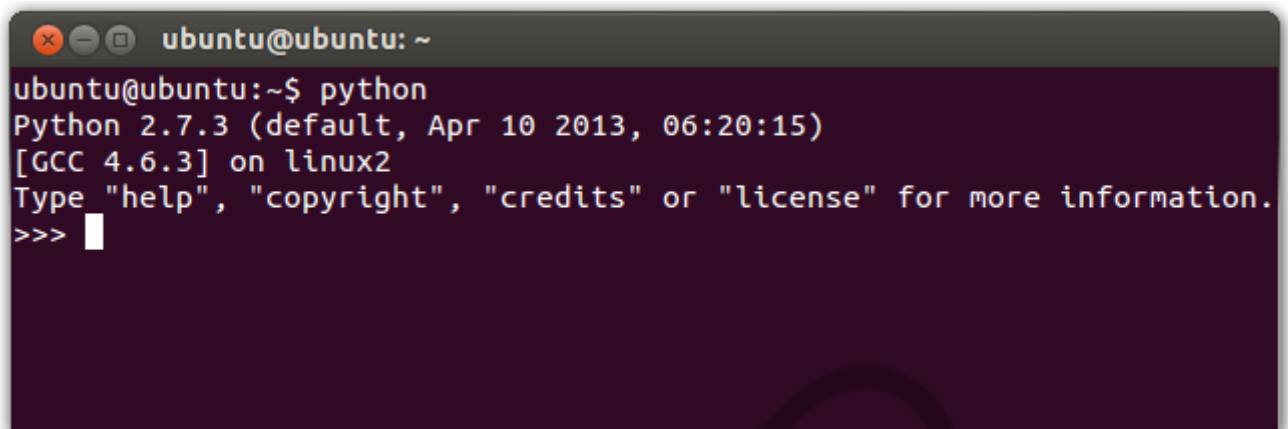
Primeiro, abra um terminal de comandos e verifique se o Python está instalado:

**Windows:** `Ctrl+R` e digite `cmd` (terminal). No terminal que abrirá, digite `python`.

**Linux:** Abra o Terminal e digite `python`.

Caso o Python já esteja instalado, será exibida uma nova linha com os símbolos `>>>`, como exemplificado a seguir.





```
ubuntu@ubuntu: ~$ python
Python 2.7.3 (default, Apr 10 2013, 06:20:15)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 6.1. Exemplo de interpretador interativo do Python instalado em um sistema Linux (distribuição Ubuntu)

Fonte: Elaborado pelo autor (2018)

No caso de ausência do Python no sistema, proceda à instalação:

**Windows:** vá para <https://www.python.org/downloads/windows/> e faça o download de um instalador, por exemplo, “Windows x86 MSI installer”.

**Linux:** No terminal, digite: `sudo apt-get install python`

### GERENCIADOR DE PACOTES PYTHON (PIP)

O pip é um utilitário indispensável para instalar alguns pacotes do Python, como o jupyter e scikit-learn. Então, vamos verificar se o pip já está no sistema:

**Windows:** Ctrl+R, digite `cmd` (terminal). Digite: `python -m ensurepip --upgrade`

**Linux:** Abra o Terminal e digite: `sudo apt-get install python-pip`

### JUPYTER

**Windows e Linux:** abra o Terminal e digite: `python -m pip install jupyter`

### SCIKIT-LEARN

O Scikit-learn é um conjunto de ferramentas para as mais diversas abordagens de aprendizado automático, tais como classificação, agrupamento, seleção de

modelos, redução de dimensionalidade e pré-processamento dos dados. Sua instalação é simples, basta:

**Windows e Linux:** abra o Terminal e digite: `python -m pip install -U scikit-learn`

## OUTROS MÓDULOS

Como pré-requisitos para os experimentos, serão necessários os seguintes módulos Python: “pandas”, “numpy”, “scipy”, “matplotlib”. Para instalar, basta executar o comando abaixo:

Windows e Linux: abra o Terminal e digite: `python -m pip install pandas numpy scipy matplotlib`

Terminada a instalação das ferramentas anteriores, estamos prontos para iniciar nosso experimento de *machine learning* com dados da área médica. Mas ainda antes, vamos falar um pouco a respeito dos dados que serão utilizados.

### 6.3.2 Dados do experimento

Há diversos dados (*datasets*) disponíveis para experimentos de aprendizado automático disponíveis para as mais diversas áreas. No caso desse primeiro experimento, vamos considerar dois *datasets*, um de câncer de mama (*Breast Cancer Wisconsin (Diagnostic)* *Database:* [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))) e outro de diabetes (*Diabetes* *Data:* <https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>)

O *dataset* de câncer de mama foi gerado a partir de imagens digitalizadas de amostras retiradas de mamas. Essas imagens descrevem características dos núcleos celulares presentes nas imagens. Foram extraídos 30 atributos numéricos, tais como o raio e perímetro dos núcleos celulares. São 569 exemplos de amostras. Desses, 212 indicam tumores malignos e 357, benignos. Esse é um típico problema de classificação binária.

O *dataset* de diabetes consiste em dados recolhidos de 442 pacientes com diabetes. Portanto, não serve para detecção de diabetes, mas para verificar a

progressão da doença um ano após sua observação. Foram analisadas 10 características, como idade, sexo, índice de massa corporal, etc. A classe desse problema é contínua, isto é, indica a progressão da doença. Esse tipo de classe é indicado para a aplicação de regressão linear.

### 6.3.3 Executando e interpretando o experimento

Podemos executar os experimentos a seguir, utilizando qualquer editor de texto ou IDE específica para o Python, como PyCharm. No entanto, vamos aprender a utilizar a ferramenta Jupyter Notebook. Essa ferramenta é executada em algum browser web, como o Google Chrome ou Mozilla Firefox.

O Jupyter Notebook permite executar partes do código Python isoladamente, considerando todo o contexto do código que já foi executado, reexecutar alguma parte sem ter de executar o resto do código já executado. Além disso, esse ambiente de edição tem muitas *features* gráficas, com o uso do Matplotlib, por exemplo, permitindo visualizar gráficos e imagens logo abaixo do código Python que gerou a informação. No caso de experimentos com IA, esse ambiente é muito útil e produtivo.

Para iniciar o Jupyter, basta abrir o terminal e digitar: `jupyter notebook`. Logo após, o browser padrão abrirá a seguinte tela da imagem, que será exibida abaixo.



Figura 6.2. Tela inicial do Jupyter. Por exemplo, é listado um notebook criado, com nome "Experiment.ipynb"

Fonte: Desenvolvido pelo autor (2018)

Para iniciar um código Python, basta clicar em “New->Python 2”, “”, como ilustrado a seguir.



Figura 6.3. Iniciando um novo código Python  
Fonte: Elaborado pelo autor (2018)

Como ilustrado na Figura 6.4, você pode iniciar a criação de seu código Python e ir executando à medida que quiser ver o resultado até então. No exemplo, no primeiro bloco, um texto foi atribuído a uma variável. No segundo bloco, o comando `print` foi utilizado para exibir o valor da variável. Veja que cada bloco foi executado independentemente. No caso de códigos mais complexos e longos, executar bloco a bloco pode ser uma “mão na roda” para verificar os resultados intermediários ou *debugar* o código.

Como não é o objetivo neste capítulo apresentar esse ambiente de edição e execução de código Python, vamos ao que interessa: experimentos de *machine learning*!

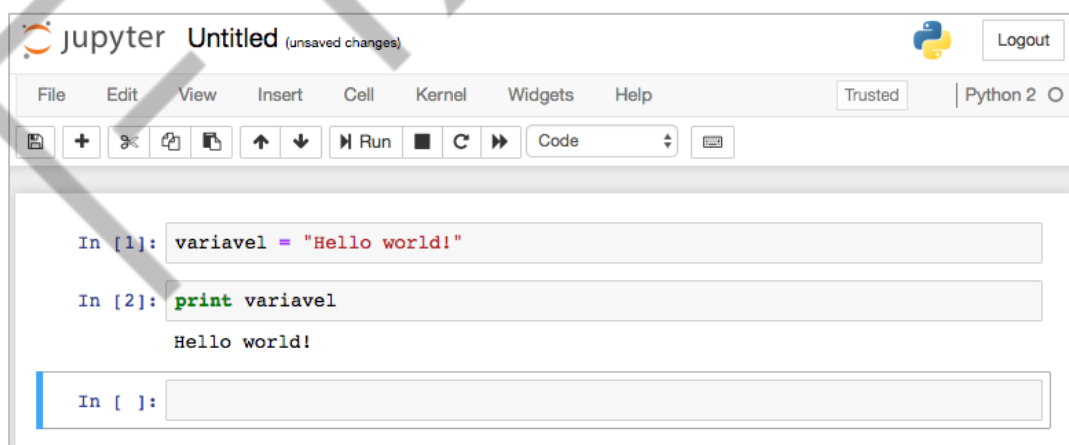


Figura 6.4. Exemplo de edição de código e execução bloco a bloco  
Fonte: Elaborado pelo autor (2018)

O roteiro do experimento é o seguinte:

1. Carregar os módulos Python a serem utilizados.
2. Carregar os *datasets* a serem utilizados para o aprendizado automático.
3. Treinar modelos de aprendizado automático, variando algum parâmetro e fazer sua avaliação.

Que fique claro uma coisa, você terá que explorar mais modelos de aprendizado automático, assim como seus parâmetros! O Scikit-Learn, dentre tantos outros módulos Python, contém inúmeros modelos de classificação, regressão, agrupamento, etc. que você deve conhecer e explorar. Tal conhecimento fará com que você escolha o melhor modelo, com os melhores parâmetros para o problema que enfrentar.

### Carregamento dos módulos

O Código-Fonte 6.1 “Importação dos módulos Python para os experimentos com aprendizado automático” mostra os módulos necessários para os experimentos a seguir. O módulo **sklearn** contém os *datasets* de câncer de mama (carregados pela função *load\_breast\_cancer*) e diabetes (*load\_diabetes*). Contém também os algoritmos de aprendizado de máquina, dentre eles o *svm* (support-vector machines), *DecisionTreeClassifier* (árvores de decisão) e *LinearRegression* (regressão linear). Além dos dados e algoritmos de treinamento, contém um importante módulo, o *train\_test\_split*, que faz a separação do *dataset* em dados de treinamento e teste. Importante lembrar que o conjunto de teste é apenas utilizado para avaliar o modelo gerado por um algoritmo de treinamento e não deve ser utilizado durante o treinamento.

```
from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_diabetes
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

Código-Fonte 6.1. Importação dos módulos Python para os experimentos com aprendizado automático

Fonte: Elaborado pelo autor (2018)

O módulo **matplotlib** faz a exibição de gráficos diretamente no jupyter notebook. Um gráfico agiliza a análise dos dados. Como será visto à frente, cada algoritmo de aprendizado tem parâmetros que devem ser ajustados para que o modelo de aprendizado gerado obtenha a melhor avaliação nos dados de teste.

O matplotlib traz diversas possibilidades de gráficos, o que ajudará na escolha do melhor parâmetro para os algoritmos. O trecho de código `%matplotlib inline` é necessário para que os gráficos gerados sejam exibidos diretamente no notebook e não salvos em um arquivo de imagem.

### Carregamento dos datasets

O Código-Fonte 6.2 e Código-Fonte 6.3 exibem, respectivamente, o carregamento dos *datasets* de câncer de mama e diabetes. Para cada *dataset*, são exibidos os nomes dos atributos (método `.features_names`). Para o dataset de câncer de mama, as classes são discretas (maligno ou benigno). Já para o de diabetes, a classe é um valor contínuo (que varia de 25.0 a 346.0).

```
dataset_cancer = load_breast_cancer()
print dataset_cancer.feature_names
print dataset_cancer.target_names
```

Código-Fonte 6.2. Carregamento do *dataset* de câncer de mama e exibição de seus atributos e classes (discreta)

Fonte: Elaborado pelo autor (2018)

```
dataset_diabetes = load_diabetes()
print dataset_diabetes.feature_names
print dataset_diabetes.target
```

Código-Fonte 6.3. Carregamento do *dataset* de diabetes e exibição de seus atributos e classes (contínua)

Fonte: Elaborado pelo autor (2018)

Para obter a visualização dos *prints* dos códigos fonte anteriores, no *jupyter* notebook, basta clicar no botão “Run”, quando o foco estiver no respectivo bloco de

código. A Figura 6.4 Exemplo de edição de código e execução bloco a bloco” mostra a saída produzida para o Código-Fonte 6.2 e Código-Fonte 6.3. O *dataset* de câncer de mama tem uma lista de 30 atributos (*features\_names*) e duas classes (*target\_names*). Já o *dataset* de diabetes tem 10 atributos (*features\_names*) e, para cada instância do *dataset*, um valor para a classe (*target*).

```
In [42]: dataset_cancer = load_breast_cancer()
print dataset_cancer.feature_names
print dataset_cancer.target_names

['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']

In [41]: dataset_diabetes = load_diabetes()
print dataset_diabetes.feature_names
print dataset_diabetes.target

['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
[151. 75. 141. 206. 135. 97. 138. 63. 110. 310. 101. 69. 179. 185.
 118. 171. 166. 144. 97. 168. 68. 49. 68. 245. 184. 202. 137. 85.
 131. 283. 129. 59. 341. 87. 65. 102. 265. 276. 252. 90. 100. 55.
 61. 92. 259. 53. 190. 142. 75. 142. 155. 225. 59. 104. 182. 128.
 52. 37. 170. 170. 61. 144. 52. 128. 71. 163. 150. 97. 160. 178.
 48. 270. 202. 111. 85. 42. 170. 200. 252. 113. 143. 51. 52. 210.
 65. 141. 55. 134. 42. 111. 98. 164. 48. 96. 90. 162. 150. 279.
 92. 83. 128. 102. 302. 198. 95. 53. 134. 144. 232. 81. 104. 59.
 246. 297. 258. 229. 275. 281. 179. 200. 200. 173. 180. 84. 121. 161.
 99. 109. 115. 268. 274. 158. 107. 83. 103. 272. 85. 280. 336. 281.
 118. 317. 235. 60. 174. 259. 178. 128. 96. 126. 288. 88. 292. 71.
 197. 186. 25. 84. 96. 195. 53. 217. 172. 131. 214. 59. 70. 220.
 268. 152. 47. 74. 295. 101. 151. 127. 237. 225. 81. 151. 107. 64.
 138. 185. 265. 101. 137. 143. 141. 79. 292. 178. 91. 116. 86. 122.
 72. 129. 142. 90. 158. 39. 196. 222. 277. 99. 196. 202. 155. 77.
 191. 70. 73. 49. 65. 263. 248. 296. 214. 185. 78. 93. 252. 150.
 77. 208. 77. 108. 160. 53. 220. 154. 259. 90. 246. 124. 67. 72.
 257. 262. 275. 177. 71. 42. 187. 175. 78. 51. 258. 215. 203. 243]
```

Figura 6.5. Exemplo de execução de dois blocos de códigos e suas respectivas saídas  
Fonte: Desenvolvido pelo autor (2018)

## Treinamento e avaliação dos modelos

Antes de realizar o treinamento, os *datasets* devem ser separados em treino e teste. Considere *X\_train\_can* como os atributos de treinamento e *X\_test\_can*, os de teste do *dataset* de câncer de mama. *Y\_train\_can* e *y\_test\_can*, respectivamente, as classes do *dataset* de câncer de mama. Analogamente, é feita a divisão para o *dataset* de diabetes.

```
X_train_can, X_test_can, y_train_can, y_test_can = train_test_split(
dataset_cancer.data, dataset_cancer.target, stratify=dataset_cancer.target,
random_state=42)
```

```
X_train_dia, X_test_dia, y_train_dia, y_test_dia = train_test_split(
dataset_diabetes.data, dataset_diabetes.target, stratify=dataset_diabetes.target,
random_state=42)
```

Código-Fonte 6.4. Separação dos datasets nos conjuntos de treino e test  
Fonte: Elaborado pelo autor (2018)



A seguir, serão exibidos o treinamento e avaliação de dois algoritmos de treinamento, a saber, o SVM e Árvores de Decisão. Vale salientar novamente que diversos outros algoritmos estão disponíveis e fica como exercício sua exploração nos *datasets* abordados e em outros *datasets*. O treinamento de modelos com SVM é apresentado no Código-Fonte 6.5. São criadas duas listas (*training\_accuracy* e *test\_accuracy*) que armazenarão as avaliações do modelo gerado tanto no conjunto de treinamento quanto no de teste.

Veja que os resultados obtidos no conjunto de treinamento devem sempre ser altos, próximos de 1.0 (100%), pois foi nesses dados que o algoritmo se baseou para gerar o modelo. Já os resultados obtidos no conjunto de teste são os valores que indicam o quão acurado está o modelo gerado. São utilizadas essas listas, pois a cada iteração do comando *for* é testado um parâmetro diferente.

O SVM trabalha com o uso de um *kernel* que faz a transformação dimensional dos atributos (pesquise um pouco mais sobre o SVM). Há diversos possíveis *kernels*, então são testados três: *linear*, *rbf* (*radial basis function*) e *sigmoid*.

```
training_accuracy = []
test_accuracy = []

kernels = ['linear', 'rbf', 'sigmoid']
for kernel in kernels:
    svm_model = svm.SVC(kernel=kernel)

    svm_model.fit(X_train_can, y_train_can)
    training_accuracy.append(svm_model.score(X_train_can, y_train_can))
    test_accuracy.append(svm_model.score(X_test_can, y_test_can))

plt.plot(kernels, training_accuracy, label='Acuracia no conj. treino')
plt.plot(kernels, test_accuracy, label='Acuracia no conj. teste')
plt.ylabel('Accuracy')
plt.xlabel('Kernels')
plt.legend()
```

Código-Fonte 6.5. Treinamento e avaliação do SVM  
Fonte: Elaborado pelo autor (2018)

O módulo *matplotlib* (*plt*) é utilizado para criar um gráfico de acurácia por *kernel* utilizado. O respectivo gráfico é visto na Figura 6.6. Essa é uma vantagem do uso do jupyter notebook: é possível executar um bloco de código e já visualizar seu resultado. A curva azul indica a acurácia do modelo com os três *kernels*, já a linha laranja, indica a acurácia no conjunto de teste. Para escolher o melhor *kernel*, utilizamos a linha laranja e, como pode ser concluído, o *kernel linear* foi o que melhor



se comportou nesse problema, com uma acurácia de aproximadamente 0.95 (95%). Já os outros *kernels* ficaram com acurácias abaixo de 0.65.

```
In [48]: training_accuracy = []
test_accuracy = []

kernels = ['linear', 'rbf', 'sigmoid']
for kernel in kernels:
    svm_classifier = svm.SVC(kernel=kernel)

    svm_classifier.fit(X_train_cancer, y_train_cancer)
    training_accuracy.append(svm_classifier.score(X_train_cancer, y_train_cancer))
    test_accuracy.append(svm_classifier.score(X_test_cancer, y_test_cancer))

plt.plot(kernels, training_accuracy, label='Acuracia no conj. treino')
plt.plot(kernels, test_accuracy, label='Acuracia no conj. teste')
plt.ylabel('Accuracy')
plt.xlabel('Kernels')
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x110ad0950>

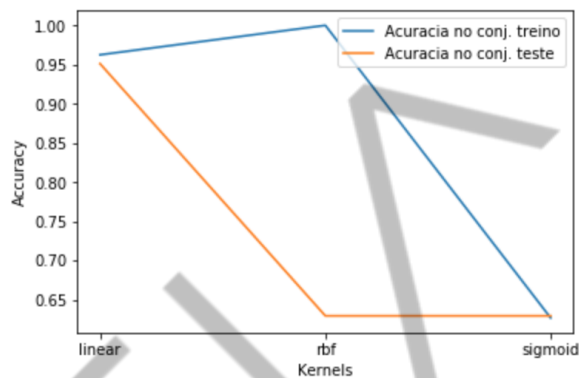


Figura 6.6. Gráfico gerado com os dados da avaliação do SVM com três *kernels*  
Fonte: Elaborado pelo autor (2018)

O uso de árvores de decisão é feito no código abaixo. Nesse caso também foram utilizadas duas listas para armazenar os resultados das avaliações. O parâmetro que foi escolhido para exemplificar o melhor ajuste foi a profundidade máxima (*max\_depth*) da árvore de decisão.

Fica, claro, pela “Figura 6.7”, que as árvores geradas com profundidade máxima 4 e 5 obtiveram os melhores resultados na avaliação.

```
training_accuracy = []
test_accuracy = []

prof_max = range(1,10)

for md in prof_max:
    tree = DecisionTreeClassifier(max_depth=md, random_state=0)
    tree.fit(X_train_can, y_train_can)
    training_accuracy.append(tree.score(X_train_can, y_train_can))
    test_accuracy.append(tree.score(X_test_can, y_test_can))
```

```
plt.plot(prof_max, training_accuracy, label='Acuracia no conj. treino')
plt.plot(prof_max, test_accuracy, label='Acuracia no conj. teste')
plt.ylabel('Acuracia')
plt.xlabel('Profundidade Maxima')
plt.legend()
```

Código-Fonte 6.6. Treinamento e avaliação de árvores de decisão  
Fonte: Elaborado pelo autor (2018)

```
In [50]: training_accuracy = []
test_accuracy = []

profundidade_maxima = range(1,10)

for md in profundidade_maxima:
    tree = DecisionTreeClassifier(max_depth=md, random_state=0)
    tree.fit(X_train_cancer, y_train_cancer)
    training_accuracy.append(tree.score(X_train_cancer, y_train_cancer))
    test_accuracy.append(tree.score(X_test_cancer, y_test_cancer))

plt.plot(profundidade_maxima, training_accuracy, label='Acuracia no conj. treino')
plt.plot(profundidade_maxima, test_accuracy, label='Acuracia no conj. teste')
plt.ylabel('Acuracia')
plt.xlabel('Profundidade Maxima')
plt.legend()
```

Out[50]: <matplotlib.legend.Legend at 0x111671f90>

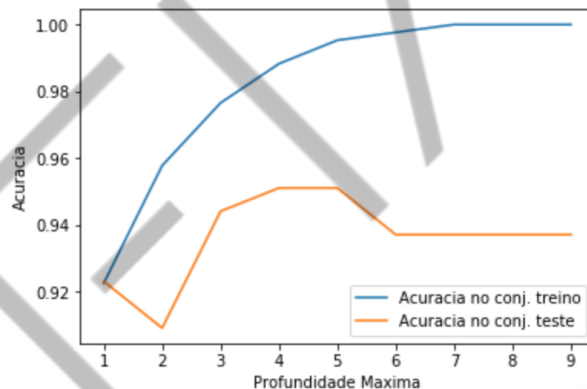


Figura 6.7. Gráfico gerado com os dados da avaliação das árvores de decisão com variações da profundidade máxima

Fonte: Elaborado pelo autor (2018)

Por fim, o Código-Fonte 6.7” apresenta o treinamento (método *.fit*) e avaliação (método *.score*) de um modelo de regressão linear, dado que o *dataset* de diabetes tem classes contínuas, indicando a progressão da doença em diabéticos, um ano após sua avaliação.

```
training_accuracy = []
test_accuracy = []
```

```

for interception in [True, False]:
    regr = LinearRegression(fit_intercept=interception)
    regr.fit(X_train_dia, y_train_dia)
    training_accuracy.append(regr.score(X_train_dia, y_train_dia))
    test_accuracy.append(regr.score(X_test_dia, y_test_dia))

plt.plot(["Interc", "No Interc"], training_accuracy, label='Acuracia no conj. treino')
plt.plot(["Interc", "No Interc"], test_accuracy, label='Acuracia no conj. teste')
plt.ylabel('Acuracia')
plt.xlabel('Fit Intercept')
plt.legend()

```

Código-Fonte 6.7. Treinamento e avaliação do algoritmo de regressão linear com o *dataset* de diabetes

Fonte: Elaborado pelo autor (2018)

Foi variado apenas um parâmetro, o *fit\_intercept*, que assume valores booleanos (*True* ou *False*). Um gráfico foi gerado para escolher o melhor parâmetro (Figura 6.8 Exemplo de gráfico de avaliação do modelo de regressão linear"). Pelo gráfico, o melhor parâmetro é *fit\_intercept* igual a *True*.

```

In [61]: training_accuracy = []
         test_accuracy = []

         for interception in [True, False]:
             regr = LinearRegression(fit_intercept=interception)
             regr.fit(X_train_dia, y_train_dia)
             training_accuracy.append(regr.score(X_train_dia, y_train_dia))
             test_accuracy.append(regr.score(X_test_dia, y_test_dia))

         plt.plot(["Interc", "No Interc"], training_accuracy, label='Acuracia no conj. treino')
         plt.plot(["Interc", "No Interc"], test_accuracy, label='Acuracia no conj. teste')
         plt.ylabel('Acuracia')
         plt.xlabel('Fit Intercept')
         plt.legend()

Out[61]: <matplotlib.legend.Legend at 0x111bb3550>

```

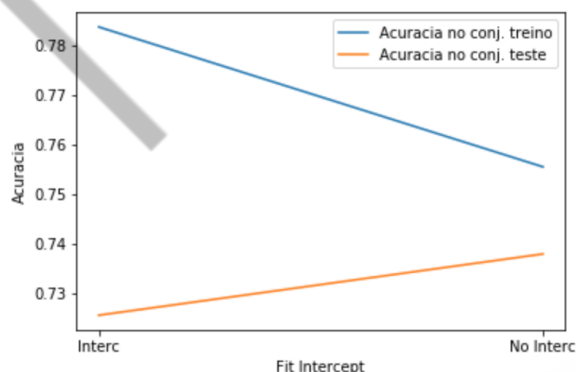


Figura 6.8. Exemplo de gráfico de avaliação do modelo de regressão linear

Fonte: Elaborado pelo autor (2018)

Após o treinamento dos modelos exemplificados acima, eles podem ser utilizados para classificar dados ainda não vistos (ou fazer a regressão linear deles), ou seja, classificando dados num cenário real.

Os dados que serão classificados devem ter os mesmos atributos encontrados no treinamento. Por exemplo, o *dataset* de diabetes tem 10 atributos: ('age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'), então, ao utilizar o modelo gerado para prever a progressão da diabetes, os dados de entrada devem ser valores para esses 10 atributos.

Não fique restrito aos *datasets*, algoritmos e parâmetros abordados neste capítulo, tem muito, muito mais a ser explorado! Mãos à obra!

## REFERÊNCIAS

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. Rio de Janeiro: Elsevier, 2013.

EMAP

## GLOSSÁRIO

<b>Acurácia</b>	Em uma avaliação de um modelo gerado por algoritmos de aprendizado de máquina, indica a porcentagem de acertos no conjunto de teste fornecido ao modelo.
<b>Bayes (Thomas Bayes)</b>	Pastor presbiteriano e matemático inglês que formulou o teorema de Bayes.