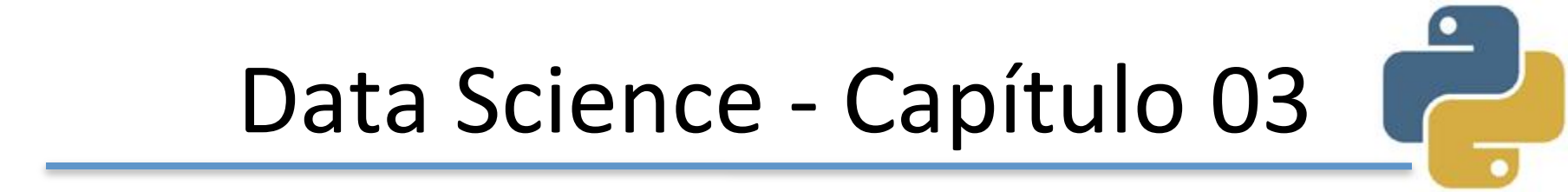
Data Science - Capítulo 03



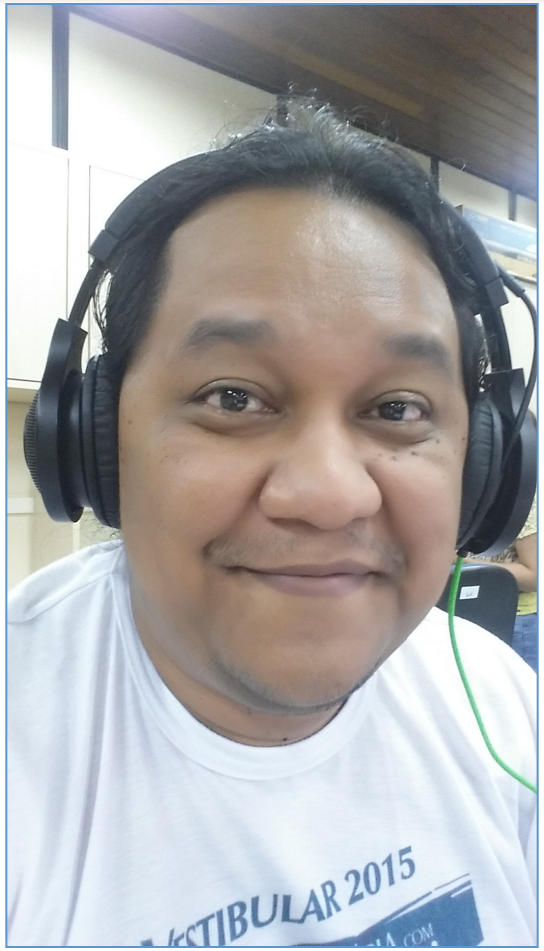
Classificação e Regressão com kNN em scikit-learn

Márcio Palheta, M.Sc.
marcio.palheta@gmail.com

Apresentação

- Programador desde 2000
- Professor de programação desde 2009
- Mestre pelo ICOMP/UFAM – 2013
- Fundador da Buritech – 2014
- Doutorando pelo ICOMP/UFAM
- Pesquisador das áreas: Banco de Dados, Recuperação da Informação, Big Data, Mineração de dados e Aprendizado de Máquina

https://sites.google.com/site/marciopalheta



Regressão Linear

- Em estatística, regressão linear é uma equação para se estimar a condicional (valor esperado) de uma variável y, dados os valores de algumas outras variáveis x
- A regressão, em geral, trata da questão de se estimar um valor condicional não esperado

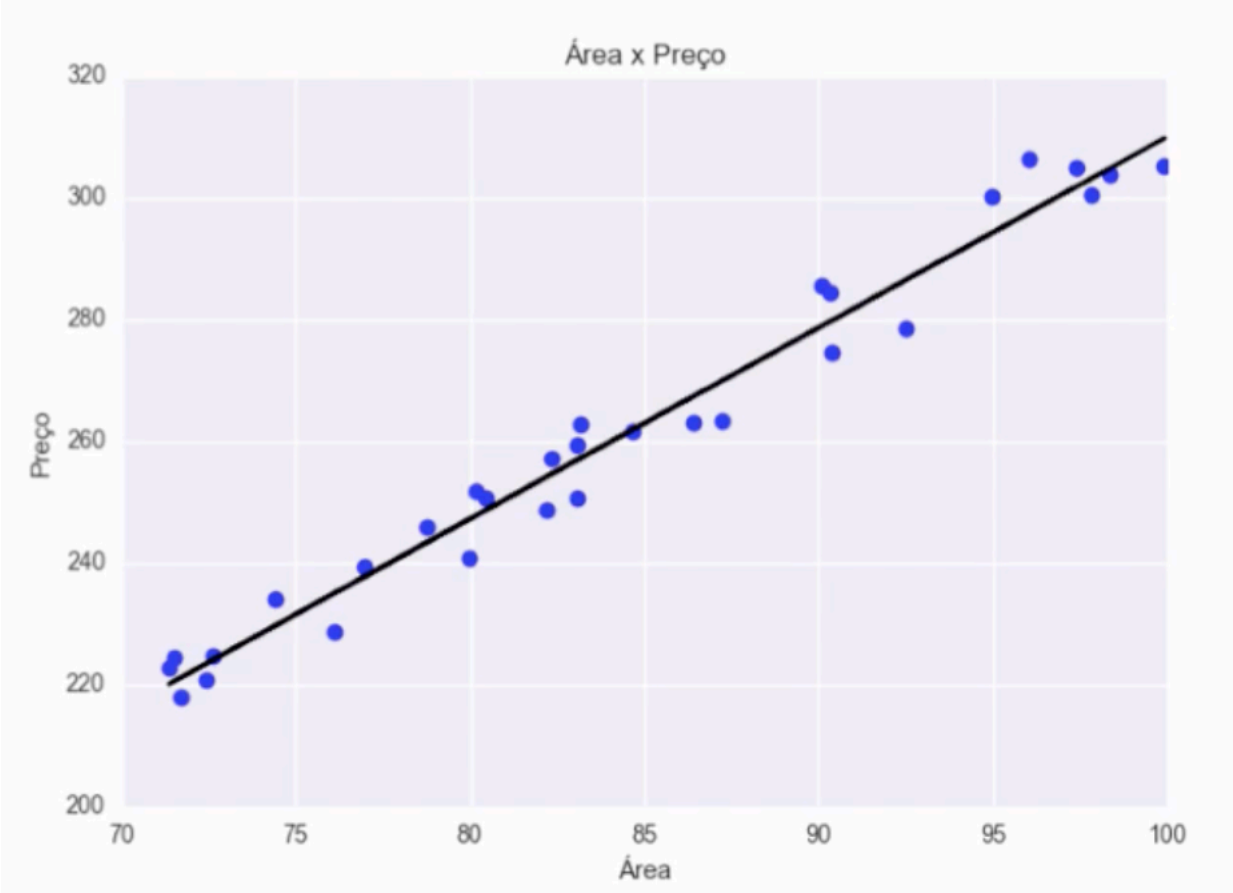
Regressão Linear

- A regressão linear é chamada "linear" porque se considera que a relação da resposta às variáveis é uma função linear de alguns parâmetros.
- Modelos que dependem de forma linear dos seus parâmetros desconhecidos, são mais fáceis de ajustar que os modelos não-lineares aos seus parâmetros

Regressão Linear

- Queremos fazer a predição de um valor real, não discreto.
- A regressão gera uma equação:
- Y = B * X
  - Onde B é um vetor de pesos e
  - X é um vetor de atributos

Regressão Linear



Problemas de Regressão

- Resultados de avaliações de Imóveis, Carros etc.
- Pontuação de créditos
- Variação salarial
- Preço de uma ação da bolsa de valores

Técnicas de regressão

- Vamos estudar algumas técnicas de regressão:
- K-NN (K-nearest neighbors algorithm)
- Redes neurais
- SVM (Suporte Vector Machine)

Paciente de cirurgia CA mama

https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival

UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Haberman's Survival Data Set

Download: Data Folder, Data Set Description

Abstract: Dataset contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer

Data Set Characteristics:	Multivariate	**Number of Instances:**	306	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	3	**Date Donated**	1999-03-04
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	115628

Source:

Donor:

Tjen-Sien Lim (limt '@' stat.wisc.edu)

Data Set Information:

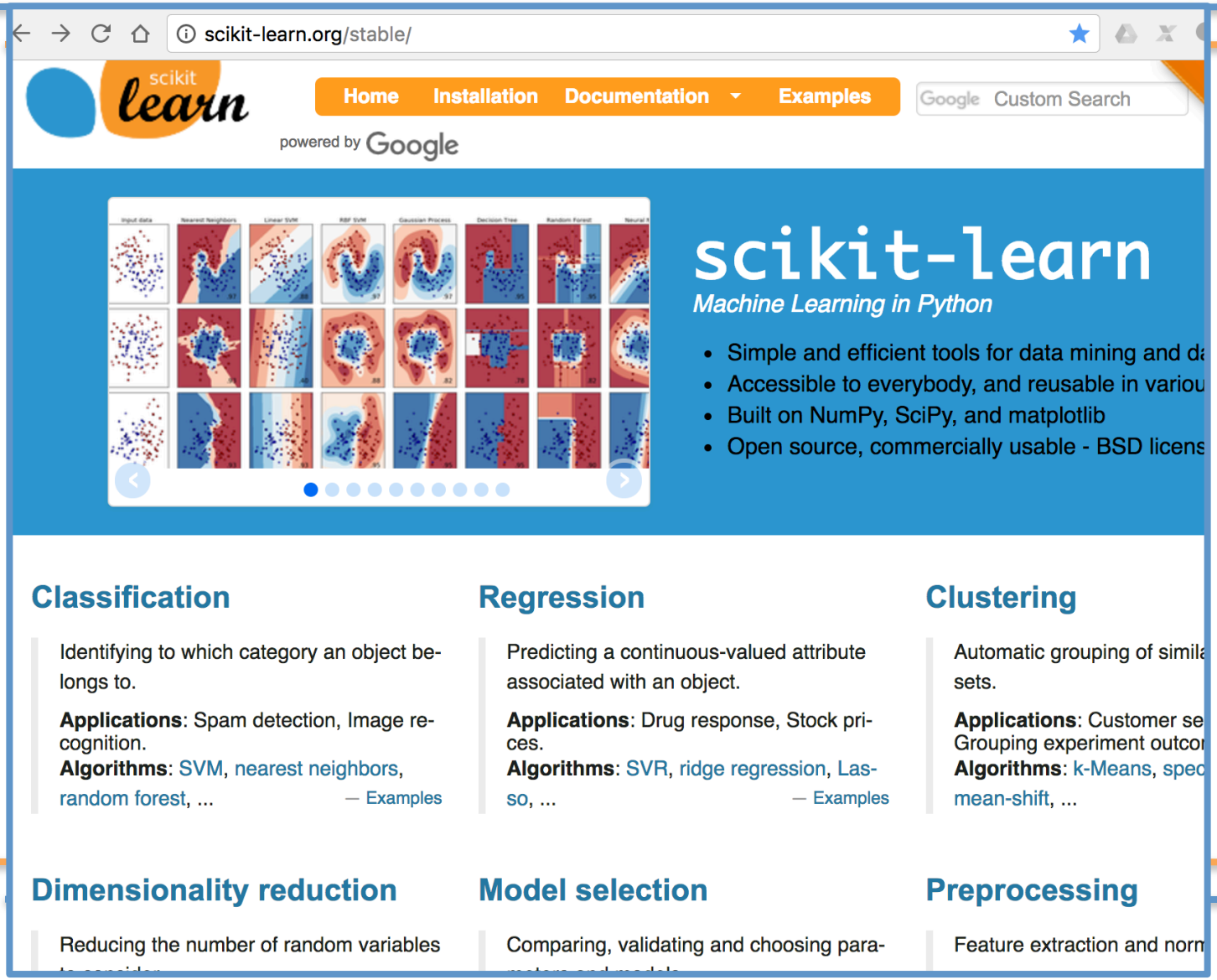
The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

Dados do dataset

- Classes: (1) Sobreviveu 5 anos ou mais, após a cirurgia; (2) Morreu antes dos cinco anos
- 3 atributos numéricos - inteiros:
  - Idade do paciente da operação
  - Ano da operação
  - Número de nódulos detectados

https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival

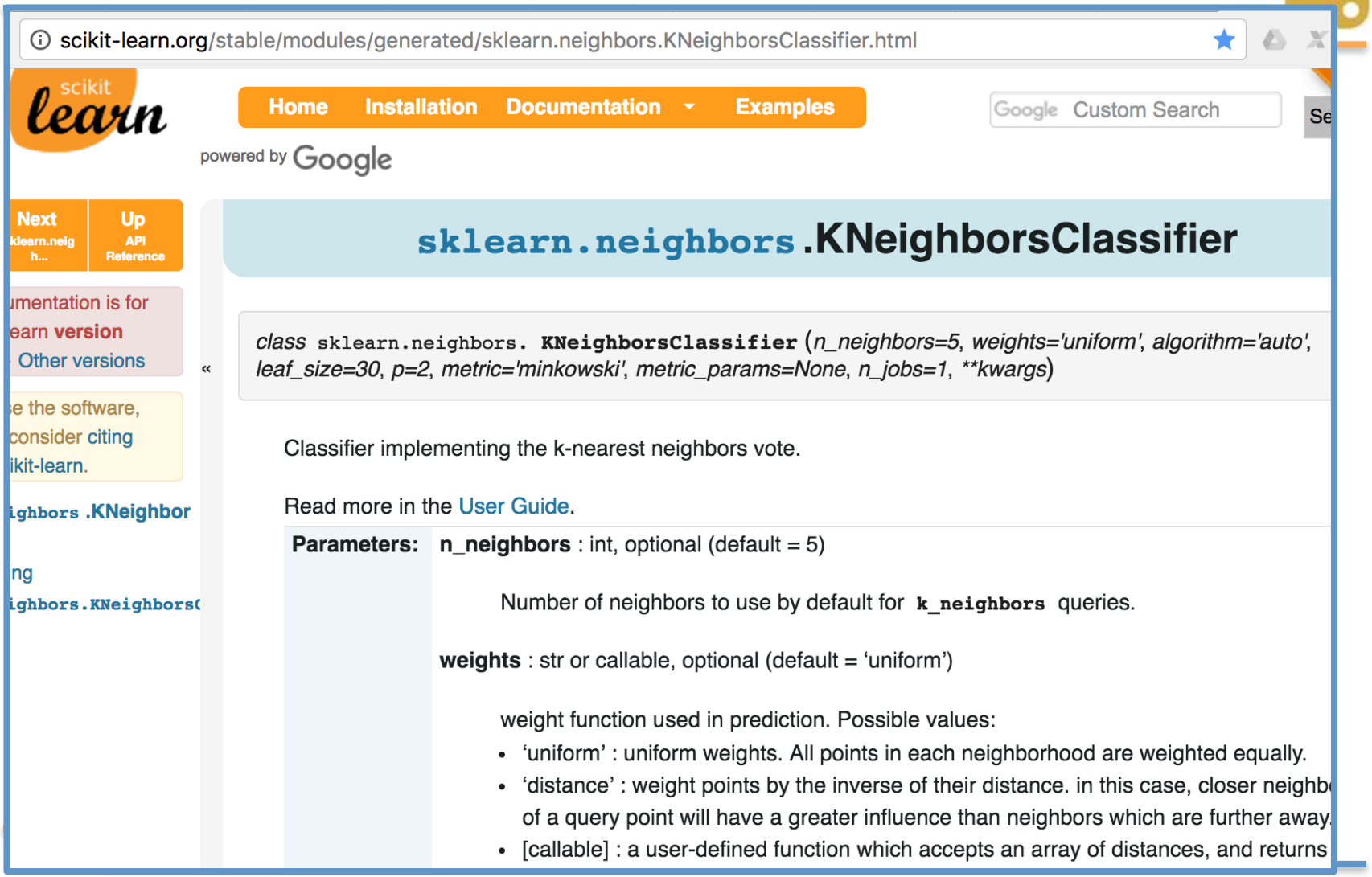
Scikit-learn



Scikit-learn

- Ferramenta simples e eficiente para Mineração e Análise de dados
- Construída em NumPy, SciPy, and matplotlib
- Código aberto, comercialmente utilizável - Licença BSD
- Instalação: $ pip install -U scikit-learn
- Site: http://scikit-learn.org/stable/

kNN com Scikit-learn



http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Começando com classificação

Como primeira implementação, vamos trabalhar usando classificação com kNN e scikit-learn

Exercício 1: carga do dataset

Implementação do kNN com sklearn

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# Entradas == Atributos; Saídas == Classes
entradas, saidas = [], []
# Carga da base de dados, sem o atributo ANO
with open('haberman.data', 'r') as f:
    for linha in f.readlines():
        atrib = linha.replace('\n', '').split(',')
        entradas.append([int(atrib[0]), int(atrib[2])])
        saidas.append(int(atrib[3]))
print 'entradas:', entradas[:5]
print 'saídas:  ',saidas[:15]
```

```
entradas: [[30, 1], [30, 3], [30, 0], [31, 2], [31, 4]]
saídas:   [1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1]
```

Exercício 2: Treino e Teste

```
# porcentagem dos dados para treinamento
p = 0.6
# limite para treinamento
limite = int(p * len(entradas))

# Criação do objeto classificador
knn = KNeighborsClassifier(n_neighbors=15)

# Treinamento do modelo, usando apenas uma parte da base
knn.fit(entradas[:limite], saidas[:limite])

# Teste do modelo, usado a segunda parte das entradas
labels = knn.predict(entradas[limite:])

# Variáveis de controle
acertos, indice_label = 0, 0

# Análise do resultado.
for i in range(limite, len(entradas)):
    # Verificando se a classe prevista é igual à real
    if labels[indice_label] == saidas[i]:
        acertos += 1
    indice_label += 1
```

Exercício 3: Acurácia do modelo

```
print('Total de treinamento: %d' % limite)
print('Total de testes: %d' % (len(entradas) - limite))
print('Total de acertos: %d' % acertos)
print('Porcentagem de acertos: %.2f' %
      (100 * acertos / (len(entradas) - limite)))
```

```
Total de treinamento: 183
Total de testes: 123
Total de acertos: 92
Porcentagem de acertos: 74.00
```

Trabalhando com numpy

Em nossa nova implementação, vamos trabalhar usando classificação com kNN, numpy e scikit-learn

Psicologia cognitiva

https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival

About Citation Policy Donate a Data Set Contact

UCI

Search

Repository Web

Google™

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

View ALL Data Sets

Haberman's Survival Data Set

Download: Data Folder, Data Set Description

Abstract: Dataset contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer

Data Set Characteristics:	Multivariate	**Number of Instances:**	306	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	3	**Date Donated**	1999-03-04
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	115628

Source:

Donor:

Tjen-Sien Lim (limt '@' stat.wisc.edu)

Data Set Information:

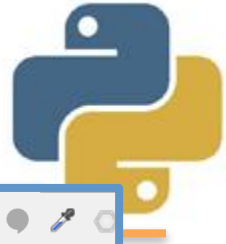
The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

Aspectos de desenv. cognitivo

- Classes: (L) Equilíbrio para esquerda; (B) Balanceado; (R) Equilíbrio para direita;
- 4 atributos numéricos - inteiros:
  - Peso à esquerda: 1, 2, 3, 4, 5
  - Distância à esquerda: 1, 2, 3, 4, 5
  - Peso à esquerda: 1, 2, 3, 4, 5
  - Distância à esquerda: 1, 2, 3, 4, 5

https://archive.ics.uci.edu/ml/datasets/Balance+Scale

sklearn.model_selection



scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

scikit learn

Home Installation Documentation Examples

powered by Google

Google Custom Search

Fork me on G

sklearn.model_selection.train_test_split

`sklearn.model_selection.` **`train_test_split`** (**arrays*, ***options*) ¶ [source]

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Read more in the User Guide.

Parameters:

***arrays** : sequence of indexables with same length / shape[0]

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : float, int, or None (default is None)

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is automatically set to the complement of the train size. If train size is also None, test size is set to 0.25.

train_size : float, int, or None (default is None)

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Exercício 4: docstrings e import

Implementação do kNN com sklearn e numpy

```
'''
Dataset: https://archive.ics.uci.edu/ml/datasets/Balance+Scale

Attribute Information:

1. Class Name: 3 (L, B, R)
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)
4. Right-Weight: 5 (1, 2, 3, 4, 5)
5. Right-Distance: 5 (1, 2, 3, 4, 5)

São 625 exemplos, 4 atributos e 3 possíveis classes.
Onde tem "L" coloquei 1, onde tem "B" coloquei 2 e onde tem "R" coloquei 3.
'''
import numpy as np
```

Exercício 5: carga de dados

```
# x são as entradas e y são as saídas
x = np.genfromtxt('balancescale.data', delimiter=',', usecols=(1,2,3,4))
y = np.genfromtxt('balancescale.data', delimiter=',', usecols=(0))
print 'len(x)', len(x)
print 'len(y)', len(y)
print(x[:5])
print(y[:15])
```

```
len(x) 625
len(y) 625
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  2.]
 [ 1.  1.  1.  3.]
 [ 1.  1.  1.  4.]
 [ 1.  1.  1.  5.]]
[ 2.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.]
```

Validação cruzada

- Técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados.
- É amplamente empregada em problemas onde o objetivo da modelagem é a predição.
- Procura-se estimar o quão preciso é o modelo, dado um novo conjunto de dados.

Validação cruzada

- Particionamos o conjunto de dados em subconjuntos mutualmente exclusivos
- Treino: Subconjuntos usados para estimar os parâmetros do modelo
- Teste: subconjuntos empregados na validação do modelo.

Método holdout

- Divisão do conjunto de dados em dois subconjuntos mutuamente exclusivos, treino e teste
- Os conjuntos de treino e teste podem ter o mesmo número de dados ou não, mas é comum usarmos 2/3 treino e 1/3 teste
- Indicado para grandes quantidades de dados

Exercício 6: usando holdout

```
from sklearn.model_selection import train_test_split

#Divisão do dataset em Treino e Teste
x_treino, x_teste, y_treino, y_teste =
    train_test_split(x, y, test_size=0.3, random_state=42)
print 'len(x_treino)', len(x_treino)
x_treino
```

```
len(x_treino) 437

array([[ 2.,  4.,  1.,  5.],
       [ 1.,  4.,  3.,  4.],
       [ 5.,  1.,  5.,  3.],
       ...,
       [ 3.,  1.,  5.,  1.],
       [ 4.,  3.,  3.,  1.],
       [ 1.,  5.,  1.,  3.]])
```

Exercício 7: teste do modelo

```
from sklearn.neighbors import KNeighborsClassifier
# Criação do classificador. Distância Euclidiana (p=2)
knn = KNeighborsClassifier(n_neighbors=17, p=2)
# Treino do classificador
knn.fit(x_treino, y_treino)
# Teste do classificador
labels = knn.predict(x_teste)
print(len(labels))
labels
```

```
188

array([ 1.,  1.,  1.,  1.,  1.,  1.,  3.,  1.,  1.,  3.,
        3.,  1.,  1.,  3.,  3.,  3.,  1.,  3.,  3.,  3.,
        3.,  1.,  1.,  1.,  1.,  3.,  3.,  1.,  3.,  1.,
        1.,  1.,  3.,  1.,  1.,  3.,  1.,  1.,  3.,  1.,
        3.,  3.,  1.,  1.,  3.,  1.,  1.,  1.,  1.,  1.,
        1.,  3.,  3.,  1.,  1.,  3.,  3.,  3.,  3.,  3.,
        1.,  3.,  3.,  3.,  3.,  1.,  3.,  1.,  1.,  1.,
```

Exercício 8: avaliação

```
# Número de acertos
np.sum(labels == y_teste)
```

```
163
```

```
# Outra forma de contar os acertos
(labels == y_teste).sum()
```

```
163
```

```
# percentual de acertos
100.0 * (labels == y_teste).sum() / len(x_teste)
```

```
86.70212765957447
```

```
# Avaliação usando a função score
knn.score(x_teste, y_teste)
```

```
0.86702127659574468
```

```
# Avaliação usando a função np.mean
np.mean(labels == y_teste)
```

```
0.86702127659574468
```

Exercício 9: Pandas e Sklearn

Implementação com Pandas e Sklearn

```
import pandas as pd
import numpy as np

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

```
train.head()
```

	shoe size	height	class
0	8.518110	73.029460	seniors
1	10.301527	68.959677	seniors
2	7.386575	73.558042	seniors
3	9.477281	68.195558	seniors
4	10.910389	75.144672	seniors

Exercício 10: teste.head()

```
In [64]: test.head()
```

Out[64]:

	shoe size	height	class
0	12.985509	66.632266	seniors
1	9.957683	70.057887	seniors
2	10.366656	67.945519	seniors
3	10.484072	74.846184	seniors
4	10.273374	66.690172	seniors

Exercício 11: definição das bases

```
from sklearn.neighbors import KNeighborsClassifier

# Definição de colunas
cols = ['shoe size', 'height']
cols2 = ['class']

# Definição das bases de treino e teste
x_train = train.as_matrix(cols)
y_train = train.as_matrix(cols2)
x_test = test.as_matrix(cols)
y_test = test.as_matrix(cols2)
```

Exercício 12: Tete e avaliação

```
# Criação do classificador.
# Visinhos mais próximos com mais importância (weights='distance')
knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
# Treino do classificador.
# Ravel matriz retornada e de entrada têm o mesmo tipo
knn.fit(x_train, y_train.ravel())
# Execução dos testes
output = knn.predict(x_test)
```

```
# Conferindo o resultado de uma amostra:
print(knn.predict([x_test[0]]))
print(y_test[0])
```

```
['seniors']
['seniors']
```

```
# Avaliação do modelo
correct = 0.0
for i in range(len(output)):
    if y_test[i][0] == output[i]:
        correct += 1
correct / len(output)
```

```
0.9933333333333333
```

Exercício 13: load_iris()

kNN, numpy, scikit learn - Iris dataset

```
import numpy as np
from sklearn import neighbors, datasets

iris = datasets.load_iris()
iris.DESCR
```

```
'Iris Plants Database\n====================\n\nNotes\n-----\nData S
Number of Instances: 150 (50 in each of three classes)\n    :Number
predictive attributes and the class\n    :Attribute Information:\n
m\n        - sepal width in cm\n        - petal length in cm\n
      - class:\n                - Iris-Setosa\n                - Ir
      - Iris-Virginica\n    :Summary Statistics:\n\n    ==========
= ====================\n                     Min  Max   Mean    SD
=========== ==== ==== ======= ===== ====================\n    sepa
4   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0
1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1
65  (high!)\n    ============= ==== ==== ======= ===== ==========
 Attribute Values: None\n    :Class Distribution: 33.3% for each of
R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa
88\n\nThis is a copy of UCI ML iris datasets.\nhttp://archive.ics.u
```

Exercício 14: Visualização

```
# Visualizando os dados do dataset
x = iris.data
x[:10]
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1]])
```

```
# Visualizando a distribuição de classes
y = iris.target
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Exercício 15: classificação

```
# Visualização dos nomes das classes
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'],
      dtype='|S10')
```

```
# Criação do classificador
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
# Treino do modelo
knn.fit(x, y)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
# Testando com toda a base
knn.predict(x)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
# Avaliação do modelo
acuracia = knn.score(x, y)
acuracia
```

```
0.96666666666666667
```

kNN para Regressão

- http://archive.ics.uci.edu/ml/datasets/Housing

archive.ics.uci.edu/ml/datasets/Housing

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Housing Data Set

Download: Data Folder, Data Set Description

Abstract: Taken from StatLib library

Data Set Characteristics:	Multivariate	**Number of Instances:**	506	**Area:**	N/A
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	14	**Date Donated**	1993-07-07
Associated Tasks:	Regression	**Missing Values?**	No	**Number of Web Hits:**	331641

Source:

Origin:

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

Exercício 16: knn para regressão

kNN para Regressão com sklearn

```
from sklearn.datasets import load_boston

#dataset de preços de casas de Boston
boston = load_boston()
boston.DESCR
```

```
"Boston House Prices dataset\n===========================\n\nNotes\n------\nData Set Characte
ristics:  \n\n    :Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categor
ical predictive\n    \n    :Median Value (attribute 14) is usually the target\n\n    :Attribu
te Information (in order):\n        - CRIM     per capita crime rate by town\n        - ZN
    proportion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    prop
ortion of non-retail business acres per town\n        - CHAS     Charles River dummy variable
(= 1 if tract bounds river; 0 otherwise)\n        - NOX      nitric oxides concentration (par
ts per 10 million)\n        - RM       average number of rooms per dwelling\n        - AGE
   proportion of owner-occupied units built prior to 1940\n        - DIS      weighted distan
ces to five Boston employment centres\n        - RAD      index of accessibility to radial hi
ghways\n        - TAX      full-value property-tax rate per $10,000\n        - PTRATIO  pupil
-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the proportion of bl
acks by town\n        - LSTAT    % lower status of the population\n        - MEDV     Median
 value of owner-occupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Crea
tor: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttp://ar
chive.ics.uci.edu/ml/datasets/Housing\n\n\nThis dataset was taken from the StatLib library wh
```

Exercício 17: dados da base

```
# Exibindo a quantidade de instâncias e atributos
boston.data.shape
```

```
(506, 13)
```

```
# Nomes de atributos
boston.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],
      dtype='|S7')
```

Exercício 18: Regressão

```
# Agora, importamos o regressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split

K = 9
knn = KNeighborsRegressor(n_neighbors=K)
knn.fit(boston.data, boston.target)
print 'Real:', boston.target[0]
print 'Previsto:', knn.predict([boston.data[0]])
```

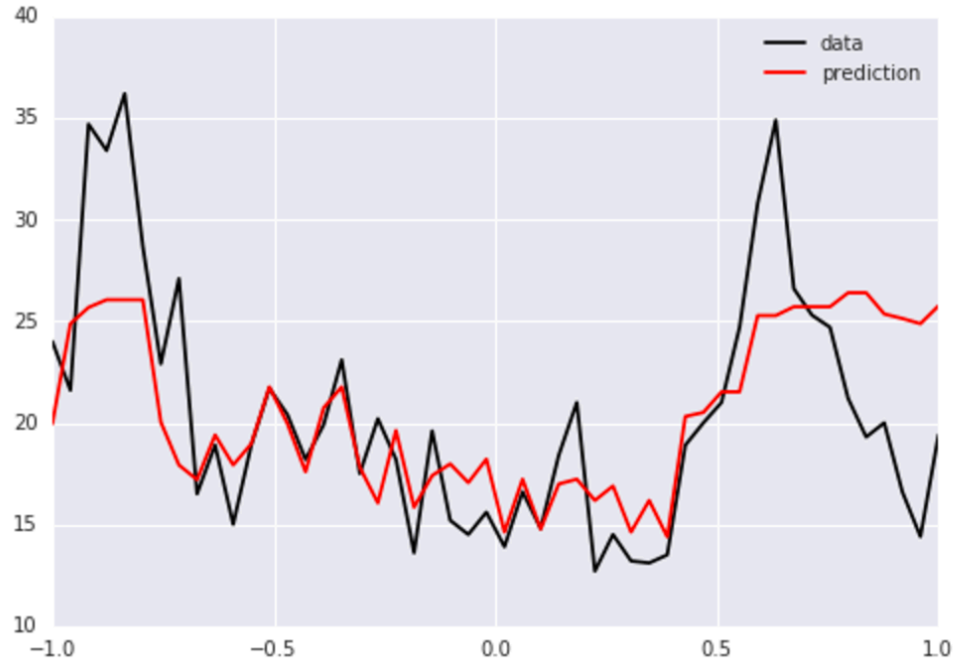
```
Real: 24.0
Previsto: [ 22.33333333]
```

```
# Previsão para a amostra 12
y_ = knn.fit(boston.data, boston.target).predict([boston.data[12]])
print'Real:', boston.target[12]
print'Previsto:', y_
```

```
Real: 21.7
Previsto: [ 23.11111111]
```

Exercício 19: Plot

```
#plot dos resultados
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn
knn = KNeighborsRegressor(n_neighbors=K)
x, y = boston.data[:50], boston.target[:50]
y_ = knn.fit(x, y).predict(x)
plt.plot(np.linspace(-1, 1, 50), y, label='data', color='black')
plt.plot(np.linspace(-1, 1, 50), y_, label='prediction', color='red')
plt.legend()
plt.show()
```



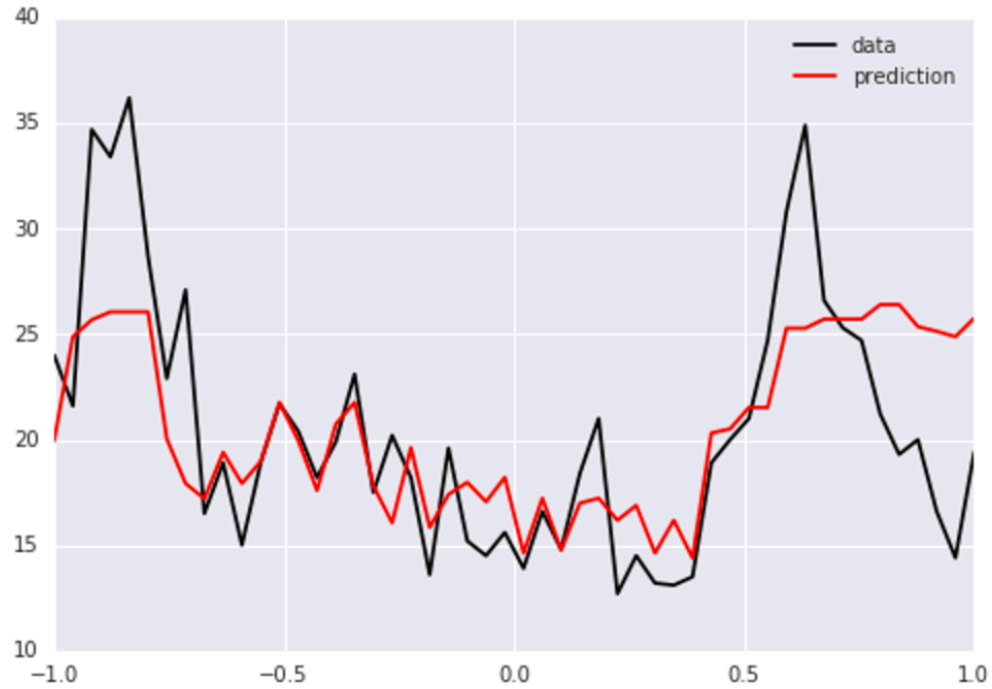
Exercício 20: Métrica

Erro Quadrado Médio

```
In [7]: from sklearn.metrics import mean_squared_error
        previsto = [1, 2, 3]
        realizado = [1, 2, 3]
        mean_squared_error(realizado, previsto)

Out[7]: 0.0

In [8]: previsto = [-1, 0, 1]
        realizado = [1, 2, 3]
        mean_squared_error(realizado, previsto)

Out[8]: 4.0

In [9]: previsto = [-10, 7.9, 4]
        realizado = [-3, 7, 8]
        mean_squared_error(realizado, previsto)

Out[9]: 21.936666666666667
```

Exercício 21: EQM de Ŷ

```
knn = KNeighborsRegressor(n_neighbors=K)
x, y = boston.data[:50], boston.target[:50]
y_ = knn.fit(x, y).predict(x)
plt.plot(np.linspace(-1, 1, 50), y, label='data', color='black')
plt.plot(np.linspace(-1, 1, 50), y_, label='prediction', color='red')
plt.legend()
plt.show()
print 'Erro Quadrado Médio', mean_squared_error(y, y_)
```




```
Erro Quadrado Médio 18.8287679012
```

Diabetes dataset

- Vamos usar um dataset nativo do sklearn
- Constituído por 10 variáveis fisiológicas, como: idade, sexo, peso e pressão sanguínea
- 442 instâncias
- A saída varia entre 25 e 346

Exercício 22: carga do dataset

Diabetes Dataset

```
from sklearn import datasets
diabetes = datasets.load_diabetes()
# objetos do dataset
dir(diabetes)
```

```
['data', 'target']
```

```
diabetes.data
```

```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986,  0.00306441]])
```

Exercício 22: carga do dataset

Diabetes Dataset

```
from sklearn import datasets
diabetes = datasets.load_diabetes()
# objetos do dataset
dir(diabetes)
```

```
['data', 'target']
```

```
diabetes.data
```

```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986,  0.00306441]])
```

Exercício 23: estatísticas

```
diabetes.target[:80]
```

```
array([ 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.])
```

```
# Total de amostras
len(diabetes.target)
```

```
442
```

```
# Valor mínimo de saída
diabetes.target.min()
```

```
25.0
```

```
# Valor máximo de saída
diabetes.target.max()
```

```
346.0
```

Exercício 24: avaliação

```
from sklearn.model_selection import train_test_split

x, y = diabetes.data, diabetes.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
print 'Tamanho da base de treino:', len(x_train)
print 'Tamanho da base de testes:', len(x_test)
```

```
Tamanho da base de treino: 309
Tamanho da base de testes: 133
```

```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=17, p=2)
knn.fit(x_train, y_train)
outputs = knn.predict(x_test)
print('Saída esperada: %f' % y_test[3])
print('Saída predita: %f' % outputs[3])

from sklearn.metrics import mean_squared_error
print 'Erro Quadrado Médio:', mean_squared_error(y_test, outputs)
```

```
Saída esperada: 230.000000
Saída predita: 226.941176
Erro Quadrado Médio: 2978.11421287
```

Método k-fold

- É um método de validação cruzada que divide o conjunto total de dados em K subconjuntos mutuamente exclusivos
- 1 conjunto para teste e k-1 para treino
- O processo é realizado k vezes, trocando o conjunto de testes e calculando a acurácia
- Acurácia do modelo = média das acurácias

Leave-one-out

- É um caso específico do k-fold, onde k é igual ao número total de amostras da base N
- Nesta abordagem, cada instância é usada para testar o modelo treinado com todas as demais, permitindo uma análise completa
- Tem alto custo computacional, sendo indicado apenas para bases pequenas

Exercício 25: carga da base

Cross-validation

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
```

```
iris = datasets.load_iris()

iris.data.shape, iris.target.shape
```

```
((150, 4), (150,))
```

Exercício 26: holdout

```
# Split da base em treino e teste
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.3, random_state=0)

print 'X_train.shape:', X_train.shape, 'y_train.shape:', y_train.shape
print 'X_test.shape: ', X_test.shape,  'y_test.shape: ', y_test.shape
```

```
X_train.shape: (105, 4) y_train.shape: (105,)
X_test.shape:  (45, 4) y_test.shape:  (45,)
```

```
# Criação e treino do classificador
clf = KNeighborsClassifier(n_neighbors=5, p=2).fit(X_train, y_train)
# Usando o método holdout
clf.score(X_test, y_test)
```

```
0.9777777777777775
```

Exercício 27: k-folds

- Documentação:
  http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

```
# Avaliação com Cross-Validation com 10 folds (cv=10)
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, iris.data, iris.target, cv=10)
print 'Acurácias:', scores
```

```
Acurácias: [ 1.          0.93333333  1.          1.          0.86
0.93333333
  0.93333333  1.          1.          1.        ]
```

```
print 'Acurácia do modelo usando 10-folds', scores.mean()
```

```
Acurácia do modelo usando 10-folds 0.966666666667
```

Exercício 28: cross_val_predict

```
'''
The function cross_val_predict has a similar interface to cross_val_score,
but returns, for each element in the input, the prediction that was obtained
for that element when it was in the test set.
Only cross-validation strategies that assign all elements to a test set exactly once
can be used (otherwise, an exception is raised).
'''
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score
# Lista de resultados previstos
predicted = cross_val_predict(clf, iris.data, iris.target, cv=10)
print 'Previsões:', predicted
print 'Acurácia do modelo usando 10-folds', accuracy_score(iris.target, predicted)
```

```
Previsões: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Acurácia do modelo usando 10-folds 0.966666666667
```

FIM

Contatos

- Márcio Palheta
  - marcio.palheta@gmail.com
  - @marciopalheta
  - https://sites.google.com/site/marciopalheta/

Bibliografia

- LIVRO: Apress - Beginning Python From Novice to Professional
- LIVRO: O'Relly - Learning Python
- http://www.python.org
- http://www.python.org.br

- Mais exercícios:
  - http://wiki.python.org.br/ListaDeExercicios

- Documentação do python:
  - https://docs.python.org/2/

Classificação e Regressão com kNN em scikit-learn

Márcio Palheta, M.Sc.
marcio.palheta@gmail.com