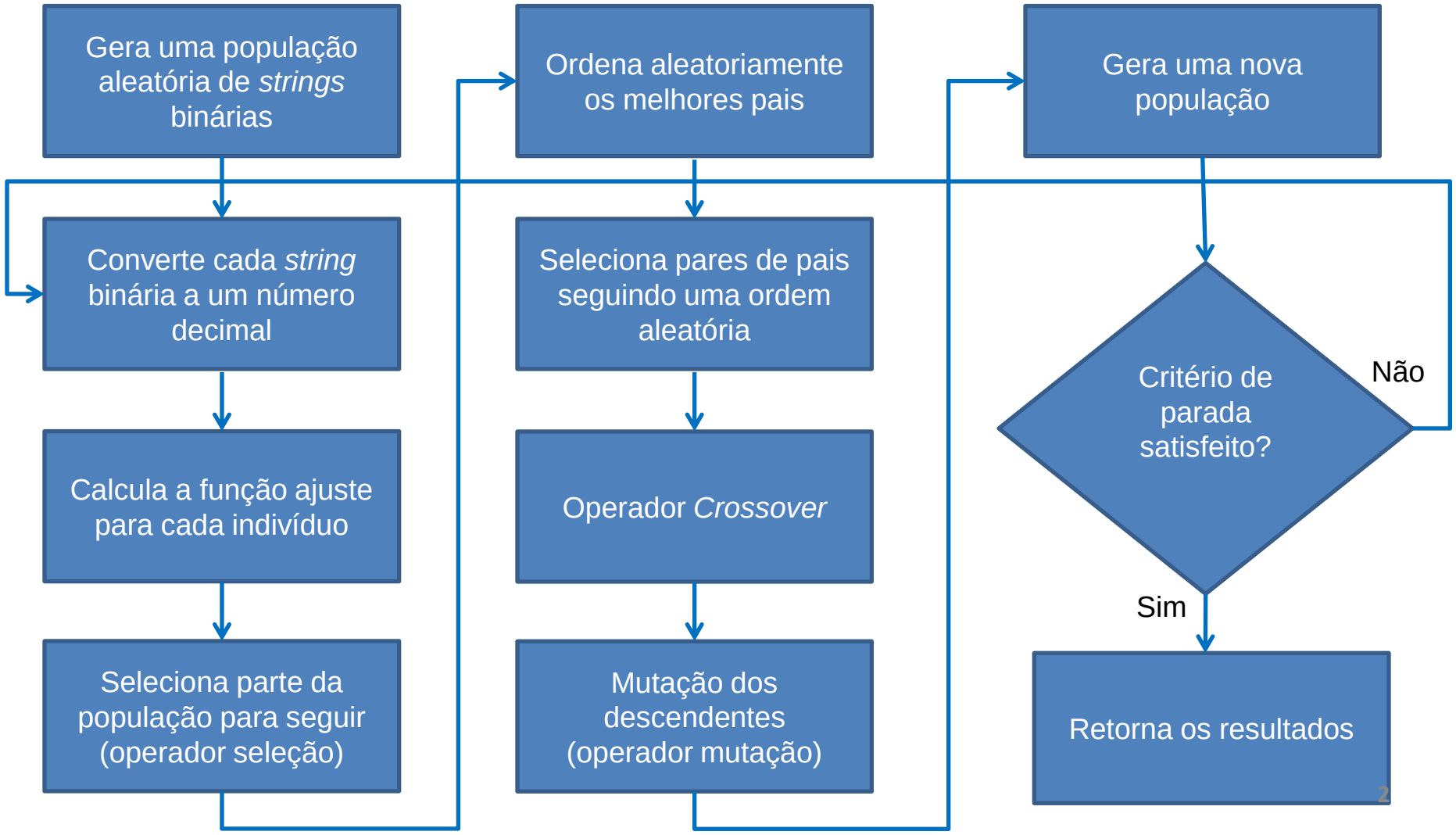
Aprendizado de Máquina. Uma Introdução com a Linguagem Python

Aula 06

azevedolab.net

Vimos na aula passada um algoritmo genético simples para resolver o problema do máximo da função x^2. Veremos sua implementação em Python.



Na função *main()* do programa *GA01.py* inicialmente definimos os valores da simulação.

```
def main():

    # Set up initial values for GA
    pop_size    = 8             # Population size
    max_iter    = 50            # Number of iterations
    p_cross     = 0.8           # Probability of cross-over
    len_str     = 12            # Length of strings
    p_mut       = 0.17          # Probability of mutation

    # Call gen_bin_strings()
    pop_in = gen_bin_strings(pop_size,len_str)

    # Call basic_ga()
    pop = basic_ga(pop_in,pop_size,max_iter,p_cross,len_str,p_mut)

    # Show final population
    print("\nFinal population")
    for line in pop:
        print(line)

main()
```

Em seguida é chamada a função *gen_bin_strings()* que gera as strings binárias.

```
def main():

    # Set up initial values for GA
    pop_size    = 8             # Population size
    max_iter    = 50            # Number of iterations
    p_cross     = 0.8           # Probability of cross-over
    len_str     = 12            # Length of strings
    p_mut       = 0.17          # Probability of mutation

    # Call gen_bin_strings()
    pop_in = gen_bin_strings(pop_size,len_str)

    # Call basic_ga()
    pop = basic_ga(pop_in,pop_size,max_iter,p_cross,len_str,p_mut)

    # Show final population
    print("\nFinal population")
    for line in pop:
        print(line)

main()
```

Agora é aplicado o algoritmo genético na população inicial gerada, com a evocação da função *basic_ga()*, que retorna a população final.

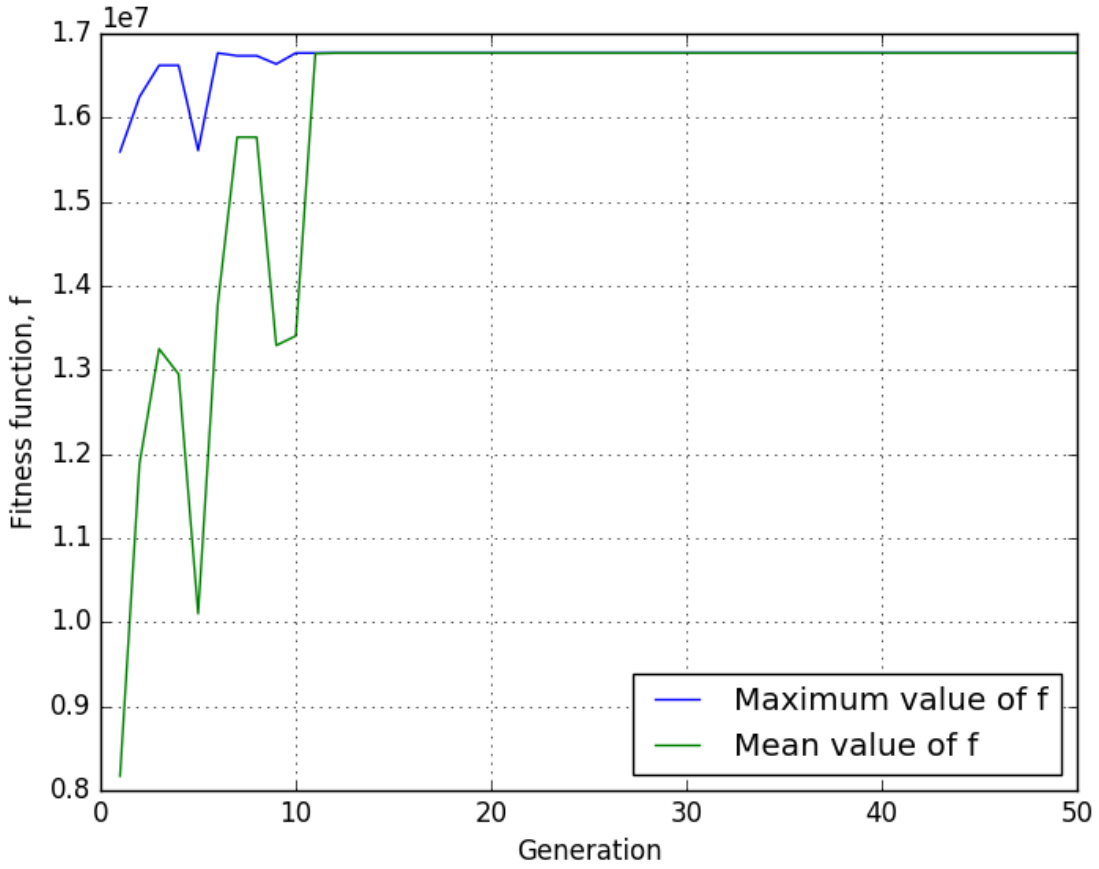
```
def main():

    # Set up initial values for GA
    pop_size    = 8             # Population size
    max_iter    = 50            # Number of iterations
    p_cross     = 0.8           # Probability of cross-over
    len_str     = 12            # Length of strings
    p_mut       = 0.17          # Probability of mutation

    # Call gen_bin_strings()
    pop_in = gen_bin_strings(pop_size,len_str)

    # Call basic_ga()
    pop = basic_ga(pop_in,pop_size,max_iter,p_cross,len_str,p_mut)

    # Show final population
    print("\nFinal population")
    for line in pop:
        print(line)

main()
```

Por último é mostrada a população final na tela.

```
def main():

    # Set up initial values for GA
    pop_size    = 8             # Population size
    max_iter    = 50            # Number of iterations
    p_cross     = 0.8           # Probability of cross-over
    len_str     = 12            # Length of strings
    p_mut       = 0.17          # Probability of mutation

    # Call gen_bin_strings()
    pop_in = gen_bin_strings(pop_size,len_str)

    # Call basic_ga()
    pop = basic_ga(pop_in,pop_size,max_iter,p_cross,len_str,p_mut)

    # Show final population
    print("\nFinal population")
    for line in pop:
        print(line)

main()
```

Ao rodarmos o programa *GA01.py* temos o seguinte resultado.

```
Final population
111111111111
111111111111
111111111111
111111111111
111111111111
111111111111
111111111111
111111111111
```



Abaixo temos uma tabela com dados experimentais sobre diversos parâmetros físico-químicos de 20 aminoácidos naturais. As variáveis explanatórias são as seguintes: LCE e LCF são parâmetros que descrevem a lipofilicidade, FET é a energia de livre observada ao transferirmos o aminoácido de um solvente orgânico para água, POL indica a polarização, VOL é o volume molecular e ASA é área acessível ao solvente.

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Usaremos esta tabela para discutir alguns métodos relacionados à normalização dos dados e a aplicação do método da análise principal de componentes (*Principal Components Analysis).* Normalmente na aplicação de métodos de aprendizado de máquina nos defrontamos com conjuntos de dados que apresentam uma ampla distribuição de valores enquanto outras variáveis no mesmo conjunto de dados não apresentam tanta variação.

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Quando elaboramos modelos de regressão linear múltipla, usando-se variáveis explanatórias com grandes variações podemos ter um superdimensionamento do peso obtido por regressão para aquela variável. Uma das formas de evitar tal situação é por meio da normalização (*scaling*).

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Para isto precisamos calcular o desvio padrão (σ) e o coeficiente de variação (v), conforme as equações abaixo.

$$\langle x \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \langle x \rangle\right)^2} \qquad v = \frac{\sigma}{\langle x \rangle}$$

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Uma das formas de normalização é chamada *autoscaling,* para isto temos que dividir as variáveis explanatórias (x_i) pelo desvio padrão (σ), como indicado abaixo.

$$\langle x \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \langle x \rangle\right)^2} \qquad v = \frac{\sigma}{\langle x \rangle}$$

$$x_i' = \frac{x_i}{\sigma}$$

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

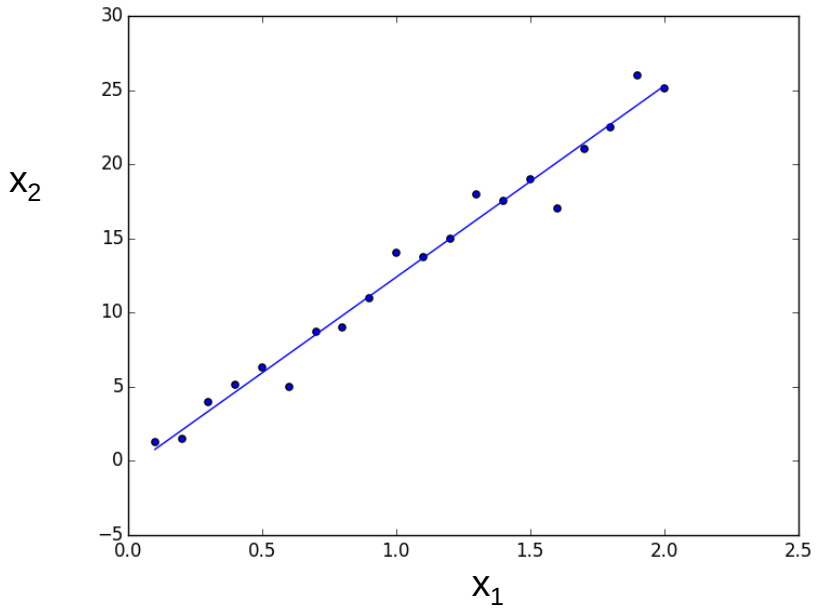
Outra forma é chamada de *mean centering* e tem a seguinte forma:

$$\langle x \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \langle x \rangle\right)^2} \qquad v = \frac{\sigma}{\langle x \rangle}$$

$$\boxed{x_i' = \frac{x_i - \langle x \rangle}{\sigma}}$$

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Quando geramos um modelo de regressão linear múltipla podemos ter variáveis explanatórias que apresentam correlação alta entre elas. Tal situação indica que podemos diminuir o número de variáveis em nosso modelo. Podemos usar uma combinação linear de variáveis que apresentam correlação alta entre si. Vejamos um exemplo bidimensional para ilustrar o caso. No gráfico abaixo vemos claramente a correlação entre as variáveis explanatórias x_1 e x_2. Num modelo de regressão poderíamos substituir a participação destas duas variáveis inserindo uma nova variável (z) que fosse uma combinação linear de x_1 e x_2. A nova variável z é chamada componente principal (*principal component)* (PC).



Para um caso com p-dimensões temos as seguintes equações para os componentes principais (PCs):

$$PC_1 = c_{1,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$
$$PC_2 = c_{2,1}x_1 + c_{2,2}x_2 + ... + c_{2,p}x_p$$
$$PC_3 = c_{3,1}x_1 + c_{3,2}x_2 + ... + c_{3,p}x_p$$
$$....$$
$$PC_i = c_{i,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$

Onde p indica o número de variáveis explanatórias e i o número de componentes principais.

Para determinarmos os coeficientes c's das equações abaixo partimos da tabela de dados, que será apresentada na forma de uma matriz com n linhas (número de dados) por p colunas (número de variáveis explanatórias). Abaixo temos a matriz nxp que chamaremos de matriz X.

aa	LCE	LCF	FET	POL	VOL	ASA
ala	0.23	0.31	-0.55	-0.02	82.2	254.2
arg	-0.79	-1.01	2	-2.56	163	363.4
asn	-0.48	-0.6	0.51	-1.24	112.3	303.6
asp	-0.61	-0.77	1.2	-1.08	103.7	287.9
cys	0.45	1.54	-1.4	-0.11	99.1	282.9
gln	-0.11	-0.22	0.29	-1.19	127.5	335
glu	-0.51	-0.64	0.76	-1.43	120.5	311.6
gly	0	0	0	0.03	65	224.9
his	0.15	0.13	-0.25	-1.06	140.6	337.2
ile	1.2	1.8	-2.1	0.04	131.7	322.6
leu	1.28	1.7	-2	0.12	131.5	324
lys	-0.77	-0.99	0.78	-2.26	144.3	336.6
met	0.9	1.23	-1.6	-0.33	132.3	336.3
phe	1.56	1.79	-2.6	-0.05	155.8	366.1
pro	0.38	0.49	-1.5	-0.31	106.7	288.5
ser	0	-0.04	0.09	-0.4	88.5	266.7
thr	0.17	0.26	-0.58	-0.53	105.3	283.9
trp	1.85	2.25	-2.7	-0.31	185.9	401.8
tyr	0.89	0.96	-1.7	-0.84	162.7	377.8
val	0.71	1.22	-1.6	-0.13	115.6	295.1

Matriz X

Usaremos a matriz X para calcular a matriz de covariança-variança, chamada aqui de S. A matriz S tem dimensões nxn e é calculada como segue:

$$S = XX^T$$

Onde X^T é a transposta da matriz X.
Os coeficientes c's das equações abaixo são os autovetores da matriz S acima.

$$PC_1 = c_{1,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$
$$PC_2 = c_{2,1}x_1 + c_{2,2}x_2 + ... + c_{2,p}x_p$$
$$PC_3 = c_{3,1}x_1 + c_{3,2}x_2 + ... + c_{3,p}x_p$$
$$....$$
$$PC_i = c_{i,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$

Foge ao objetivo do curso entrar nos detalhes da álgebra linear para determinação dos autovetores. Do ponto de vista de programação, temos métodos em Python para o seu cálculo.

Vamos ilustrar a aplicação do método PCA com o programa *pca1.py*. Vejamos a função *main*. Inicialmente é feita a leitura do arquivo *aa_table01.csv*, a partir da evocação da função *read_2_Matrix()* que retorna a matriz de dados.

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

Em seguida é calculada a matriz de correlação, esta etapa é meramente ilustrativa. A matriz de correlação é calculada pela função *corr_matrix().*

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

A função *zero_mean_data()* normaliza os dados lidos e retorna para a matriz *YC*.

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

A função *covariance_matrix()* determina a matriz de covariança-variança vista anteriormente.

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

Agora são calculados os autovalores e autovetores com a função *eigen_val_vect()*.

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

A função *PCA()* calcula os componentes principais.

```
def main():
    # Data file name
    data_file = "aa_table01.csv"

    # Call read_csv()
    X = read_CSV_2_Matrix(data_file)

    # Call corr_matrix()
    corr_matrix(X)

    # Call zero_mean_data()
    YC = zero_mean_data(X)

    # Call covariance_matrix()
    S = covariance_matrix(YC)

    # Call eigen_val_vect()
    _,U = eigen_val_vect(S)

    # Call PCA()
    PCA(YC,U)

main()
```

A função *read_CSV_2_Matrix()* faz a leitura dos dados e retorna uma matriz. Vamos destacar as novidades em vermelho. Uma forma de evitar erro de leitura ao digitarmos um nome de arquivo que não existe é com o comando *try.*

```
def read_CSV_2_Matrix(file_in):
    """Function to read a CSV file and return a matrix"""
    # Import libraries
    import sys
    import scipy as sp
    import numpy as np
    # Try to open CSV file
    try:
        data = sp.genfromtxt(file_in, delimiter=",")
    except IOError:
        sys.exit("I can't find "+file_in+" file!")
    x = data[1:,1:]  # data[1:,1:] for second column and second line
    # Get number of rows and columns
    rows= len(x[:])
    columns = len(x[0])
    # Define a zero matrix
    A = np.array([[0]*(columns)]*rows,float)
    # Looping through data to get a matrix
    print("\nData:")
    for i in range(rows):
        aux = x[i]
        for j in range(columns):
            A[i][j] = aux[j]
            print(A[i][j],end=" ")
        print()
    return A
```

O *try* testa a linha de comando vinculada a ele e em caso de erro de leitura executa as linhas vinculadas ao *except IOError:* Se deixarmos só *except:* pegaremos qualquer erro que ocorra, não só o erro de leitura.

```
def read_CSV_2_Matrix(file_in):
    """Function to read a CSV file and return a matrix"""
    # Import libraries
    import sys
    import scipy as sp
    import numpy as np
    # Try to open CSV file
    try:
        data = sp.genfromtxt(file_in, delimiter=",")
    except IOError:
        sys.exit("I can't find "+file_in+" file!")
    x = data[1:,1:]  # data[1:,1:] for second column and second line
    # Get number of rows and columns
    rows= len(x[:])
    columns = len(x[0])
    # Define a zero matrix
    A = np.array([[0]*(columns)]*rows,float)
    # Looping through data to get a matrix
    print("\nData:")
    for i in range(rows):
        aux = x[i]
        for j in range(columns):
            A[i][j] = aux[j]
            print(A[i][j],end=" ")
        print()
    return A
```

Definimos uma matriz com *np.array()*. Depois varremos os dados de entrada num loop para montarmos a matriz A. Por último retornamos a matriz.

```
def read_CSV_2_Matrix(file_in):
    """Function to read a CSV file and return a matrix"""
    # Import libraries
    import sys
    import scipy as sp
    import numpy as np
    # Try to open CSV file
    try:
        data = sp.genfromtxt(file_in, delimiter=",")
    except IOError:
        sys.exit("I can't find "+file_in+" file!")
    x = data[1:,1:]  # data[1:,1:] for second column and second line
    # Get number of rows and columns
    rows= len(x[:])
    columns = len(x[0])
    # Define a zero matrix
    A = np.array([[0]*(columns)]*rows,float)
    # Looping through data to get a matrix
    print("\nData:")
    for i in range(rows):
        aux = x[i]
        for j in range(columns):
            A[i][j] = aux[j]
            print(A[i][j],end=" ")
        print()
    return A
```

A função *corr_matrix()* calcula a matriz de correlação que mostra o coeficiente de correlação entre as variáveis explanatórias. Usamos o método *pearsonr* da biblioteca *scipy* para o cálculo da correlação.

```
def corr_matrix(A):
    """Function to determine the correlation matrix"""

    # Import libraries
    import numpy as np
    from scipy.stats import pearsonr

    # Get the number of rows and columns
    _,columns = np.shape(A)

    # Set up a matrix
    B = np.array( [[0]*columns]*columns,float)

    # Looping rows and columns of matrix A
    print("\nCorrelation matrix:")
    for i in range(columns):
        for j in range(columns):
            corr,_ = pearsonr(A[:,i],A[:,j])
            B[i][j] = corr
            print(B[i][j],end=" ")
        print()

    return B
```

A função *zero_mean_data()* calcula o valor médio de cada variável explanatória e subtrai este valor de cada elemento na coluna, retornando uma nova matriz.

```
def zero_mean_data(X):
    """Function to subtract the mean of column from each element in a column"""

    # Import library
    import numpy as np

    # Get rows and columns
    rows, columns = np.shape(X)

    # Set up a zero array
    my_mean = np.zeros(columns,float)

    # Define a matrix
    Y = np.array([[0]*columns]*rows,float )

    # Looping through columns to get the mean for each column
    for i in range(columns):
        my_mean[i] =  np.mean(X[:,i])

    # Looping through data to get zero mean data
    print("\nZero-mean data:")
    for i in range(rows):
        for j in range(columns):
            Y[i][j] = X[i][j] - my_mean[j]
            print(Y[i][j],end=" ")
        print()
    return Y
```

A função *covariance_matrix()* calcula a matriz de covariança-variança como foi definido anteriormente.

```
def covariance_matrix(Y):
    """Function to calculate covariance matrix"""

    # Import library
    import numpy as np

    # Get rows and columns
    rows, columns = np.shape(Y)

    # Determine the transpose
    YT = np.transpose(Y)

    # Determine the covariance matrix (S)
    S = (1/(rows-1))*np.dot(YT,Y)

    # Show results
    print("\nCovariance matrix:")
    for i in range(columns):
        for j in range(columns):
            print(S[i][j],end=" ")
        print()

    return S
```

A função *eigen_val_vector()* usa um método da biblioteca *NumPy* para calcular os autovalores e autovetores, como mostrado abaixo.

```
def eigen_val_vect(A):
    """Function to calculate eigenvalues and eigenvectors for a given nxn matrix"""

    # Import library
    import numpy.linalg as linalg

    # Get eigenvectors
    eVal, eVec = linalg.eig(A)

    # Show results
    print("\nEigenvalues: ",eVal)
    print("\nMatrix of eigenvectors:")
    print(eVec)

    return eVal, eVec
```

A *PCA()* calcula os componentes principais e mostra na tela.

```
def PCA(YC,U):
    """Function to calculate principal components"""

    # Import library
    import numpy as np

    # Principal components
    F = np.dot(YC,U)

    # Show results
    print("\nPrincipal components:")
    print(F)
```

Ao rodarmos o código *pca1.py* temos os seguintes resultados. Inicialmente o programa mostra a matriz X.

```
Data:
0.23 0.31 -0.55 -0.02 82.2 254.2
-0.79 -1.01 2.0 -2.56 163.0 363.4
-0.48 -0.6 0.51 -1.24 112.3 303.6
-0.61 -0.77 1.2 -1.08 103.7 287.9
0.45 1.54 -1.4 -0.11 99.1 282.9
-0.11 -0.22 0.29 -1.19 127.5 335.0
-0.51 -0.64 0.76 -1.43 120.5 311.6
0.0 0.0 0.0 0.03 65.0 224.9
0.15 0.13 -0.25 -1.06 140.6 337.2
1.2 1.8 -2.1 0.04 131.7 322.6
1.28 1.7 -2.0 0.12 131.5 324.0
-0.77 -0.99 0.78 -2.26 144.3 336.6
0.9 1.23 -1.6 -0.33 132.3 336.3
1.56 1.79 -2.6 -0.05 155.8 366.1
0.38 0.49 -1.5 -0.31 106.7 288.5
0.0 -0.04 0.09 -0.4 88.5 266.7
0.17 0.26 -0.58 -0.53 105.3 283.9
1.85 2.25 -2.7 -0.31 185.9 401.8
0.89 0.96 -1.7 -0.84 162.7 377.8
0.71 1.22 -1.6 -0.13 115.6 295.1
```

Em seguida é mostrada a matriz de correlação, onde os elementos indicam os coeficientes correlação (R) entre as variáveis explanatórias. A correlação entre duas variáveis explanatórias x_i e x_j é dada por:

$$R = \frac{\sum_{k=1}^{N}\left[\left(x_{i,k} - \langle x_i \rangle\right)\left(x_{j,k} - \langle x_j \rangle\right)\right]}{\sqrt{\sum_{k=1}^{N}\left(x_{i,k} - \langle x_i \rangle\right)^2 \sum_{k=1}^{N}\left(x_{j,k} - \langle x_j \rangle\right)^2}}$$

```
Correlation matrix:
1.0 0.970991028036 -0.965011588456 0.735762447015 0.373703092234 0.38332862329
0.970991028036 1.0 -0.963971624253 0.771643849468 0.28326220809 0.290237882987
-0.965011588456 -0.963971624253 1.0 -0.777742320994 -0.263714399555 -0.274160879527
0.735762447015 0.771643849468 -0.777742320994 1.0 -0.34932010546 -0.3311224516
0.373703092234 0.28326220809 -0.263714399555 -0.34932010546 1.0 0.990447280629
0.38332862329 0.290237882987 -0.274160879527 -0.3311224516 0.990447280629 1.0
```

Em seguida é feito o deslocamento para o centro.

```
Covariance matrix:
0.603742105263 0.775944736842 -1.00267105263 0.437289473684 8.71963157895
13.0552368421
0.775944736842 1.05774184211 -1.32572763158 0.607033157895 8.74831052632
13.0837342105
-1.00267105263 -1.32572763158 1.78813552632 -0.795502631579 -10.5896052632 -
16.0691710526
0.437289473684 0.607033157895 -0.795502631579 0.585074736842 -8.02370526316 -
11.1015105263
8.71963157895 8.74831052632 -10.5896052632 -8.02370526316 901.760947368
1303.66152632
13.0552368421 13.0837342105 -16.0691710526 -11.1015105263 1303.66152632
1921.21313158
```

Agora é calculada a matriz de covariança-variança.

```
Zero-mean data:
-0.095 -0.1605 0.0975 0.663 -41.51 -60.805
-1.115 -1.4805 2.6475 -1.877 39.29 48.395
-0.805 -1.0705 1.1575 -0.557 -11.41 -11.405
-0.935 -1.2405 1.8475 -0.397 -20.01 -27.105
0.125 1.0695 -0.7525 0.573 -24.61 -32.105
-0.435 -0.6905 0.9375 -0.507 3.79 19.995
-0.835 -1.1105 1.4075 -0.747 -3.21 -3.405
-0.325 -0.4705 0.6475 0.713 -58.71 -90.105
-0.175 -0.3405 0.3975 -0.377 16.89 22.195
0.875 1.3295 -1.4525 0.723 7.99 7.595
0.955 1.2295 -1.3525 0.803 7.79 8.995
-1.095 -1.4605 1.4275 -1.577 20.59 21.595
0.575 0.7595 -0.9525 0.353 8.59 21.295
1.235 1.3195 -1.9525 0.633 32.09 51.095
0.055 0.0195 -0.8525 0.373 -17.01 -26.505
-0.325 -0.5105 0.7375 0.283 -35.21 -48.305
-0.155 -0.2105 0.0675 0.153 -18.41 -31.105
1.525 1.7795 -2.0525 0.373 62.19 86.795
0.565 0.4895 -1.0525 -0.157 38.99 62.795
0.385 0.7495 -0.9525 0.553 -8.11 -19.905
```

O programa calcula e mostra os autovalores (eigenvalues) e autovetores (eigenvectors).

```
Eigenvalues:  [  2.81163072e+03   1.17506612e+01   3.54507120e+00   1.48304025e-03
   5.31300159e-02   2.77120010e-02]

Matrix of eigenvectors:
[[ 0.00558751  0.02039751  0.37552961  0.74181029 -0.26465913  0.48806073]
 [ 0.00560302  0.02220063  0.51380307 -0.0158183  -0.54069933 -0.66548941]
 [-0.00685091 -0.03896199 -0.67382422  0.03216318 -0.73345131  0.07355721]
 [-0.00486667  0.03798402  0.37059703 -0.66950749 -0.3156645   0.55973835]
 [ 0.56381016 -0.82445989  0.04475932 -0.0138486  -0.00183196  0.01361793]
 [ 0.82582375  0.56249101 -0.03999113  0.00086437 -0.00123493 -0.00417549]]
0.325
0.4705
-0.6475
-0.683
123.71
315.005
```

Por último são mostrados os componentes principais.

```
Principal components:
[[ -7.36232977e+01   3.69479217e-02   6.35568273e-01   1.36168650e-02
   -1.77376452e-02   1.27333971e-01]
 [  6.20943133e+01  -5.40133588e+00  -3.83573800e+00   3.58376140e-02
   -3.85451908e-01  -8.18411263e-02]
 [ -1.58673090e+01   2.88543603e+00  -1.89330667e+00  -2.19247994e-02
    1.53711502e-01  -1.48733817e-02]
 [ -3.36886940e+01   1.11745014e+00  -2.19218469e+00  -9.50722370e-02
   -2.41408393e-01   1.23565514e-01]
 [ -4.03793820e+01   2.30856109e+00   1.49824661e+00  -1.89584094e-02
   -1.55582023e-01  -5.86438244e-01]
 [  1.86389316e+01   8.04231754e+00  -1.96772406e+00   2.26252059e-02
   -7.07246518e-02   5.09630519e-04]
 ...
```

Os componentes principais podem ser usados como novas variáveis explanatórias.

$$PC_1 = c_{1,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$

$$PC_2 = c_{2,1}x_1 + c_{2,2}x_2 + ... + c_{2,p}x_p$$

$$PC_3 = c_{3,1}x_1 + c_{3,2}x_2 + ... + c_{3,p}x_p$$

$$....$$

$$PC_i = c_{i,1}x_1 + c_{1,2}x_2 + ... + c_{1,p}x_p$$

Onde p indica o número de variáveis explanatórias e i o número de componentes principais.

Estas novas variáveis explanatórias (PCs) podem ser usadas para a elaboração de um novo modelo de regressão, como mostrado abaixo.

$$y = \omega_o + \omega_1 PC_1 + \omega_2 PC_2 + ...$$

Os modelos com PCA normalmente usam um número menor de variáveis explanatórias.

Referências

-BRESSERT, Eli. SciPy and NumPy. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.

-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

-**HACKELING G.** Mastering Machine Learning with scikit-learn. Birmingham: Packt Publishing Ltd., 2014. 221 p.

-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.

-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.

-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.

-MODEL, Mitchell L. **Bioinformatics Programming Using Python**. Sebastopol: O'Reilly Media, Inc., 2011. 1584 p.

-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.

Última atualização: 29 de novembro de 2016.