Introdução às Redes Neurais Artificiais

Pré-processamento e Redução de Dimensionalidade

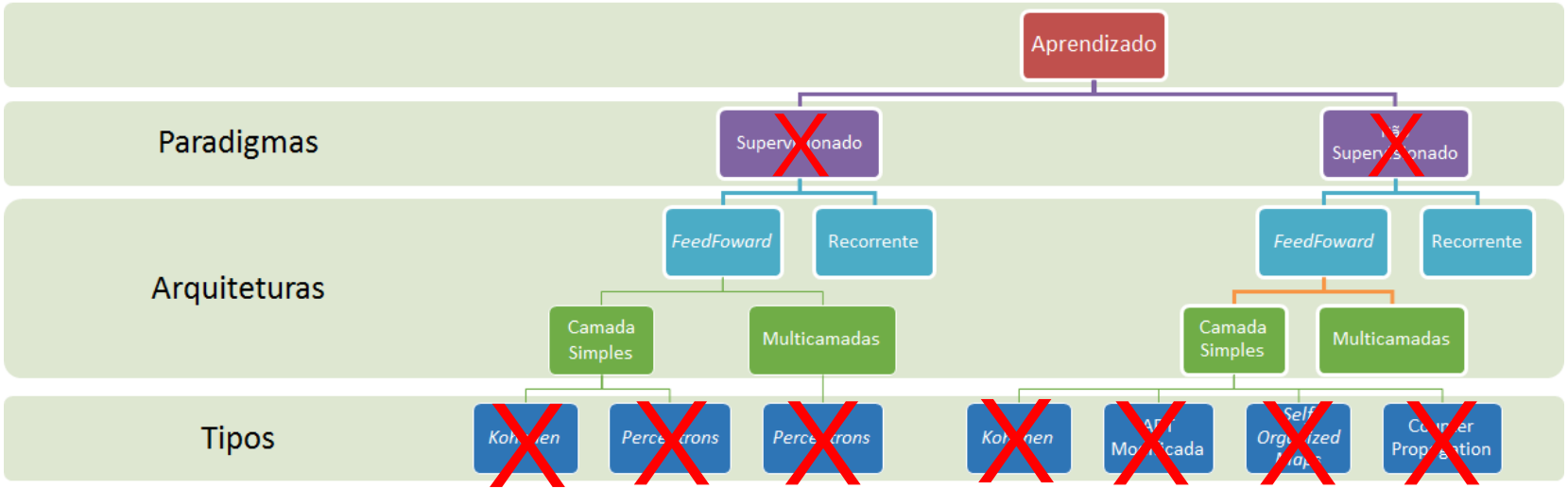
Prof. João Marcos Meirelles da Silva
jmarcos@id.uff.br

Universidade Federal Fluminense

www.latelco.uff.br
www.professores.uff.br/jmarcos

Mapa de Aprendizado

Redução de Dimensionalidade | Tipos de Dados | Detecção de Anomalias



Introdução

Em geral, os dados a serem entregues a uma rede neural, seja para treinamento ou para operação, necessitam de algum tipo de pré-processamento. Como, por exemplo:

1. Normalização dos valores de cada atributo;

2. Média zero dos valores de atributo e variância unitária;

3. Transformações para exacerbar determinados atributos;

4. Selecionar atributos que contribuam mais, eliminando os que contribuem menos.

Introdução

5. Tratamento de dados ausentes;

6. Busca e eliminação de *outliers* → Próxima aula!

Escalonamento e Remoção da Média

Na prática, normalmente ignoramos a forma da distribuição de dados e apenas centralizamos os dados removendo a *média* de cada atributo, depois escalonamos os atributos não constantes dividindo-os por seus *desvio-padrões*.

Ex: $\begin{bmatrix} 1.0 & -1.0 & 2.0 \\ 2.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & -1.0 \end{bmatrix}$ scale $\begin{bmatrix} 0.0 & -1.22 & 1.33 \\ 1.22 & 0.0 & -0.26 \\ -1.22 & 1.22 & -1.06 \end{bmatrix}$

Média = $\begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix}$

desvio-padrão = $\begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix}$

Escalonamento e Remoção da Média

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> X_scaled = preprocessing.scale(X_train)

>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

```
>>> X_scaled.mean(axis=0)
array([ 0.,  0.,  0.])

>>> X_scaled.std(axis=0)
array([ 1.,  1.,  1.])
```

Escalonamento e Remoção da Média

O pacote *preprocessing* inclui uma classe chamada de *StandardScaler* que permite computar a média e a variância sobre um conjunto de testes e, depois, aplicar os mesmo valores para escalonar os dados de testes.

```
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler(copy=True, with_mean=True, with_std=True)

>>> scaler.mean_
array([ 1. ...,  0. ...,  0.33...])

>>> scaler.scale_
array([ 0.81...,  0.81...,  1.24...])

>>> scaler.transform(X_train)
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Escalonamento e Remoção da Média

O objeto *scaler* pode ser utilizado depois sobre novos dados para transformá-los, do mesmo jeito que foi feito sobre os dados de treinamento.

```
>>> X_test = [[-1., 1., 0.]]
>>> scaler.transform(X_test)
array([[-2.44...,  1.22..., -0.26...]])
```

Escalonamento de Atributos em um Intervalo

Uma forma alternativa de padronização dos dados é escalonar os atributos dentro de um intervalo de valores, geralmente entre 0 e 1, de forma que o valor máximo de cada atributo seja 1. Isso pode ser feito usando *MinMaxScaler* e *MaxAbsScaler*, respectivamente.

A motivação para usar essa escala é a de preservar desvios padrões pequenos dos atributos e preservando entradas nulas em dados esparsos.

Escalonamento de Atributos em um Intervalo

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[ 0.5       ,  0.        ,  1.        ],
       [ 1.        ,  0.5       ,  0.33333333],
       [ 0.        ,  1.        ,  0.        ]])
```

Escalonamento de Atributos em um Intervalo

Se valores explícitos forem passados para *MinMaxScale* como *feature_range = (min, max)*, a fórmula de calcula será:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min
```

Escalonamento de Atributos em um Intervalo

A classe *MaxAbsScaler* funciona de uma forma muito parecida, porém escalonando os dados no intervalo [-1, 1]. A classe pressupõe que os dados já estão centralizados na origem.

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> max_abs_scaler = preprocessing.MaxAbsScaler()
>>> X_train_maxabs = max_abs_scaler.fit_transform(X_train)
>>> X_train_maxabs                # doctest +NORMALIZE_WHITESPACE^
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
>>> X_test = np.array([[ -3., -1.,  4.]])
>>> X_test_maxabs = max_abs_scaler.transform(X_test)
>>> X_test_maxabs
array([[-1.5, -1. ,  2. ]])
>>> max_abs_scaler.scale_
array([ 2.,  1.,  2.])
```

Escalonamento de Dados Esparsos

A centralização de dados esparsos destruiria a estrutura de esparsidade dos dados, logo, raramente é uma boa ideia fazê-lo.

- Analise se a esparsidade é natural ou por falha na coleta de dados.

- Se for falha na coleta, use as classes *MaxAbsScaler* e *maxabs_scale*.

- Ver a documentação do pacote *preprocessing* para maiores informações.

Normalização

- Normalização é o processo de escalonamento individual de cada amostra de dados de forma a ter módulo unitário;

- Útil caso você use uma forma quadrática, como produto interno, para quantificar similaridade de qualquer par de amostras;

- A função *normalize* permite uma forma fácil e rápida de normalização, usando a norma L_1 ou L_2.

Normalização

```
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]
>>> X_normalized = preprocessing.normalize(X, norm='l2')

>>> X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

Normalização

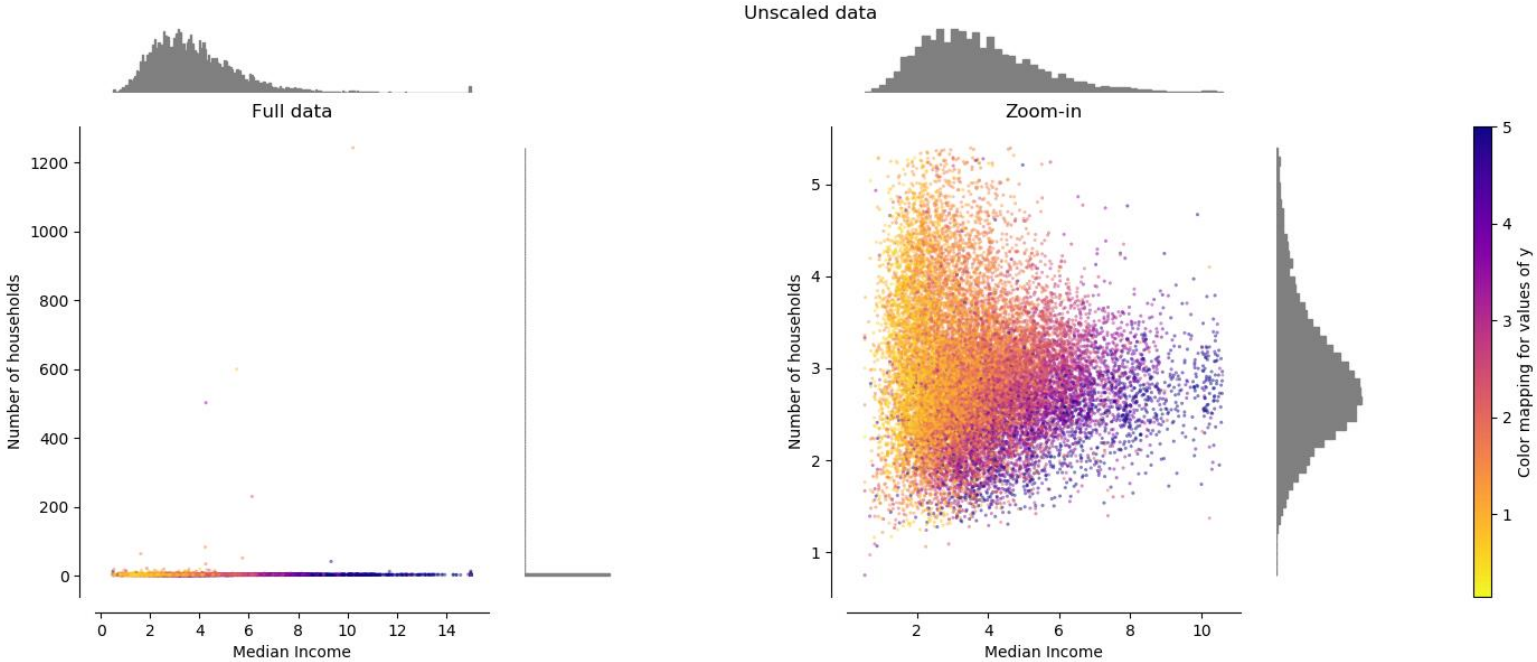
O pacote *preprocessing* inclui uma classe chamada de *Normalizer*, que permite normalizar os dados de treinamento e, em seguida, aplicar os mesmos parâmetros de normalização sobre os dados de testes.

```
>>> normalizer = preprocessing.Normalizer().fit(X)  # fit does nothing
>>> normalizer
Normalizer(copy=True, norm='l2')
```

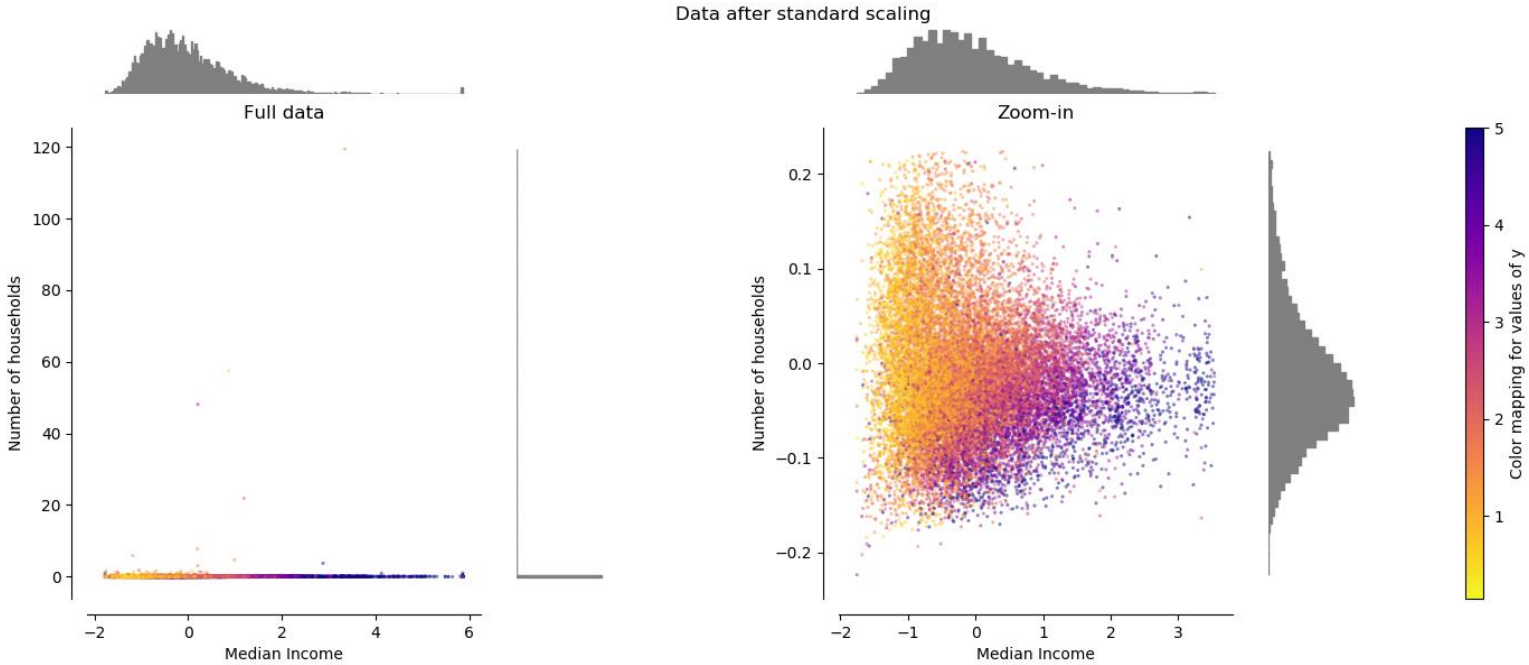
```
>>> normalizer.transform(X)
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])

>>> normalizer.transform([[-1.,  1., 0.]])
array([[-0.70...,  0.70...,  0.  ...]])
```

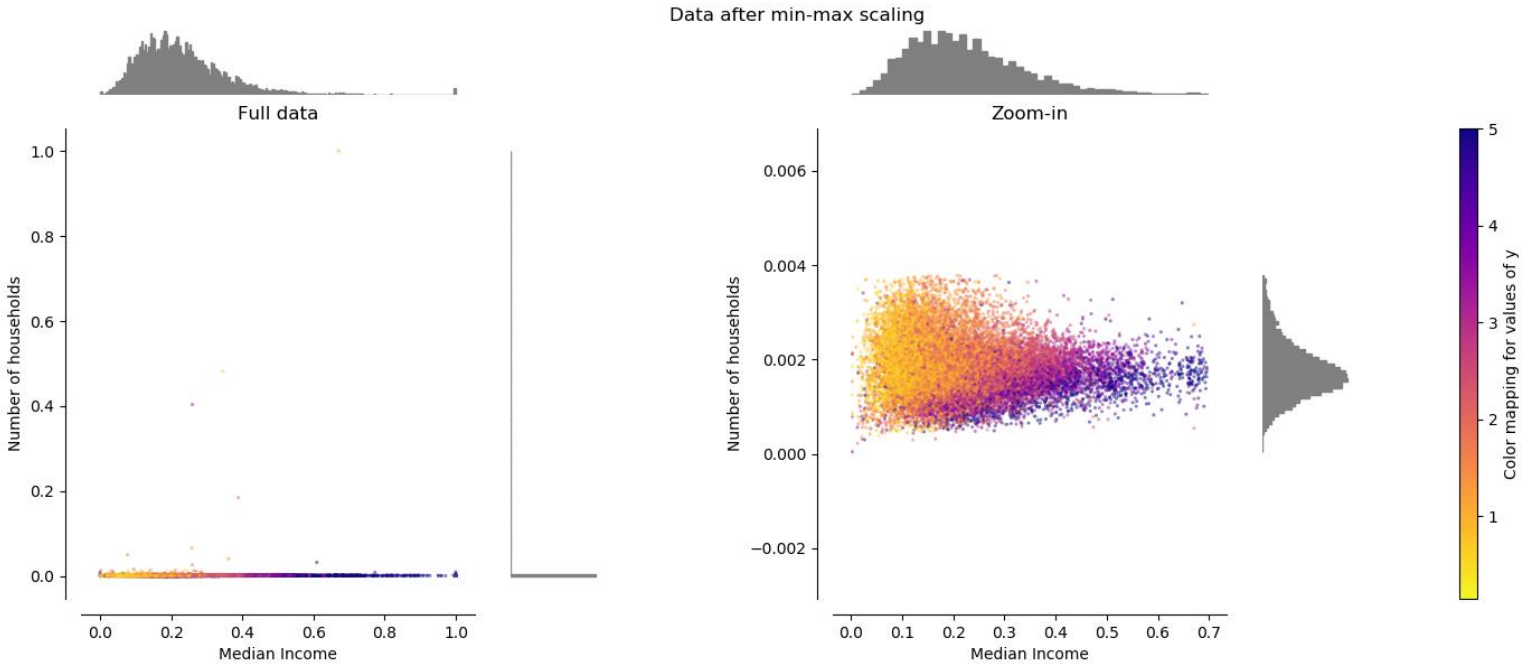
Dados Originais



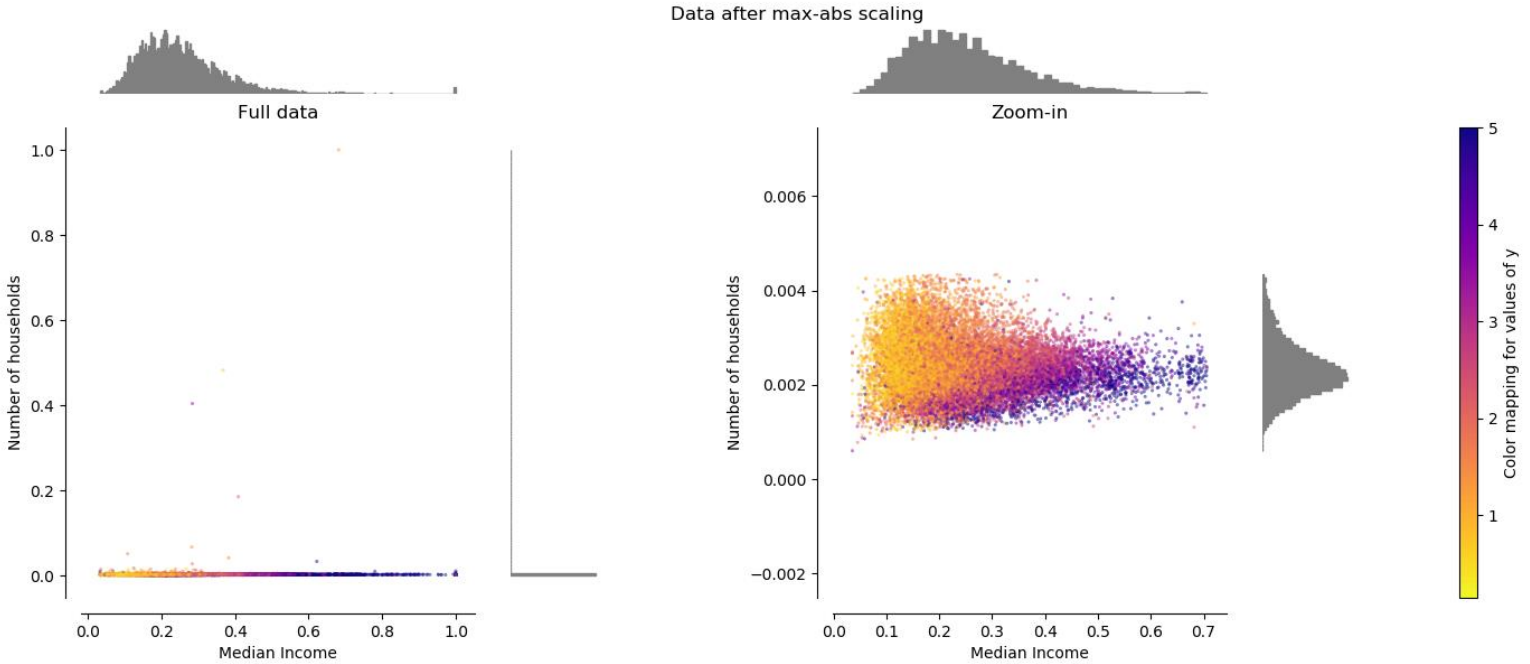
StandardScaler



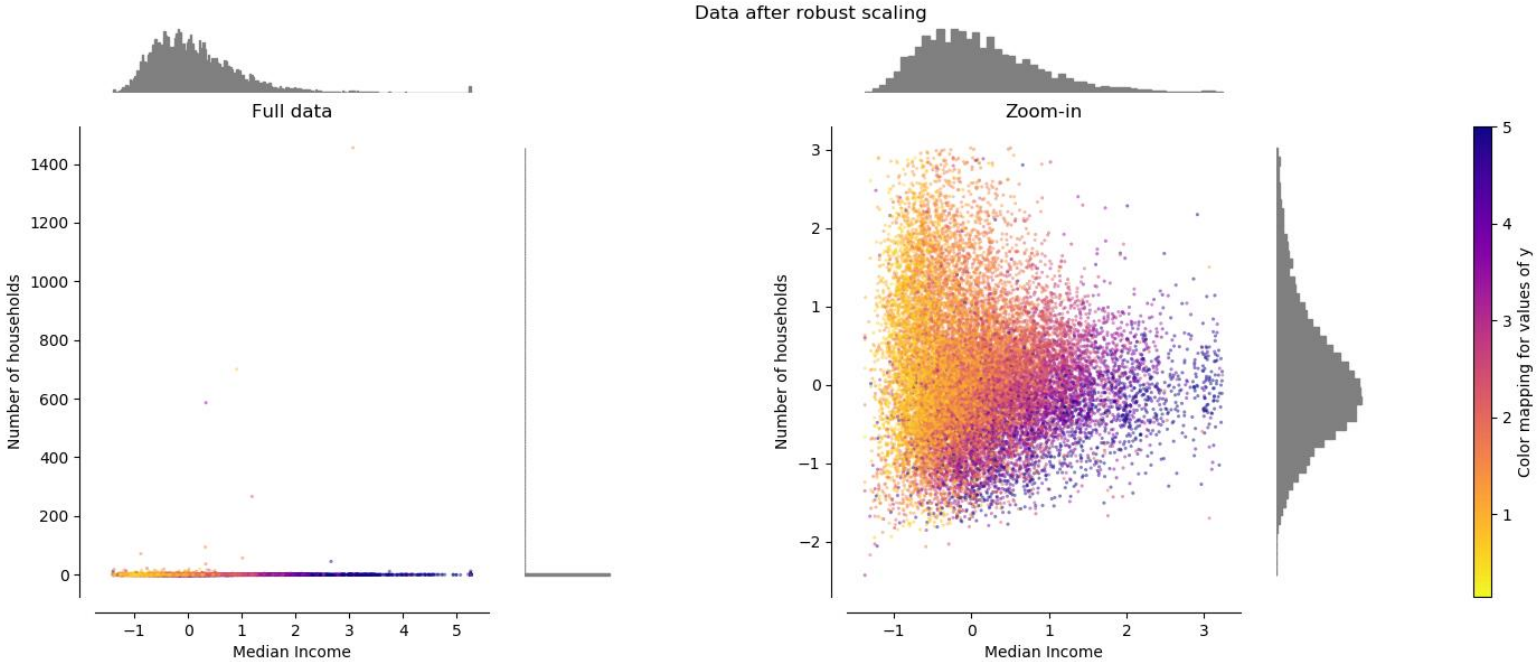
MinMaxScaler



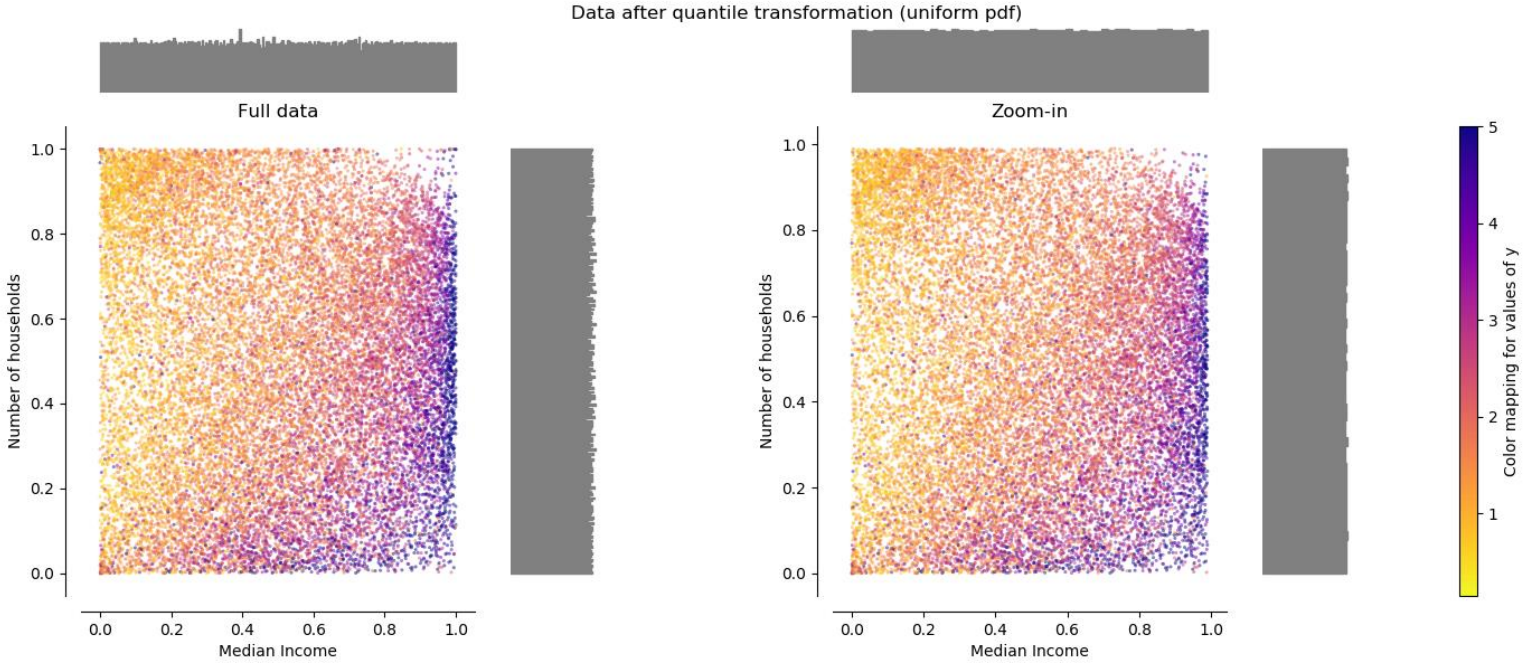
MaxAbsScaler



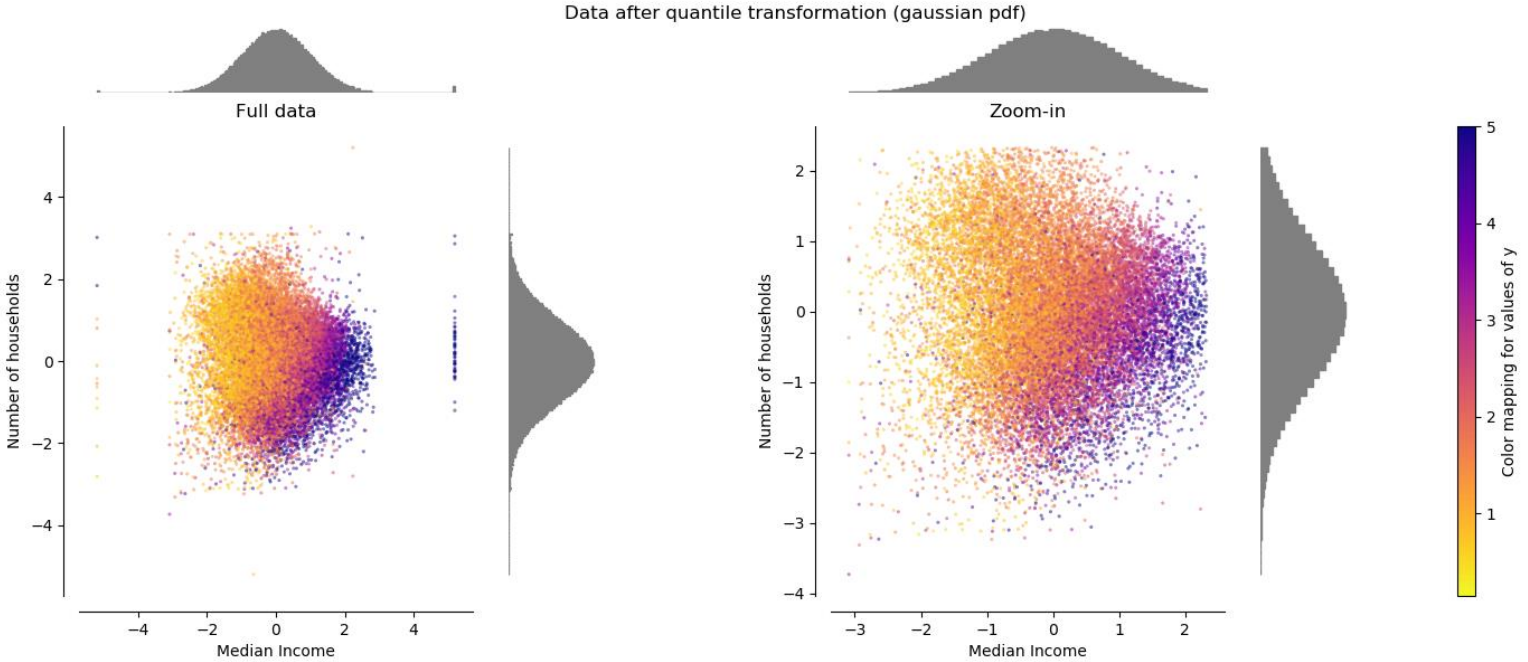
RobustScaler



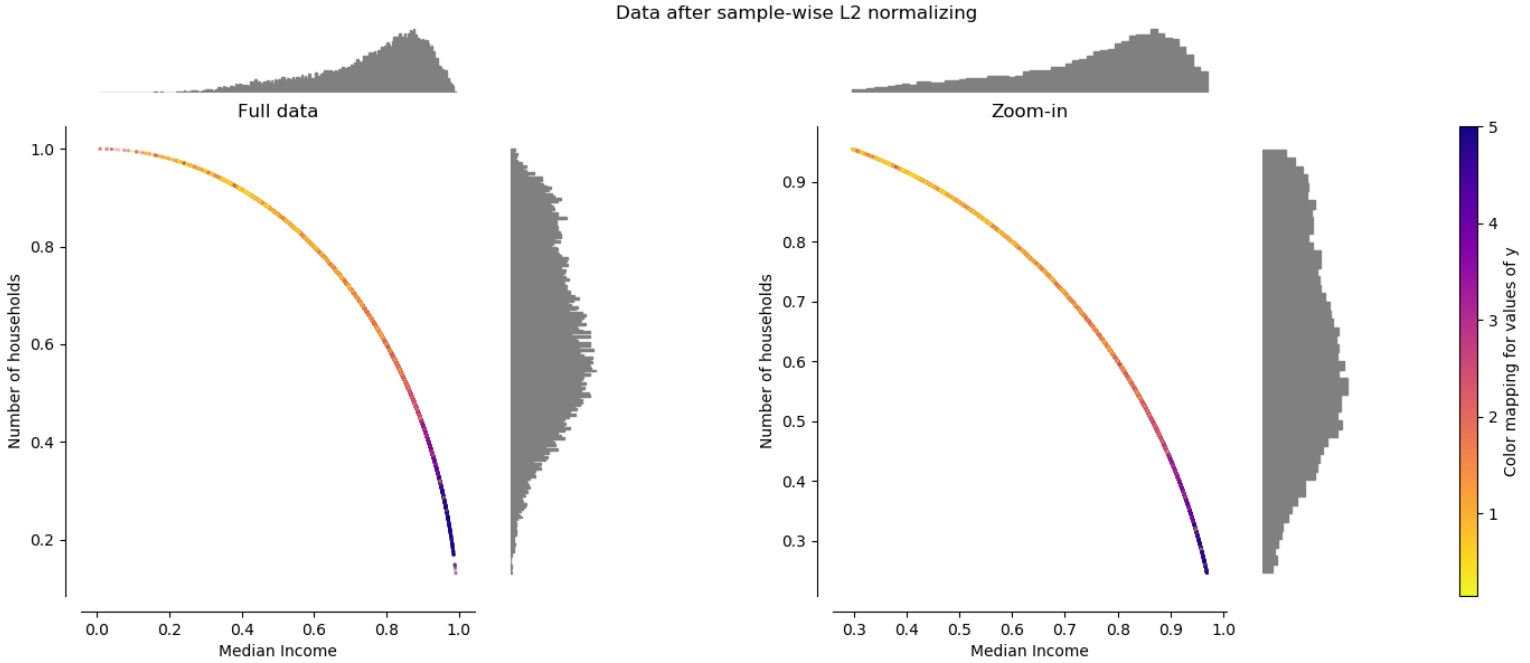
QuantileTransformer (Saída Uniforme)



QuantileTransformer (Saída Gaussiana)



QuantileTransformer (Saída Gaussiana)



Codificação de Atributos Categóricos

- Atributos categóricos como, por exemplo: ["Homem", "Mulher"], ["Brasil", "EUA", "França"], ["FireFox", "Chrome", "IE", "Safari"].

  Geralmente são codificados como:

  Ex: ["Mulher", "França", "FireFox"] = [1, 2, 0]

- Tal representação não pode ser utilizada com o *scikit-learn* (assumiria que há uma ordenação nos dados).

Codificação de Atributos Categóricos

- Deve-se usar a classe: *OneHotEnconder*.

Ex:

Codificação normal:

["Brasil", "EUA", "França"] → [0, 1, 2, 3]

Usando OneHotEnconder:

["Brasil", "EUA", "França"] → $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Codificação de Atributos Categóricos

Total = 9

```
>>> enc = preprocessing.OneHotEncoder(n_values=[2, 3, 4])
>>> # Note that there are missing categorical values for the 2nd and 3rd
>>> # features
>>> enc.fit([[1, 2, 3], [0, 2, 0]])
OneHotEncoder(categorical_features='all', dtype=<... 'numpy.float64'>,
       handle_unknown='error', n_values=[2, 3, 4], sparse=True)
>>> enc.transform([[1, 0, 0]]).toarray()
array([[ 0.,  1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.]])
```

Total = 9

Valores Ausentes ou Constantes

1) Não é incomum que um objeto não tenha um ou mais valores de atributos:

• Informações não foram coletadas: Pessoas podem declinar de revelar idade ou rendimentos, sensores podem falhar, etc.

2) Ou o atributo seja constante, ou aproximadamente constante:

• Informações que não apresentam variância ou variância muito pequena.

Valores Ausentes ou Constantes

Algumas estratégias para lidar com valores ausentes:

- Eliminar instâncias onde o dado está ausente

- Preencher o dados ausente através de alguma estimativa:
    a. Média aritmética
    b. Polinomial
    c. Aleatoriamente segundo a distribuição probabilística mais adequada

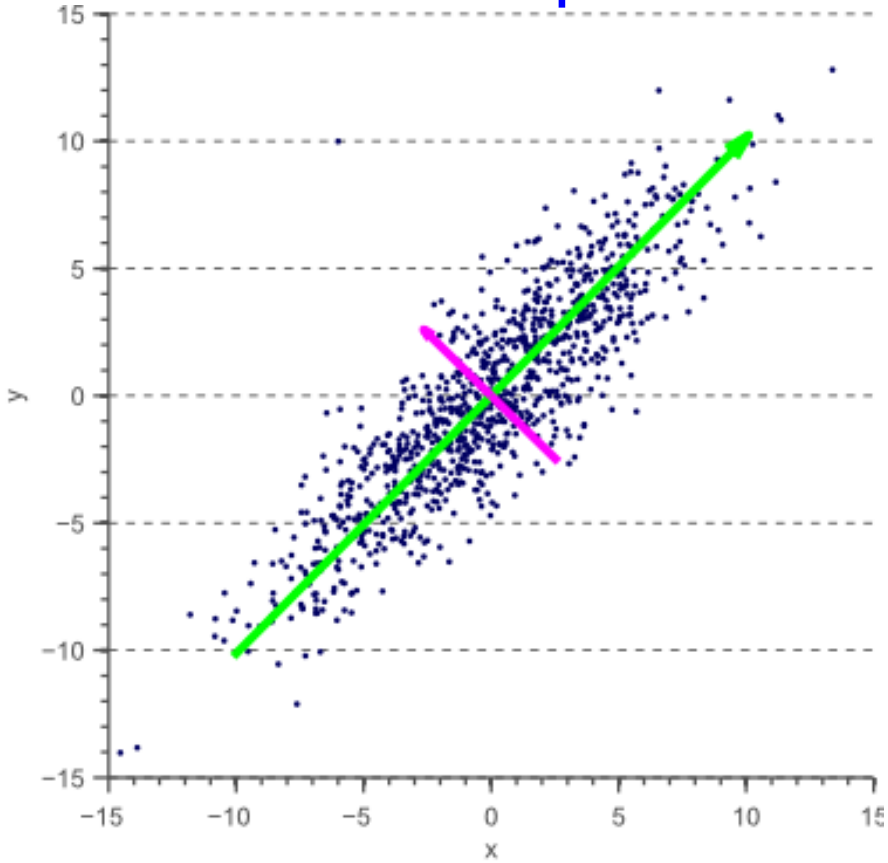
Valores Ausentes ou Constantes

```
>>> import numpy as np
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
Imputer(axis=0, copy=True, missing_values='NaN', strategy='mean', verbose=0)
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[ 4.          2.        ]
 [ 6.          3.666...]
 [ 7.          6.        ]]
```

The `Imputer` class also supports sparse matrices:

```
>>> import scipy.sparse as sp
>>> X = sp.csc_matrix([[1, 2], [0, 3], [7, 6]])
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit(X)
Imputer(axis=0, copy=True, missing_values=0, strategy='mean', verbose=0)
>>> X_test = sp.csc_matrix([[0, 2], [6, 0], [7, 6]])
>>> print(imp.transform(X_test))
[[ 4.          2.        ]
 [ 6.          3.666...]
 [ 7.          6.        ]]
```

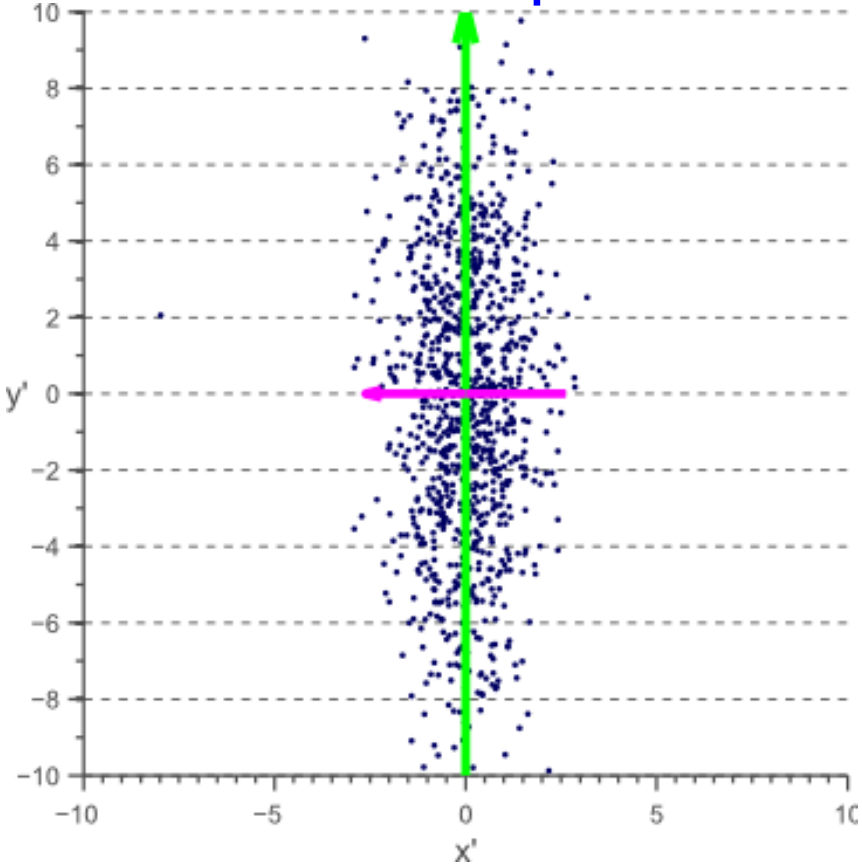
Análise de Componentes Principais



Observando a figura, podemos notar claramente que há uma correlação entre x e y. De fato, a matriz de correlação dos pontos é dada por:

$$\Sigma = \begin{bmatrix} 16.87 & 14.94 \\ 14.94 & 17.27 \end{bmatrix}$$

Análise de Componentes Principais



Os dados X podem ser descorrelacionados rotacionando cada um deles de forma que seus autovetores V tornem-se o nome sistema de eixos.

$$X' = V.X$$

A matriz de covariância para X' é dada por:

$$\Sigma' = \begin{bmatrix} 1.06 & 0.0 \\ 0.0 & 16.0 \end{bmatrix}$$

Análise de Componentes Principais

Calculando os autovetores da matriz de convariância inicial, temos que:

$$V = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$$

$\underline{v}_1$ $\underline{v}_2$

Podemos notar que a matriz de autovetores V é uma matriz de rotação, correspondendo a uma rotação de 45o, pois cos(45) = 0,7071.

Algoritmo

Seja uma base de dados $X_{m \times N}$, onde m é o número de dimensões de cada dado experimental (número de atributos) e N o número total de amostras:

1. Eliminar instâncias onde o dado está ausente
2. Faça um vetor de médias de cada atributo:

$$\underline{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix}^T$$

Onde:

$$u_i = \frac{1}{N} \sum_{j=1}^{N} X(i, j) \qquad i = 1, 2, \ldots, m.$$

Algoritmo

3. Calcule o desvio de cada média, subtraindo o vetor $\underline{u}$ da matriz de dados X:

$$B_{mxN} = X_{mxN} - \underline{u} I_{1xN}$$

4. Encontre a matriz de covariâncias:

$$\Sigma = \frac{1}{N} \sum BB^T$$

Algoritmo

5. Encontre os autovalores e autovetores da matriz de covariâncias:

$$V^{-1}\Sigma V = D$$

onde D é a matriz diagonal contendo os autovalores.

6. Reordene os autovetores e autovalores (λ_i) em ordem crescente (dos autovalores)

Algoritmo

7. Calcule a "energia cumulativa" de cada autovetor. Os autovalores representam a distribuição de energia da fonte de informação.

$$g_i = \sum_{j=1}^{m} d_{jj} \quad , \qquad \text{j = 1, 2, ..., N.}$$

Prática 01

1. Digite e execute o script PCAIris.py
2. Compare com os *plots* original feito na aula 04.
3. O que você pode concluir a partir do resultado da PCAIris.py?

Prática 02

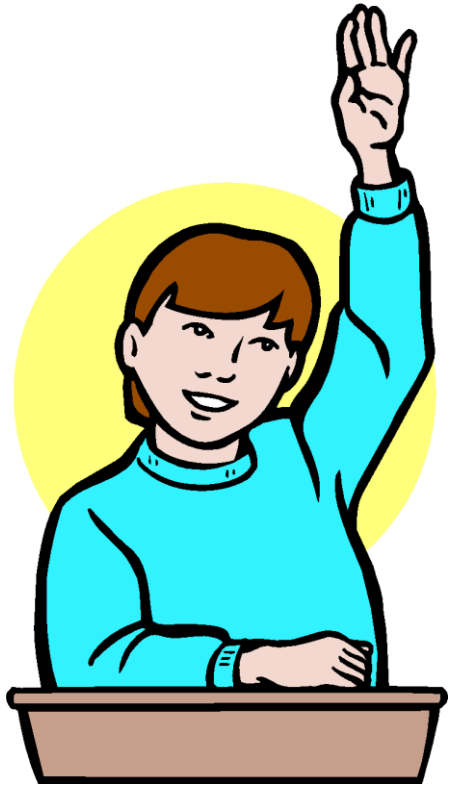
1. Acesse a url http://scikit-learn.org/stable/modules/decomposition.html#decompositions
2. Leia atentamente a seção 2.5.1.3
3. Execute os exemplos:
   a. Faces recognition example using eigenfaces and SVMs
   b. Faces dataset decompositions
4. Discuta em grupos os resultados e apresente-os

Referências

1. *Neural Networks and Learning Machines*, 3rd. Edition, Simon Haykin

2. *Fundamental of Neural Networks - Architectures, Algorithms and Applications*, Laurene Fausett

3. *Pattern Classification*, Richard O. Duda, Peter E. Hart, David G. Stork



Perguntas ?

Deixe suas perguntas nos comentários!

Inscreva-se no canal e ative o sino para ficar sabendo das próximas aulas e dos próximos cursos !!!