

Capítulo

3

Algoritmos de Clusterização e Python Científico apoiando Modelagem de Usuário

Andrey Araujo Masiero, Leonardo Anjoletto Ferreira e Plinio Thomaz Aquino Junior

Abstract

Python is an interactive, interpreted, object oriented, multiplatform programming language that has various applications in industry and in scientific works. There are many libraries available for Python that aid the development of many applications, including in Human-Computer Interaction, by making use of machine learning algorithms. These algorithms, mainly the data clusterization algorithms, are going to be presented as a way to construct the user model as Personas. Personas may aid both the initial user interface development and its enhancement during use. The moment when Personas will help the developer depends on how the information will be acquired. The information obtained by surveys or application logs may be used in pre or post development. However, the latter requires a working version of the system to collect the usage information. In this tutorial, we show how to capture these kinds of information and how to use them with clustering algorithms so that, by automating the clustering of the collected profiles, we are able to develop adaptive interfaces. Lastly, techniques used to acquire Personas will be demonstrated based on dispersion metrics methods and the Python toolkits available to present the results in a graphical manner, so that it can also be analytically compared.

Resumo

Python é uma linguagem de programação interativa, interpretada, orientada a objeto e multiplataforma que possui diversas aplicações no mercado de trabalho e científicas. Existem diversas bibliotecas disponíveis para Python que auxiliam o desenvolvimento das diversas aplicações, inclusive em Interação Humano-Computador, através de algoritmos de aprendizado de máquina. Esses algoritmos, em especial os algoritmos de clusterização de dados, serão apresentados para auxiliar na construção de modelos de usuários do tipo Personas. As Personas podem auxiliar o projetista de interface durante o projeto

desta, ou posteriormente, na melhoria de interfaces que já estão em utilização. O momento em que as Personas irão auxiliar o projetista dependerá de como as informações serão capturadas. Essas informações podem ser capturadas através de surveys ou logs de aplicação. Esses dois tipos de informação podem ser utilizados em momentos pré e pós projeto, contudo o segundo requer uma versão funcional do sistema para coletar as informações de uso. Ao longo deste tutorial será demonstrado como capturar essas informações e ainda como realizar o tratamento desses dados para aplicar os algoritmos de clusterização para agrupar os perfis coletados. Também será mostrado como automatizar esse processo para que os dados sejam utilizados em interfaces adaptativas. Por fim, serão demonstradas as técnicas para obter as Personas com base em métodos de medida de dispersão utilizadas em estatística dos dados e ainda as ferramentas disponíveis no Python para demonstração dos resultados de maneira gráfica para sua comparação analítica.

3.1. Instalação do Enthought Python Distribution

O Enthought Python Distribution (EPD) é uma distribuição de bibliotecas e interpretadores de Python reunida pela Enthought, Inc., a empresa que desenvolve e mantém as bibliotecas NumPy e SciPy.

Dentre as versões disponibilizadas pela empresa, o EPD Free oferece grande parte das bibliotecas disponibilizadas nos outros pacotes, porém não provê suporte pela empresa nem acesso a conteúdos como webinars e repositórios para atualizações. Entretanto, é possível obter a distribuição completa utilizando a EPD Academic que, apesar de necessitar de um e-mail de uma instituição de ensino, é fornecido gratuitamente para Windows, Linux e OS X e OS X 10.4 para PowerPC.

O instalador do pacote pode ser obtido no site da empresa,¹ exibido na figura 3.1a ou através da chave EPD Free nos mecanismos de busca disponíveis *on-line*. Na área de *download* é possível realizar o cadastro para o recebimento de notícias e atualizações sobre a distribuição gratuita e mais abaixo localiza-se o botão para escolha da versão que será obtida (figura 3.1b).

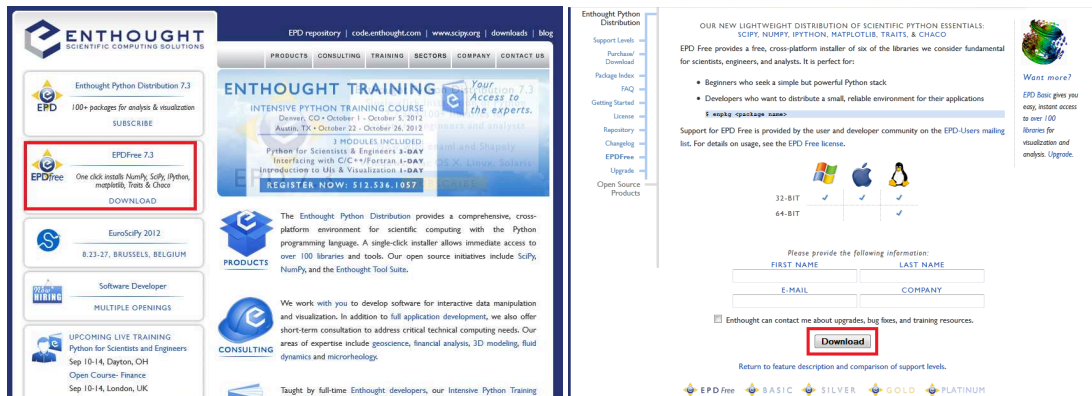
Para realizar o *download* da versão disponível (32 bits para Windows e OS X e 32/64 bits para Linux – conforme mostrado na figura 3.2), basta escolher a respectiva versão e esperar que o *download* termine. Ao final do *download* do instalador, basta executá-lo seguindo as especificações do sistema escolhido e seguir as instruções fornecidas pelo próprio instalador.

3.2. Programando com Python

Python é uma linguagem de programação de alto nível, assim como C, C++, Perl e Java. Ela também é uma linguagem interpretada, com tipagem dinâmica, interativa e possui suporte a orientação a objetos. Assim como o Java, a interpretação do Python é realizada através de *bytecodes* executados em uma máquina virtual, o que torna os programas portáteis.

A linguagem possui uma sintaxe clara e concisa, tornando o código legível e de

¹http://www.enthought.com/products/epd_free.php







(a) Site da Enthought Inc.

(b) Página para o cadastro e *download* da distribuição EPD Free.

Figura 3.1: Site da Enthought, Inc. (a) página principal e (b) página para o *download* da distribuição EPD Free. Último acesso: 18 de setembro de 2012

EPD Free installers

Name	Size	MD5
 epd_free-7.3-2-macosx-i386.dmg	72.36 MB	2e0e47e346bd5bb4b57799536a4E705f
 epd_free-7.3-2-rh5-x86.sh	91.80 MB	272c82acd37bacc4256af6433e556446
 epd_free-7.3-2-rh5-x86_64.sh	101.76 MB	4ae74b49152e28c45c0d1c050d06097e
 epd_free-7.3-2-win-x86.msi	89.28 MB	0c2db71b9d30571d49cf5f25c366e0df

Updated: 2012-08-16 15:45:07 UTC

Figura 3.2: Versões disponíveis para *download* do EPD Free. Último acesso: 18 de setembro de 2012

fácil compreensão ao desenvolvedor. Diversos tipos de estruturas, utilizadas em programação de alto nível, compõem o Python, como, por exemplo, listas e dicionários, além de um grande número de bibliotecas e *frameworks* de terceiros que podem aumentar o poder computacional dessa linguagem de programação. Python tem seu código aberto com licença compatível a GPL (*General Public License*) e é mantido pela *Python Software Foundation*.

Outro ponto forte no Python é a possibilidade de integração com outras linguagens de programação como o C. Tais características tornam o Python uma linguagem muito utilizada no mercado de trabalho, codificando *plugins* e *scripts* para ferramentas como OpenOffice, PostgreSQL, GIMP e Blender. Está presente em algumas gigantes do mercado de tecnologia como o Google, Yahoo!, Microsoft e Nokia, e também em empresas de entretenimento como a Disney, na criação de animações 3D.

3.2.1. Primeiro programa em Python

Para executar e escrever programas, o Python possui um *shell* e editor nativo, que se parece em alguns aspectos com o *prompt* de comando. O *shell* do Python é chamado de IDLE. Nele é possível executar expressões e visualizar seus resultados em tempo de execução, e também criar programas inteiros antes de executá-los. Todos os exemplos deste capítulo serão executados a partir do IDLE.

O nosso primeiro programa apresenta a instrução para exibir textos na tela. Vejam a seguir como funciona o comando `print` para exibirmos a frase "Hello, Python".

```
Python 2.7.2 |EPD 7.2-2 (32-bit)| (default, Sep 7 2011,
09:16:50)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more infor-
mation.

>>> print "Hello, Python"
Hello, Python
```

Notem no exemplo que as três primeiras linhas são compostas pelo cabeçalho do IDLE, a linha iniciada por três sinais de maior (> > >) é a instrução do programa e a linha subsequente o resultado obtido. Note que não há necessidade de nenhum ponto nem ponto-e-vírgula para encerrar a linha de código. Em Python, cada linha representa uma única instrução de programa. Caso seja necessário continuar a mesma instrução em uma linha diferente utiliza-se a barra invertida(\) para informar isso ao interpretador.

Alguns outros pontos importantes sobre a sintaxe do Python são: (I) o caractere # é utilizado para realizar comentários no código; (II) existem comentários funcionais que são pré-condições para o interpretador realizar antes de executar o programa, como por exemplo, para determinar o unicode do código "# - * - coding :< *tipodoencoding* >"; e (III) os blocos de código no Python são delimitados através de endentação e qualquer instrução antes de começar um bloco é finalizada com dois pontos(:).

Ao longo desta seção os exemplos da sintaxe do Python serão demonstrados em detalhes. Continuando os exemplos, será apresentada a atribuição de valores em variáveis. Como não existe tipagem explícita de variáveis em Python, pode-se atribuir qualquer tipo de valor. Para identificar o tipo da variável em algum ponto do código existe uma função chamada `type`. O código a seguir demonstra as atribuições e o uso da função `type`.

```
>>> x = 5
>>> print x
5

>>> y = 3.
>>> print y
3.0
```

```
>>> z = 'texto'
>>> print z
texto

>>> type(x)
<type 'int'>

>>> type(y)
<type 'float'>

>>> type(z)
<type 'str'>
```

Com os pontos básicos apresentados, serão abordados nas próximas subseções os pontos fundamentais sobre a linguagem para que consigamos atingir o objetivo de gerar o modelo de usuário *Personas* a partir dos algoritmos de *clustering* com Python.

3.2.2. Operadores Matemáticos

Python contém sete operadores matemáticos básicos. Na tabela 3.1 são apresentados cada um dos operadores e as descrições que representam as operações por eles executadas.

Alguns exemplos de uso dos operadores matemáticos podem ser observados no trecho abaixo.

```
>>> 3 + 5
8
>>> 3 ** 2
9
>>> 5.4 // 2
2.0
>>> 8 - 3
5
```

Além dos operadores matemáticos básicos existe a combinação de cada um desses com a atribuição de valores, facilitando alguns tipos de operações, como na tabela 3.2.

Veja o exemplo abaixo:

```
>>> base = 4
>>> expoente = 2
>>> print base
4
>>> base**=expoente
```

```
>>> print base
16
```

Tabela 3.1: Operadores Matemáticos em Python

Operador	Descrição
+	realiza a operação de adição de valores
-	realiza a operação de subtração de valores
*	realiza a operação de multiplicação de valores
/	realiza a operação de divisão de valores
%	realiza a operação de módulo na divisão de valores
**	realiza a operação de potência de valores
//	realiza a operação de divisão com arredondamento para o valor inteiro mais baixo

Tabela 3.2: Operadores Matemáticos Combinados a Atribuição

Operador	Descrição
+=	quando temos uma operação b+=a significa b = b + a
-=	quando temos uma operação b-=a significa b = b - a
=	quando temos uma operação b=a significa b = b * a
/=	quando temos uma operação b/=a significa b = b / a
%=	quando temos uma operação b%=a significa b = b % a
=	quando temos uma operação b=a significa b = b ** a
//=	quando temos uma operação b//=a significa b = b // a

3.2.3. Operadores de Comparação

Além dos operadores matemáticos, Python possui operadores de comparação utilizados quando se quer verificar alguma condição durante o programa. Eles são muito utilizados com elementos de controle de fluxo e laços de repetições, que serão apresentados nas próximas subseções. Na tabela 3.3 pode-se verificar quais são os operadores de comparação existentes no Python.

3.2.4. Controle de Fluxo

O controle de fluxo é utilizado para possibilitar tomadas de decisão ao longo de um programa. Tais tomadas de decisão determinam a sequência em que o programa será executado. Dessa forma, controles de fluxo são utilizados para controle do caminho feito pelo programa.

Em Python existe apenas um tipo de declaração para controle de fluxo de um programa, essa declaração é a `if . . else`. Ela funciona de forma semelhante a qualquer outra linguagem. Observe o exemplo abaixo.

```
>>> x = 3
>>> y = 3
```

Tabela 3.3: Operadores de Comparação em Python

Operador	Descrição
==	Verifica se o valor de um elemento é igual ou não ao valor do outro
!=	Verifica se o valor de um elemento é diferente ou não ao valor do outro
<>	Semelhante ao operador !=
>	Verifica se o valor do elemento à esquerda é maior do que o valor do elemento à direita
<	Verifica se o valor do elemento à esquerda é menor do que o valor do elemento à direita
>=	Verifica se o valor do elemento à esquerda é maior ou igual do que o valor do elemento à direita
<=	Verifica se o valor do elemento à esquerda é menor ou igual do que o valor do elemento à direita

```
>>> if x == y:
        print "X e Y possuem o mesmo valor"

X e Y possuem o mesmo valor
```

Note que é importante manter a indentação no bloco que deseja executar quando a condição for verdadeira, e ao final da linha onde é realizada a declaração da condição deve ser colocado o sinal de dois pontos para que o interpretador do Python entenda que existe a necessidade de uma linha indentada abaixo. Existe ainda a variação com a instrução `else`.

```
>>> if x!=y:
        print "X eh diferente de Y"
    else:
        print "X eh igual a Y"

X eh igual a Y
```

3.2.5. Laço de Repetição

Laços de repetição são inseridos em um programa quando se deseja que um bloco de instruções sejam executadas um certo número de vezes. O número de repetição é determinado através de uma condição ou critério de parada. No Python existem dois tipos de laços de repetição: o `while` e o `for`. O `while` repete o bloco de instrução enquanto a condição de parada retornar como verdadeira, como no exemplo abaixo.

```
>>> i = 0
>>> while i < 10:
        print i
        i+=1
```

```
0
1
.
.
.
8
9
```

No programa acima, foi solicitado que o número contido na variável `i` seja exibido enquanto `for` menor que 10, e depois disso termine o programa.

O `for` no Python é um pouco diferente das outras linguagens de programação. Nas linguagens de programação em geral, o `for` possui uma condição de parada e um incremento. Já no Python, esse laço de repetição percorre uma lista inteira, atribuindo o valor de cada posição a uma variável auxiliar. Veja no exemplo uma lista de 0 a 9 como se comporta esse tipo de declaração.

```
>>> for item in lista:
      print item

0
1
.
.
.
8
9
```

Perceba que todo o conteúdo dentro da lista é passado um por um para a variável `item`, e essa é utilizada no bloco de instrução. Para trabalhar com o `for` como nas demais linguagens é necessário o auxílio de uma função chamada `range`. A função `range` possui 3 argumentos: (I) valor de início; (II) valor final; e (III) incremento. O valor final é o único parâmetro obrigatório da função, e tem seu valor real igual a $n - 1$, pois o índice no Python inicia-se em 0, como na linguagem C. Um exemplo da função `range` será demonstrado a seguir.

```
>>> for i in range(1,10,2):
      print i

1
3
5
7
9
```


3.2.6. Listas

Listas ou sequências são as estruturas de dados mais básicas que podemos encontrar em Python. Cada elemento da lista possui um índice para que possa ser acessado individualmente em qualquer ponto do programa. O índice do primeiro elemento da lista é zero, do segundo é um e assim por diante. Contudo, é possível acessar um elemento através de um índice negativo, dessa forma é possível acessar o último elemento através do índice (-1).

```
>>> listaDeNomes = ['Jose', 'Maria', 'Joao', 'Leopoldina']
>>> listaDeNomes[1]
'Maria'
>>> listaDeNomes[-3]
'Maria'
```

Uma *string*, em Python, também é uma lista e pode-se acessar qualquer um de seus caracteres através do seu índice. Esse tipo de operação pode auxiliar o desenvolvedor caso possua uma *string* de parâmetro e em determinada posição gostaria de gravar algum valor para comparação, por exemplo.

```
>>> 'Leopoldo'[3]
'p'
```

Para manipulação das listas, existe uma técnica em Python chamada de *slicing*, onde é possível obter intervalos da lista sem a utilização de um laço de repetição. Esse tipo de operação é realizada a partir da inclusão de dois pontos no índice da lista, como nos exemplos a seguir.

```
# Lista utilizada para o exemplo.
>>> exemplo
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Selecionando um intervalo de valores.
# Posição inicial é o índice de número 4.
# Posição final é o índice de número 7, pois é percorrido
# o valor do índice final - 1.
>>> exemplo[4:8]
[5, 6, 7, 8]

# Os índices negativos também funcionam.
>>> exemplo[-3:-1]
[8, 9]

# Dessa forma você inclui todos os elementos antes do
```

```
# índice final.
>>> exemplo[:4]
[1, 2, 3, 4]

# Dessa forma você inclui todos os elementos após o
# índice inicial.
>>> exemplo[2:]
[3, 4, 5, 6, 7, 8, 9, 10]

# Um terceiro parâmetro pode ser inserido para determinar o
# passo da seleção.
>>> exemplo[1:8:2]
[2, 4, 6, 8]

# Também pode ser utilizado para determinar a ordem da
# seleção, tornando-a crescente ou decrescente.
>>> exemplo[8:1:-2]
[9, 7, 5, 3]

>>> exemplo[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Com a apresentação das listas, todos os elementos básicos da linguagem Python foram apresentados nesse capítulo. Nas próximas seções serão apresentadas algumas das bibliotecas científicas do Python para que seja possível a manipulação dos dados e criação do modelo de usuário Persona.

3.3. Numpy - Trabalhando com Números e Matrizes em Python

NumPy² é uma biblioteca ou módulo dos mais importantes para o trabalho de pesquisas científicas, pois contém funções de alto desempenho para manipulação de dados multidimensionais, como matrizes. Esse módulo disponibiliza operações matemáticas, lógicas, de ordenação, I/O, álgebra linear e estatística básica, entre outras (Jones et al.).

As funções disponíveis no NumPy foram preparadas para trabalhar com informações numéricas e matrizes com maior desempenho computacional, o que as tornam mais recomendadas. No trecho de código abaixo é apresentado como criar vetores ou *arrays* com o NumPy, porém é necessário informar ao interpretador que o módulo será utilizado. Para isso, utiliza-se a palavra `import` mais o nome do módulo que deseja.

```
import numpy
```

Ainda é possível aplicar um *alias* para esse módulo, facilitando o seu uso durante o programa.

²NumPy Website - <http://numpy.scipy.org/>

```
# np é o alias para o módulo do numpy neste caso.  
import numpy as np
```

Agora que o módulo do NumPy foi incluído no programa, pode-se continuar com o exemplo de *arrays*.

```
>>> import numpy as np  
  
# Criando um array com o numpy. Não é diferente da lista  
# tradicional, só deve passá-la como parâmetro da função.  
>>> a = np.array([2,3,5,6])  
  
# Visualizando o array  
>>> a  
array([2, 3, 5, 6])  
  
# Existe uma outra maneira de criar este array, determinando  
# o tipo de seu conteúdo.  
# Para isso, basta utilizar o parâmetro dtype.  
>>> a = np.array([1,3,5,7],dtype = float)  
  
# Visualizando o array  
>>> a  
array([ 1.,  3.,  5.,  7.]
```

O parâmetro `dtype` também é um atributo do *array*, e pode ser utilizado para identificar o tipo de conteúdo que está armazenado durante a execução do programa, como a seguir.

```
# o atributo dtype do array equivale à função type()  
>>> a.dtype  
dtype('int32')
```

Outras funções possibilitam a conversão do tipo do conteúdo (`astype`) ou verificar o formato do *array* (`shape`). Um ponto de atenção que se deve considerar quando trabalha-se com *array* é a operação matemática. Ao realizar uma operação matemática entre *arrays*, o NumPy simplesmente realiza a operação matemática entre os elementos deste de maneira simples.

```
# Exemplo da operação de multiplicação entre dois arrays  
>>> vetor1 = np.array([2,2,2])  
>>> vetor2 = np.array([3,3,3])
```

```
>>> vetor1*vetor2
array([6, 6, 6])
```

Note que a multiplicação foi realizada da seguinte maneira: o primeiro elemento de vetor1 com o primeiro elemento de vetor2, segundo elemento de vetor1 com o segundo elemento de vetor2, e assim por diante. Para realizar a multiplicação entre vetores utiliza-se a função `dot`.

```
>>> vetor1.dot(vetor2)
18
```

A função `array` é aconselhada quando se trabalha com vetores. Para trabalhar com matrizes é recomendado utilizar a função `matrix`.

```
>>> A = np.matrix('1 2;3 4')
>>> A
matrix([[1, 2],
        [3, 4]])
```

Ao se criar um objeto com a função `matrix`, pode-se acessar os valores de matriz transposta ou de matriz inversa através dos atributos `T` e `I`, respectivamente.

```
>>> A.T
matrix([[1, 3],
        [2, 4]])

>>> A.I
matrix([[ -2. ,  1. ],
        [ 1.5, -0.5]])
```

Contudo, as operações da função `matrix` são inversas à função `array`. Veja como fica o exemplo da operação de multiplicação.

```
# Multiplicação Matricial
>>> A*A
matrix([[ 7, 10],
        [15, 22]])

# Multiplicação Elemento a Elemento
>>> np.multiply(A,A)
matrix([[ 1,  4],
        [ 9, 16]])
```

Além de uma melhor estrutura para trabalhar com vetores e matrizes, o NumPy possui algumas funções de estatística que são úteis durante a modelagem do usuário como, por exemplo, as medidas de dispersão dos dados. As quatro principais que iremos utilizar ao longo do capítulo são a média (*average*), a mediana (*median*) e o desvio padrão (*std*). Exemplos de cada uma dessas funções serão demonstrados a seguir.

```
# Vetor para demonstração das medidas de dispersão
>>> exemplo = np.array([1,2,3,4,5,6,7,8,9,10])

# Média
>>> np.average(exemplo)
5.5

# Mediana
>>> np.median(exemplo)
5.5

# Desvio Padrão
>>> np.std(exemplo)
2.8722813232690143
```

O NumPy possui diversas funções que auxiliam principalmente na manipulação de números e operações matemáticas, desenvolvidas para obter maior desempenho computacional. As funções apresentadas nesta seção são fundamentais para a modelagem de usuário que será apresentada nesse capítulo.

3.4. Scipy - Biblioteca Científica

SciPy³ é um módulo de Python com foco em cientistas, engenheiros e matemáticos. Ele é uma extensão do NumPy, porém oferece uma gama maior de métodos para estatística, otimizações, integrações, álgebra, processamento de sinais e imagens, dentre outras áreas de utilização de algoritmos (Jones et al.).

Para modelagem de usuário precisamos conhecer algumas funções presentes no SciPy, que podem auxiliar durante a execução do algoritmo ou tratamento e validação das informações coletadas a fim de gerar um modelo de usuário. As funções são encontradas nos módulos *stats*, de estatística, e *linalg*, de álgebra linear.

No módulo *stats* pode-se usar, para a criação do modelo de usuário, a função de moda (*mode*) para verificar qual valor dentre os usuários ocorre com maior frequência. Ainda nesse módulo, são encontradas as funções para auxiliar na realização de teste de hipótese, gerar números aleatórios e também a distribuição normal ou binomial dos dados. Veja a seguir o exemplo da moda.

```
# Outra boa maneira para importar um módulo
>>> from scipy import stats
```

³SciPy Website - <http://www.scipy.org/>

```
>>> import numpy as np
>>> exemplo = np.array([3, 4, 3, 3, 2, 1, 5, 6, 7, 4, 3, 4, 6, 6])
>>> stats.mode(exemplo)
(array([ 3.]), array([ 4.]))
```

Este outro exemplo apresenta como gerar uma distribuição normal aleatória.

```
>>> r = stats.norm.rvs(size=100)
```

Existem diversas funções no módulo `stats` que podem auxiliar na análise de qualquer grupo de informações. É um complemento do que já existe no NumPy, apresentado na seção 3.3. Outro módulo é o `linalg`, que possui as funções de manipulação de matrizes. Veja o exemplo de cálculo da matriz seno.

```
>>> from scipy import linalg as la
>>> M = np.matrix('1 2;4 5')
>>> la.sinm(M)
array([[ -0.31500257,  0.18115826],
       [ 0.36231652,  0.04731395]])
```

O SciPy possui um grande número de módulos que podem ser úteis em diversos tipos de aplicações. Contudo, para o propósito desse capítulo, os exemplos apresentados nesta seção são os mais importantes, dependendo da manipulação que for necessária das informações escolhidas para a modelagem de usuário.

3.5. SciKits: Expandindo o Python com Bibliotecas Científicas

Existem diversas maneiras de expandir as funcionalidades que o Python oferece em sua biblioteca padrão, desde a simples instalação de pacotes cujos instaladores podem ser obtidos nas páginas de seus projetos até a instalação e atualização a partir de gerenciadores de pacotes (como os normalmente encontrados em distribuições Linux).

Algumas dessas bibliotecas foram desenvolvidas com o objetivo específico de complementar o SciPy e oferecer novas funcionalidades em diversas áreas, como processamento de imagens, modelos estatísticos e aprendizado de máquina. O projeto que reúne essas bibliotecas específicas foi dado o nome de *SciPy Toolkits*, mais conhecido como SciKits⁴.

Na área de aprendizado de máquina, um dos projetos que fazem parte do SciKits e que possui um desenvolvimento muito ativo é o SciKit-Learn, que disponibiliza diversas técnicas de aprendizado supervisionado e não-supervisionado, validação cruzada e pré-processamento de dados.

Algumas dessas técnicas serão apresentadas de forma mais aprofundada na seção 3.6, mas uma lista extensiva de técnicas oferecida atualmente pelo SciKit-Learn é:

⁴O site oficial do projeto encontra-se em <http://scikits.appspot.com/> e as bibliotecas que atualmente fazem parte do projeto estão no endereço <http://scikits.appspot.com/scikits>

1. Aprendizado Supervisionado: Método dos Mínimos Quadrados, Regressão Linear, Logística e Bayesiana, Gradiente Descendente Estocástico, Perceptron, Análise de Discriminante Linear e Quadrática, Máquinas de Vetores de Suporte (SMV), Processo Gaussiano, Árvores de Decisão e *Naïve Bayes*, entre outros.
2. Aprendizado Não-Supervisionado: Mistura Gaussiana, *Manifold Learning*, K-Means, *Affinity Propagation*, DBSCAN, Cluster Hierárquico, Análise de Componentes Principais (PCA) e Independentes (ICA) e Modelos Escondidos de Markov, entre outros;
3. Transformação de base de dados: pré-processamento de dados (normalização, binarização e escalonamento), extração de informações em texto e imagens e aproximação de kernel (RBF, Chi Quadrado Aditivo), entre outros;
4. Validação Cruzada: *Leave-One-Out*, *Leave-P-Out* e Permutação Aleatória;
5. Base de dados de exemplos instaladas juntamente com a biblioteca, juntamente com a possibilidade de *download* direto do código de certas bases disponíveis na internet.

3.6. SciKit-Learn – Aprendizado de Máquina em Python

O Scikit-Learn é uma biblioteca para Python que oferece uma grande variedade de métodos de aprendizado de máquina, tanto nas áreas de aprendizado supervisionado quanto em aprendizado não-supervisionado.

O objetivo do desenvolvimento do Scikit-Learn é oferecer soluções simples e eficientes para problemas de aprendizado de máquina (Pedregosa et al., 2011). Por se tratar de uma biblioteca que visa a viabilização de reutilização de código de forma acessível (Pedregosa et al., 2011), o Scikit-Learn se integra facilmente com o NumPy, SciPy e Matplotlib.

Apesar do Scikit-Learn possuir funções para a realização de diversos tipos de aprendizado, neste tutorial serão apresentadas apenas três técnicas relacionadas a clusterização de dados (K-Means, DBSCAN e Affinity Propagation) e como podem ser aplicadas à área de IHC para o reconhecimento de Personas.

Fundamentalmente, o Scikit-Learn provê duas formas de realizar a classificação de uma amostra. A primeira é utilizando funções disponíveis que recebem como parâmetros, além da amostra, informações específicas do algoritmo que será utilizado, e retornam a classificação da amostra segundo os parâmetros informados.

A segunda maneira é utilizando as classes disponíveis. Estas se baseiam na classe `estimator` e implementam os métodos `fit()` e (muitas vezes) `predict()`, que são os responsáveis por realizar o aprendizado e a classificação (ou previsão, no caso de algoritmos de aprendizado supervisionado), respectivamente. A única diferença na implementação do método `fit()` entre os métodos de aprendizado supervisionado e não-supervisionado é que, no primeiro, deve-se passar como parâmetros os valores observados e as respostas, enquanto que no segundo passa-se apenas os dados da amostra.

A vantagem de se utilizar classes em vez das funções implementadas no Scikit-Learn é que as funções somente realizam a classificação durante o processo de aprendizado e não realizam a predição de um novo dado, enquanto que as classes armazenam as informações pertinentes à classificação realizada e podem ser utilizadas para verificar a qual grupo pertence um novo dado que não está contido na amostra utilizada para o treinamento.

3.6.1. Clusterização de Dados Usando o Scikit-Learn

O objetivo da clusterização é identificar similaridades entre dados, de forma que seja possível agrupar automaticamente os dados que pertençam à mesma classe (Marsland, 2009).

3.6.1.1. K-Means

Proposto por Lloyd (1982); MacQueen et al. (1967), o K-Means realiza a classificação de uma amostra através da relação entre a distância Euclidiana (equação 1) do centro de um cluster existente e cada ponto da amostra.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

Por se tratar de um algoritmo iterativo, a cada ponto da amostra que é classificado, o K-Means realiza a atualização da posição de todos os centros de clusters existentes. Sendo o centro do cluster representado pela média de todos os pontos existentes no cluster, a nova posição deste centro pode ser calculada através da equação $\mu_k = \frac{\sum_{i=1}^{n_k} x_k}{n_k}$.

Uma vez que cada ponto pode ser relacionado com todos os centros de clusters existentes, o K-Means classifica um novo ponto não incluso na amostra inicial como pertencente ao cluster que possui a menor distância Euclidiana, como na equação $d_p = \min_k (x_p, \mu_k)$.

Uma das dificuldades na utilização do K-Means é a escolha da quantidade de núcleos k necessária para a realização da classificação e que deve ser definida previamente. Apesar de ser um valor necessário para a utilização do algoritmo, não há uma maneira empírica de calcular o valor de k que garanta a melhor classificação possível.

Desta forma, é possível obter grupos vazios, nos quais não há nenhum elemento, ou grupos que poderiam ser classificados como apenas um, mas que pela quantidade de núcleos foram divididos em dois ou mais. Da mesma maneira, um valor de k baixo pode fazer com que grupos que deveriam ser separados sejam classificados como um grupo único.

Entretanto, utilizar o K-Means oferecido pelo Scikit-Learn é simples, necessitando apenas de uma instância do objeto encontrado em `sklearn.cluster.KMeans`, para o qual pode-se passar o número de núcleos (k) que se deseja encontrar (por padrão, a biblioteca assume 8 clusters). Para realizar o aprendizado sobre a amostra obtida, utiliza-se o método `fit(amostra)`, em que se passa os valores de amostra medida sem os resultados da classificação (quando conhecidos). Para obter o grupo em que um novo

ponto pode ser classificado, utiliza-se o método `predict()`, passando como parâmetro o(s) ponto(s) que se deseja classificar.

A seguir, encontra-se um exemplo de utilização do K-Means oferecido pelo Scikit-Learn. Na seção 3.6.3 são apresentados exemplos da utilização destes métodos para a classificação dos dados, juntamente com os outros métodos que serão apresentados ao longo do texto, bem como uma forma de carregar a amostra armazenada em um arquivo texto para utilizar nos algoritmos demonstrados.

```
#importa o K-Means implementado pelo Scikit-Learn
from sklearn.cluster import KMeans

#instancia um objeto do K-Means utilizando 3 núcleos (k=3)
kmeans = KMeans(k=3)

#considerando que a variável "amostra" possui os valores
#medidos, realiza-se o treinamento utilizando o método fit
kmeans.fit(amostra)

#classificação de um novo ponto P não existente na amostra
#inicial utilizada para realizar o treinamento
kmeans.predict(P)

#um método interessante disponibilizado pelo Scikit-Learn
#no K-Means é o fit_predict, que realiza o treinamento
#utilizando a amostra informada e retorna o resultado da
#classificação nela realizada
kmeans.fit_predict(amostra)
```

3.6.1.2. DBSCAN

Uma outra maneira de classificar os dados obtidos é a partir da densidade dos pontos da amostra, em que é possível considerar que existe uma densidade maior de pontos dentro do cluster do que fora dele (Ester et al., 1996). Entretanto, é possível que exista ruído fora do cluster, interferindo no treinamento do algoritmo de clusterização. Para tentar resolver este problema, Ester et al. (1996) propuseram o *Density Based Spatial Clustering of Applications with Noise* (DBSCAN).

Ao contrário do K-Means, o DBSCAN não precisa que o número k de clusters seja definido previamente, pois este definirá a quantidade de grupos em que a amostra será dividida à medida que itera sobre os dados utilizados para a classificação. Entretanto, duas outras informações devem ser estimadas previamente para que seja possível realizar a classificação com o DBSCAN: *Eps* e *MinPts*.

O valor de *Eps* indica a distância máxima que um ponto pode estar para ser considerado do mesmo grupo, e *MinPts* (nomeado de `min_samples` no algoritmo imple-

mentado no Scikit-Learn) o número mínimo de pontos na vizinhança para que o dado atual possa ser considerado o centro do grupo (*core point*).

A seguir encontra-se um exemplo classificando a mesma base utilizada no exemplo do K-Means, porém utilizando o DBSCAN.

```
#importa o DBSCAN implementado pelo Scikit-Learn
from sklearn.cluster import DBSCAN
#distância euclidiana disponível no Scikit-Learn
from sklearn.metrics import euclidean_distances as \
    euclidean

#instancia um objeto do DBSCAN
dbscan = DBSCAN()

#considerando que os dados estão armazenados na
#variável "amostra", calcula distância entre os
#valores da amostra
damostra = euclidean(amostra)

#realiza-se o treinamento utilizando o método fit
#usando Eps e MinPts padrões da classe
dbscan.fit(damostra)

#exibe resultado da classificação
dbscan.labels_
```

3.6.1.3. Affinity Propagation

A terceira e última técnica de clusterização que será apresentada neste trabalho é conhecida como *Affinity Propagation*, proposta por Frey and Dueck (2007).

Da mesma forma que o DBSCAN, o *Affinity Propagation* não precisa ser informado do número de núcleos que se deseja formar sobre a base de dados que está sendo classificada. Porém, ao contrário dos dois métodos apresentados anteriormente, o *Affinity Propagation* utiliza uma matriz de similaridade (afinidade) calculada para cada par de pontos pertencentes à amostra para realizar a classificação, não utilizando os valores originais pertencentes à amostra.

Uma das maneiras de calcular a matriz de similaridade necessária para realizar a classificação com o *Affinity Propagation* é utilizando a distância Euclidiana, já apresentada na equação 1.

Uma vez que a afinidade entre os pontos tenha sido calculada, o *Affinity Propagation* itera sobre os pontos em busca do *exemplar* (que o centro do grupo encontrado pelo *Affinity Propagation*) e então marca a distância entre o *exemplar* e os outros pontos que pertencem ao grupo.

Assim como apresentado para o DBSCAN, o exemplo a seguir introduz uma forma de utilizar o *Affinity Propagation* a partir dos mesmos dados utilizados para realizar a classificação com o K-Means. Neste caso, foi utilizada a distância Euclidiana para calcular a matriz de afinidade.

```
#importa o Affinity Propagation do Scikit-Learn
from sklearn.cluster import AffinityPropagation
#importa o cálculo da distância euclidiana
from sklearn.metrics import euclidean_distances as \
    euclidean

#instancia um objeto do Affinity Propagation
ap = AffinityPropagation()

#calcula-se a distância euclidiana entre os
#pontos existentes na amostra
damostra = euclidean(amostra)

#realiza-se o treinamento utilizando o metodo fit
#com os dados contidos na variável amostra "damostra"
ap.fit(damostra)
```

A principal função do Scikit-Learn é oferecer uma vasta biblioteca de algoritmos de aprendizado de máquina que sejam fáceis de utilizar. Entretanto, o Scikit-Learn também provê métodos para transformação dos dados, para que se torne mais fácil trabalhar com eles.

3.6.2. Transformação de Dados

Em todos os algoritmos disponibilizados pelo Scikit-Learn, é necessário que certas informações lhes sejam passadas para que o aprendizado possa ser feito. Em alguns desses casos, é importante que os dados sejam transformados (normalizados, binarizados ou tenham as médias subtraídas) para que o resultado obtido se torne mais simples de ser analisado. Para realizar essas tarefas, o Scikit-Learn oferece algumas funções dentro de seu módulo `preprocessing`.

Assim como os algoritmos de aprendizado, o módulo de pré-processamento de dados oferece objetos e funções. A principal diferença entre os dois métodos de pré-processamento é o objetivo final de cada um: enquanto as funções oferecem uma forma rápida de transformar os dados, os objetos retêm as informações da transformação inicial, permitindo que novos dados possam ser pré-processados mantendo as mesmas métricas que a transformação inicial.

Os três métodos existentes no Scikit-Learn que serão apresentados são o `scale` (que centra os dados na média com um valor de variância unitário, o `normalize` que realiza a normalização dos dados e os apresenta com valores entre 0.0 e 1.0. O último método apresentado é o `binarize`, que recebe dois parâmetros: as amostras que se deseja transformar e o valor de limiar que será utilizado para dividir os dados entre 0 e 1.

Assim como nas seções anteriores, o exemplo a seguir realiza as transformações na mesma base de dados utilizada anteriormente.

```
#importa biblioteca de pré-processamento
from sklearn import preprocessing
from numpy import mean

#considerando que os dados desejados estão armazenados
#na variável "amostra"

#realiza a normalização
norm = preprocessing.normalize(amostra)

#realiza o escalonamento
scale = preprocessing.scale(amostra)

#realiza a binarização
binarizado = preprocessing.binarize(amostra, mean(amostra))
```

Utilizando as funções como mostrado no exemplo anterior, tem-se os resultados armazenados nas variáveis `norm`, `scale` e `bin`. Porém, se for desejado que uma nova amostra seja transformada da mesma forma que a anterior, é necessário que se utilize objetos no lugar nas funções.

Assim como os algoritmos apresentados anteriormente, as classes de pré-processamento de dados são baseadas na classe `Transformer`, e implementam dois métodos: `fit()` e `transform()`. A seguir, será realizado o mesmo pré-processamento que no exemplo anterior, porém uma nova amostra também será transformada após o pré-processamento da primeira.

```
#importa biblioteca de pré-processamento
from sklearn import preprocessing
from numpy import mean # para cálculo da média

#considerando que os valores com os quais se
#deseja realizar o treinamento estão armazenados
#na variável "amostra"

#realiza a normalização
normalizer = preprocessing.Normalizer()
normalizer.fit(amostra)

#realiza o escalonamento
scaler = preprocessingScaler()
scaler.fit(amostra)
```

```
#realiza a binarização
binarizer = preprocessing.Binarizer(mean(amostra))
binarizer.fit(amostra)

#pré-processamento de uma segunda amostra a partir da
#transformação realizada na primeira

#considerando que os novos dados foram carregados
#na variável "amostra2"

#normalização da segunda amostra a partir da primeira
norm = normalizer.transform(amostra2)

#escalonamento da segunda amostra a partir da primeira
scale = scaler.transform(amostra2)

#binarização da segunda amostra a partir da primeira
binarizado = binarizer.transform(amostra2)

#assim como o K-Means, os objetos de pré-processamento
#apresentam o método fit_transform, que
#transforma os dados e retorna o resultado
norm1 = normalizer.fit_transform(amostra)
scale1 = scaler.fit_transform(amostra)
bin1 = binarizer.fit_transform(amostra)
```

Nesta seção foram apresentados alguns algoritmos de clusterização disponibilizados pelo Scikit-Learn juntamente com o módulo de pré-processamento. A seguir, será apresentado um exemplo, utilizando a mesma base de dados, que reúne todas as técnicas que foram apresentadas.

3.6.3. Exemplo de Utilização do Scikit-Learn

Considerando que os dados referentes à amostra foram carregados na variável `amostra`, o exemplo a seguir realiza o treinamento dos algoritmos de classificação e a previsão de alguns pontos utilizando os métodos descritos anteriormente, juntamente com o pré-processamento dos dados da amostra.

```
#importa os módulos necessários
from sklearn import preprocessing # normalização
#algoritmos de clusterização apresentados
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.cluster import AffinityPropagation
#importa o cálculo da distância euclidiana
```

```
from sklearn.metrics import euclidean_distances as \
euclidean

#carrega os dados do arquivo 'data.txt' e formata
f = open('data.txt')
temp = f.readlines()
f.close()

amostra = []
for i in temp:
    amostra.append(i.split())
amostra = [[float(i), float(j)] for i, j in amostra]

#realiza a normalização na amostra
norm = preprocessing.Normalizer()
amostra_norm = norm.fit_transform(amostra)
damostra = euclidean(amostra_norm)

#instancia objetos para classificação
kmeans = KMeans(k=3)
dbscan = DBSCAN()
ap = AffinityPropagation()

#treinamento dos algoritmos de classificação
kmeans.fit(amostra_norm)
dbscan.fit(damostranorm)
ap.fit(damostra)

#resultado da classificação
kmeans.predict(amostra_norm)
dbscan.labels_

#prevê os pontos de uma segunda amostra
#pré-processada a partir da primeira amostra

#carrega os dados do arquivo 'data_110.txt' e formata
f = open('data.txt')
temp = f.readlines()
f.close()

amostra2 = []
for i in temp:
    amostra2.append(i.split())
amostra2 = [[float(i), float(j)] for i, j in amostra2]

#normalização da segunda amostra
```

```
amostra2_norm = norm.transform(amostra2)

#classifica nova amostra a partir do aprendizado
#feito na primeira amostra
kmeans.predict(amostra2_norm)
```

Nesta seção foram apresentadas algumas técnicas de classificação e pré-processamento de dados, porém tão importante quanto a utilização dos algoritmos é a visualização dos dados e resultados. Na próxima seção será apresentada a biblioteca Matplotlib, que permite a geração de diversas formas de gráficos, inclusive de animações.

3.7. Personas

Persona é uma técnica que ajuda a entender e aplicar o perfil de usuários no processo de criação de produtos eficientes e efetivos. Na psicologia, Jung (1975) definiu personas na psicologia como a aparência que as pessoas assumem de acordo com suas convenções coletivas. Definiu que persona é a forma como o indivíduo se apresenta ao mundo, expondo seu caráter no relacionamento entre indivíduos, sendo que uma pessoa pode assumir várias personas, de acordo com seu papel. Declara personas como arquétipos da conformidade (Jung and Rocha, 1990; Jung and do Amaral, 1986).

O conceito de personas foi popularizado por Cooper (1999), que definiu personas como: “Personas não são pessoas reais... são arquétipos hipotéticos dos usuários atuais... definidos com rigor e precisão significativos”. Cooper et al. (2007) se referiu a personas como arquétipos hipotéticos porque não há como provar que as personas representam realmente os usuários atuais antes do lançamento de um produto.

Personas também foram são definidas como modelos descritivos de usuários que representam um grupo de pessoas reais e suas características (Aquino Junior and Filgueiras, 2005). Esses modelos agrupam informações sobre conhecimento, habilidades, motivações e preocupações, descrevendo os padrões de ação que um grupo de usuários tem sobre um determinado projeto, permitindo a troca de informações entre os perfis de usuário e a equipe do projeto (Pruitt and Grudin, 2003). Personas podem servir como ferramentas de apoio para documentação de soluções de projeto de interface, armazenando e organizando a representação de usuários (Aquino Junior, 2008). Neste mesmo sentido, Bagnall (2007) descreve a técnica de Personas como uma poderosa ferramenta de design e troca de informações auxiliando a equipe de criação de sistemas interativos, tanto na área de marketing quanto nas equipes de engenharia do conhecimento.

Neste tutorial definimos as personas como modelos criados para representar pessoas reconhecidas por algum processo de estudo do perfil de usuários alvos de um produto ou serviço. As personas ajudam a estabelecer um consenso entre os projetistas e *stakeholders* sobre as necessidades e motivações do público-alvo. Esta técnica produz um conjunto de personagens fictícios ou representações reais identificadas de acordo com as definições iniciais do projeto.

Esses personagens possuem nome e descrições detalhadas sobre suas preferências, comportamentos, expectativas, objetivos, cenários de uso associados, histórias ilus-

trativas, e qualquer informação adicional que ajude o projetista a compreender melhor esse segmento de usuário. Diversos trabalhos demonstram formatos diversificados de representação descritiva das personas em diversas áreas de aplicação, como por exemplo, governo eletrônico (Filgueiras et al., 2005), mercado de celulares (Cirelli et al., 2010), educação (Aquino Junior, 2010, 2012), padrões de interface (Aquino Junior, 2008; Aquino Junior and Filgueiras, 2008), entre outros.

A divulgação das personas pode ocorrer por publicação dos personagens em sites, blogs, cartões de visita, pôster ou qualquer meio que facilite o acesso à informação, tornando o conjunto de personas mais próximo do dia a dia da equipe de projeto e desenvolvimento. De acordo com Cooper (1999), “um dos benefícios mais visíveis das personas é prover uma face humana que coloca o foco da empatia nas pessoas representadas por dados demográficos”.

Existem várias maneiras de criar personas. Pruitt and Adlin (2006) apresentaram um ciclo de vida para criação, uso e descarte de personas. Trata-se de um ciclo completo de aplicação de personas no desenvolvimento de produtos. Neste trabalho, considera-se apenas um processo de concepção de personas, com o objetivo de evidenciar a vantagem de integração do Python científico na modelagem de usuários.

Desta maneira, as seguintes fases são consideradas para a concepção de personas: “Planejamento”, “Pesquisa”, “Criação e Modelagem”.

a) Fase de Planejamento:

No planejamento devem-se compor as suposições, as condições e a busca por informações. Define-se o conjunto de suposições possíveis ou não sobre os usuários, e da quais se tirarão as consequências a serem verificadas na fase de pesquisa. Define-se o conjunto de condições que se toma como ponto de partida para desenvolver o raciocínio com base no problema (projeto). Busca-se o maior número de informações possíveis dos envolvidos com o produto alvo. Podem ser informações provenientes de campanhas publicitárias, banco de dados de clientes, registros de suporte, etc.

Com as informações de planejamento defina: (a) o que as pessoas fazem antes, durante e depois de realizarem as tarefas?; (b) esboce os tipos de papéis que as pessoas assumem para realizar as tarefas; (c) suponha o que essas pessoas gostam de fazer, quais são seus hábitos e preferências; (d) agrupe perfis fictícios com as informações produzidas até esse momento; (e) valide com o cliente quais são os grupos mais importantes de acordo com a estratégia da empresa para esse projeto.

b) Fase de Pesquisa:

Deve-se aplicar iniciativas para coleta de dados com base nas definições da fase anterior. A pesquisa pode ser qualitativa ou quantitativa. A pesquisa qualitativa tem caráter exploratório, estimulando os entrevistados a se expressarem livremente sobre assuntos, conceitos e objetivos. Trata-se de uma pesquisa que auxilia a busca por justificativas para questões em geral, apoiando a interpretação de hipóteses. A amostra da pesquisa qualitativa geralmente é pequena, pois não há preocupação em mapear uma população. A pesquisa qualitativa pode ser executada com discussões em grupo, análise de documentos, mesa-redonda, entrevistas de profundidade, questionário baseado em roteiros.

A pesquisa quantitativa busca a representação numérica de informações pertencentes ao domínio, apurando opiniões e atitudes explícitas dos entrevistados. Essa pesquisa é apoiada por questionários em diversos formatos de mídias. O objetivo é medir e apoiar o teste de hipóteses, pois os resultados são concretos e menos suscetíveis a erros de interpretação. As pesquisas quantitativas permitem estabelecer métricas e índices para comparar os resultados das pesquisas em função do tempo. Pode ser aplicada para informações quantificáveis com inferências a partir de amostras numéricas. A amostra exige um grande número de entrevistados para garantir maior exatidão dos resultados que mapeiam uma população. A pesquisa quantitativa pode ser executada por intermédio de entrevista por atributos (idade, escolaridade, faixa salarial, etc) com questionários padronizados ou análise de *logs* de acesso, ambos possibilitando a preparação de tabelas, quadros e relatórios de uma população. A pesquisa quantitativa permite fazer a análise demográfica, estudando a dinâmica populacional da sociedade ou um grupo específico, considerando as dimensões, as estatísticas, as distribuições e as estruturas da população.

c) Fase de Criação e Modelagem

Para criar personas, procura-se agrupar uma série de comportamentos, motivações e características dos usuários, coletadas através de pesquisas. Desta maneira, com as informações preliminares coletadas na fase de planejamento e os dados coletados durante a fase de pesquisa, é possível identificar padrões de comportamento, de desejos e de necessidades que diferenciam grupos de pessoas. Esse padrão identificado é usado para definição da persona. Esses padrões podem ser extraídos das seguintes variáveis:

- Habilidades: descreve as capacidades do usuário de acordo com o contexto alvo analisado
- Aptidões: descreve os conhecimentos que o usuário possui quanto à tecnologia e tarefa envolvida
- Motivações: descrevem quais são as motivações que o usuário possui com o contexto e produto previsto
- Visão: descreve as opiniões do usuário sobre o produto ou serviço considerado
- Tarefas: descrevem quais são os papéis que o usuário assume no produto ou serviço. A informação é complementada com descrição de “como” as tarefas são executadas
- Demografia: dados que caracterizam o perfil etnográfico do grupo de pessoas, como experiência computacional, escolaridade, idade, sexo, profissão, local de trabalho, dados de moradia, dados culturais, etc.

As descrições das personas podem ter maior ou menor grau de detalhamento, de acordo com as condições de realização do projeto e de acordo com o seu escopo. Mas alguns elementos devem constar das descrições para ajudar os projetistas a tomar decisões sobre o produto ou serviço Aquino Junior (2008). Considera-se que a modelagem de personas deve possuir:

- Nome: objetivo de facilitar a referência ao modelo específico, criando uma associação com pessoas reais e suas características. Algumas equipes preferem aplicar apelidos relacionados à ação do usuário ou considerando traços fortes de sua personalidade;
- Imagem: em conjunto com o nome ajuda a equipe a identificar e memorizar o personagem. Algumas equipes usam imagens de pessoas reais que participaram da pesquisa. Outras usam caricaturas produzidas especialmente para ajudar a lembrar de traços da personalidade da persona;
- Pensamentos e afirmações: apresenta pensamentos expressados pelos usuários reais, assim como afirmações realizadas espontaneamente, ajudando a conhecê-lo quanto a suas atitudes e ações;
- Informações demográficas: a narrativa sobre os atributos etnográficos (culturais) e fatores demográficos como localização (onde mora e trabalha), perfil estatístico (idade, sexo, tamanho da família, renda, ocupação, educação), perfil psicológico genérico (hábitos, características de personalidade, atitude) (vide Figura 3.3 – indicação 1);
- Necessidades: descreve as necessidades que motivam a pessoa no uso do produto ou serviço (vide Figura 3.3 – indicação 2);
- Dificuldades: representa as dificuldades que normalmente esse grupo de pessoas encontra no uso de sistemas interativos. Podem-se descrever os recursos com o qual o usuário interage, desde funcionalidades da interface e de bancos de dados à demanda por determinado conteúdo. Descreve familiaridade de uso de computadores e da internet (vide Figura 3.3 – indicação 3);
- Comportamento: apresenta uma história que aborda o comportamento deste grupo de pessoas ao lidar com pessoas e computadores. Apresentam características afetivas ou sociais comuns a alguns tipos de usuários, motivações que os levam a procurar o produto ou serviço (ocasião de uso, frequência, tipo de dispositivo, necessidade de segurança exigência de declarações legais), linguagem oral, hábitos e preferências. Estes traços proveem a textura visual, emocional e sensorial em que a persona atua (vide Figura 3.3 – indicação 4).

A figura 3.3 apresenta um exemplo de persona usando a estrutura acima. Essa persona foi definida no contexto de governo-eletrônico Aquino Junior (2008).

As personas podem ser apoiadas por informações que complementam o conhecimento a ser aplicado. A descrição de cenários de uso, baseados em tarefas, apoia o entendimento de como um grupo de usuários alcança seus objetivos em interfaces diversificadas ou resolvem problemas na execução de seus papéis.

Com o conjunto de personas definido é possível realizar o alinhamento final com os *stakeholders* do produto e serviço encomendado, com objetivo de classificar a persona principal e a persona secundária. A persona principal representa o público prioritário, enquanto que a persona secundária considera os perfis que também usam o produto ou

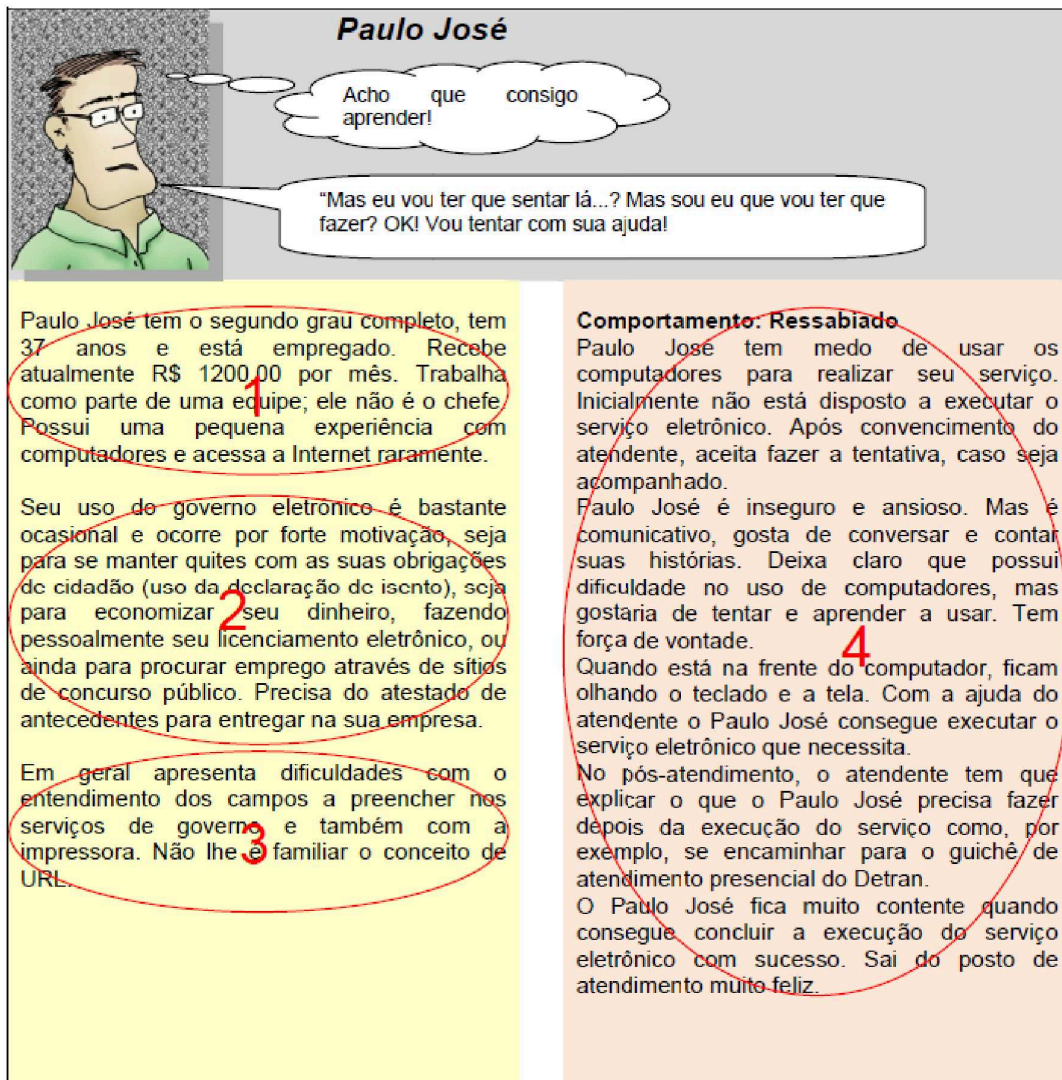


Figura 3.3: Persona definida no contexto de governo-eletrônico Aquino Junior (2008)

serviço, porém suas necessidades não são priorizadas quando sobrepõem requisitos da persona principal. As personas ainda podem ser classificadas em perfis de baixa importância (usam o produto ou serviço esporadicamente), perfis não considerados (não fazem parte do público-alvo) e perfis produzidos (representam pessoas importantes que não foram identificadas na fase de pesquisa, mas identificadas e requisitadas pelos *stakeholders*).

Deve-se ressaltar que a classificação das personas depende do tipo de pesquisa realizado. Quando as personas são criadas apenas por pesquisa qualitativa, normalmente é mais difícil classificar grupos de personas, pois depende apenas da opinião dos *stakeholders*. Quando os projetistas possuem informações provenientes de pesquisa quantitativa é possível identificar prioridades entre personas, com decisões apoiadas por dados estatísticos.

As personas podem apresentar relacionamentos. Os relacionamentos entre perso-

nas podem simplesmente ficar indicados na descrição de cada uma, ou serem apresentados em mapas e diagramas que estabeleçam a ligação de cada um e indiquem as motivações ou preferências com relação a algumas funcionalidades.

A seleção e discriminação de grupos de usuários, bem como a sua tipificação e criação de padrões pode gerar mecanismos arbitrários que reflitam os valores dos agentes envolvidos na sua especificação. Eventualmente pode ser necessário remanejar os resultados para atender as estratégias institucionais ou as visões sobre o alcance do produto ou serviço.

As personas são úteis em todo o ciclo de vida de criação de produtos interativos. As personas fornecem informações desde a concepção do produto/interface até a seleção de perfis para testes de usabilidade. Personas servem de base para os roteiros dos testes de usabilidade com usuários verdadeiros, que a rigor verificam a aderência das narrativas fictícias às situações de uso reais.

3.8. Criando Personas

Agora que o Python, os módulos com algoritmos e o modelo de usuário Personas foram apresentados, será demonstrado um processo completo para criação de Personas. Dessa maneira, pretende-se deixar melhor compreendido o conteúdo apresentado ao longo deste capítulo.

Como exemplo, será utilizado um questionário de pesquisa que contém algumas perguntas sobre a satisfação do usuário relativa a um produto. O questionário completo não será utilizado nessa seção, pois ele é muito extenso; apenas algumas questões serão utilizadas. O questionário é apresentado na sequência do texto.

1. Qual é a sua idade?
2. Você realizou a tarefa de forma fácil e rápida?
 - (a) Concordo totalmente
 - (b) Concordo em parte
 - (c) Discordo em parte
 - (d) Discordo totalmente
3. Você utilizaria o sistema frequentemente?
 - (a) Concordo totalmente
 - (b) Concordo em parte
 - (c) Discordo em parte
 - (d) Discordo totalmente
4. O sistema atende as suas necessidades?
 - (a) Concordo totalmente
 - (b) Concordo em parte

- (c) Discordo em parte
- (d) Discordo totalmente

Com base no questionário acima serão criadas as Personas. Em primeiro lugar precisa-se analisar quais são os tipos das variáveis e quais informações elas trazem. Quando existem variáveis categóricas e alternativas, como no caso das perguntas 2, 3 e 4, deve-se analisar se possui razão entre as respostas, ou seja, há uma hierarquia nas possibilidades de respostas (Hair, 2007).

No exemplo utilizado existem quatro variáveis, uma numérica e contínua, a idade, e três variáveis categóricas que estão divididas em quatro opções de resposta. Nas três variáveis categóricas pode-se notar uma certa hierarquia, ou seja, possuem razão entre as respostas, então fica mais fácil para transformá-las em números, pois o algoritmo trabalha apenas com números. Para transformar as três variáveis em números, como existe razão, será utilizado apenas o método de codificar as repostas em números. Dessa forma as respostas ficam assim:

1. Concordo totalmente
2. Concordo em parte
3. Discordo em parte
4. Discordo totalmente

Codificadas as repostas, há necessidade de carregar as informações dos usuários para uma variável na memória, para que assim possa ser realizada a análise dos dados.

```
#Carregar o arquivo para a memória
arq = open('../BD/persona_exemplo.txt', 'r')

#Criando a matriz de usuários
usuarios = []

#Separando as variáveis em suas linhas e colunas
#corretamente.
for linha in arq.read().split('\r'):
    usuarios.append(linha.split('\t'))

#Convertendo as variáveis de string para número
listaUsuarios = [[float(i[0]), float(i[1]), float(i[2]), \
                    float(i[3])] for i in usuarios]
```

Com os dados carregados na memória, inicia-se o primeiro passo da análise. Existem quatro variáveis: uma pode conter números entre 0 e ∞ , que é a idade, as outras contêm números entre 1 e 4, que são as possibilidades de respostas. Essa diferença de escala pode gerar uma tendência ao algoritmo e diminuir o desempenho de seu resultado.

Para resolver esse problema realiza-se uma normalização dos dados, geralmente entre 0 e 1 (Lattin et al., 2011), mantendo-os dentro de uma mesma escala. A normalização realizada nesse exemplo é apresentada na equação 2.

$$X_{norm} = \frac{\min(X) - X}{\max(X)} \quad (2)$$

Antes de normalizar os dados é necessário verificar como está a distribuição dos valores para não tornar os valores tendenciosos, por exemplo, no caso da idade temos uma concentração de valores entre 15 e 17 anos, porém existe um usuário que tem 58 anos. Se a normalização for realizada considerando como 58 o valor máximo, os usuários que possuem 15 e 17 anos terão o desvio padrão praticamente anulado e serão considerados iguais ao ponto de vista do algoritmo.

Assim, precisa-se definir os melhores valores de máximo e mínimo para cada variável de tal forma, que a distribuição das variáveis fique sem ruídos. Dessa forma, com o auxílio do desvio padrão e a média dos dados pode-se verificar se essa distribuição está adequada ou não para a normalização. Baseando-se na estatística, para a distribuição dos dados ser considerada sem ruído o desvio padrão deve ser menor a 30% da média (McKay , 1932; Verrill , 2003; Hair , 2007). Veja a verificação dessa condição para a idade dos usuários do exemplo.

```
>>> np.std(listaUsuarios[:,0])
6.6755124296150345

>>> np.average(listaUsuarios[:,0])
17.255813953488371

>>> max(listaUsuarios[:,0])
58.0

>>> min(listaUsuarios[:,0])
13.0

>>> 0.3*np.average(listaUsuarios[:,0])
5.1767441860465109

>>> np.std(listaUsuarios[:,0]) < 0.3 * \
      np.average(listaUsuarios[:,0])
False
```

Observa-se que a idade da base de dados utilizada não atende a condição para normalização. Assim, deve-se excluir o valor que está causando o ruído. Para isso, considera-se qualquer valor acima do máximo definido como o valor máximo. Na base de dados do exemplo, basta remover a idade de 58 anos, tornando como o máximo o valor de 27 anos (segundo maior da amostra).

Para as demais variáveis não há a necessidade de alterar o mínimo e o máximo devido ao significado perante o que se espera como resultado. Sendo assim, normaliza-se as informações considerando 13 e 27 como os mínimo e máximo da idade e 1 e 4 para as demais variáveis. A função de normalização fica assim:

```
#Normalizando as informações dos usuários
for usuario in listaUsuarios:
    #Normalizando a variável idade do usuário
    if usuario[0] <= 13:
        usuario[0] = 0
    elif usuario[0] >= 27:
        usuario[0] = 1
    else:
        usuario[0] = abs(13 - usuario[0]) / 27
    #Normalizando a variável tarefa fácil e rápida
    if usuario[1] <= 1:
        usuario[1] = 0
    elif usuario[1] >= 4:
        usuario[1] = 1
    else:
        usuario[1] = abs(1 - usuario[1]) / 4
    #Normalizando a variável utilizaria frequentemente
    if usuario[2] <= 1:
        usuario[2] = 0
    elif usuario[2] >= 4:
        usuario[2] = 1
    else:
        usuario[2] = abs(1 - usuario[2]) / 4
    #Normalizando a variável atende necessidades
    if usuario[3] <= 1:
        usuario[3] = 0
    elif usuario[3] >= 4:
        usuario[3] = 1
    else:
        usuario[3] = abs(1 - usuario[3]) / 4
```

Com os dados normalizados, pode-se aplicar um dos algoritmos de clusterização apresentados na seção 3.6.3. Para este exemplo, utilizaremos o DBScan, que tem demonstrado um melhor resultado, sem depender de informar o número de grupos que se deseja.

```
#Executando o DBScan
S = np.vstack(listaUsuarios)
core_samples, labels = dbscan(S, eps=0.6, min_samples=1, \
                               metric='euclidean', random_state=None)
```

O DBScan trabalha com a distância euclidiana entre os usuários para determinar se estão ou não em um mesmo grupo. O valor de distância mínima (eps) considerado nesse exemplo foi de 0.6 e o número mínimo para considerar um grupo foi de 1 elemento, para não se excluir ninguém. Ao executar o algoritmo, ele retorna na variável `labels` o valor do índice do grupo ao qual o usuário pertence.

Após esse passo precisamos separar os elementos em seus grupos para gerar as Personas. Para criar os grupos basta realizar o seguinte código.

```
#Criando uma lista de grupos
grupos = []

#Gerando os grupos
for i in set(labels):
    grupos.append([index for index in core_samples \
                   if labels[index] == i])
```

Grupos formados, este é o último passo para criarmos as Personas, que consiste simplesmente de combinar os valores das variáveis e encontrar um valor em comum entre elas. Conforme apresentado por Masiero et al. (2011), para encontrar o valor em comum pode-se utilizar três métodos: a média, a moda ou a mediana. Como já foi realizada a análise da distribuição dos dados antes a normalização, a média deve ser descartada devido ao ruído encontrado na amostra, deixando o desvio padrão maior do que 30% da média. Nesse caso, é indicado que se utilize da mediana ou moda. Para finalizar o exemplo, será utilizada a mediana, conforme ilustrado no código.

```
#Criando uma lista para as personas
personas = []

#Construindo as personas
for grupo in grupos:
    temp = []
    for membro in grupo:
        temp.append(listaUsuarios[membro])
    temp = np.array(temp)
    personas.append([np.median(temp[:,0]), \
                    np.median(temp[:,1]), \
                    np.median(temp[:,2]), \
                    np.median(temp[:,3])])
```

Assim, foram construídas as personas ao longo do processo apresentado nesta seção. As personas construídas nesse exemplo podem ser conferidas na tabela 3.4.

Tabela 3.4: Personas construídas como exemplo

Idade	Frequência de Utilização	Facilidade	Necessidades
16	Concordo em parte	Concordo em parte	Concordo em parte
27	Discordo totalmente	Discordo totalmente	Discordo em parte
58	Concordo em parte	Discordo totalmente	Concordo em parte

3.9. Matplotlib – Gerando Gráficos com Python

A visualização dos dados e dos resultados obtidos através da classificação é importante para a análise dos resultados. Utilizando os exemplos da classificação do K-Means, serão gerados gráficos para a visualização dos dados utilizando o Matplotlib.

O Matplotlib inicialmente desenvolvido por Hunter (2007) tem como objetivo a geração de gráficos em diversos formatos (inclusive vídeos com gráficos animados) com qualidade para publicações em diversas plataformas.

Sendo uma das bibliotecas de Python mais utilizadas para a geração de gráficos, o Matplotlib segue os conceitos de facilidade que são fundamentais para o Python, sendo ainda familiar aos usuários de MATLAB, facilitando o aprendizado e a utilização dos recursos disponíveis para usuários de diversas áreas.

Nesta seção serão apresentadas algumas formas de geração de gráficos disponíveis no Matplotlib.

3.9.1. Geração de Gráficos

Dos diversos tipos de gráficos existentes, pode-se citar o de dispersão, de linha e histograma como alguns dos fundamentais e largamente utilizados. No Matplotlib, esses três tipos de gráficos estão reunidos dentro do objeto `pyplot` que, segundo a descrição apresentada no site da biblioteca,⁵ fornece um *framework* para geração de gráficos parecido com as funções do MATLAB.

A seguir, será apresentado um exemplo que, a partir dos dados gerados na seção anterior, plota os gráficos de linha, dispersão e histograma, exhibe o gráfico em uma janela e os salva como imagens, para uso posterior.

```
#importa o K-Means implementado pelo Scikit-Learn
from sklearn.cluster import KMeans
from sklearn import preprocessing # normalização
#funções para gerar os gráficos
from matplotlib import pyplot as plt

#instancia um objeto do K-Means utilizando 3 núcleos (k=3)
kmeans = KMeans(k=3)

#considerando que os dados foram carregados e
#formatados corretamente e que encontram-se
```

⁵http://matplotlib.sourceforge.net/api/pyplot_api.html

```
#na variável "amostra"

#realiza a classificação da amostra e
#armazena o resultado para plotagem
resultados = kmeans.fit_predict(amostra)

X = amostra.T[0]
Y = amostra.T[1]

#plota os dados em um gráfico de dispersão
plt.scatter(X, Y) # cria o gráfico de dispersão
plt.xlabel('Variavel 1') # insere nome do eixo X
plt.ylabel('Variavel 2') # insere nome do eixo Y
plt.savefig('dispersao.png') # salva o gráfico como figura
plt.show() # exibe uma janela com o gráfico

#mesmos dados em um gráfico de linha
plt.figure() # nova figura para não sobrepor o gráfico
plt.plot(X, 'b', Y, 'r') # cria o gráfico de linha
#os parâmetros para cor e tipo de linha seguem
#o mesmo padrão que o MATLAB
plt.xlabel('Número da variável')
plt.ylabel('Valor da variável na amostra')
plt.legend(('variável 1', 'variável 2'))
plt.savefig('linha.png')
plt.show()

#o resultado da classificação plotado com histograma
plt.figure() # cria uma terceira figura
#plota o histograma com o resultado da classificação
plt.hist(resultados)
plt.savefig('histograma.png')
plt.show()
```

Os resultados do código anterior encontram-se na figura 3.4.

Entretanto, em alguns momentos se torna mais fácil analisar o gráfico quando plotado em conjunto com os outros gráficos. O exemplo a seguir apresenta a divisão do gráfico de linha da figura 3.4b em dois gráficos plotados na mesma janela. O resultado deste exemplo pode ser visualizado na figura 3.5.

```
#importa o K-Means implementado pelo Scikit-Learn
from sklearn.cluster import KMeans
from sklearn import preprocessing # normalização
from matplotlib import pyplot as plt
```

```

#instancia um objeto do K-Means utilizando 3 núcleos (k=3)
kmeans = KMeans(k=3)

#considerando que os dados foram carregados e
#formatados corretamente e que encontram-se
#na variável "amostra"

#realiza a classificação da amostra e
#armazena o resultado para plotagem
resultados = kmeans.fit_predict(amostra)

X = amostra.T[0]
Y = amostra.T[1]

#gera um gráfico que aceita subplots
plt.subplot(121) # 2 posições na horizontal e 1 vertical
plt.plot(X) # plota o gráfico na primeira posição
plt.subplot(122) # escolhe a segunda posição
plt.plot(Y, 'r') # plota o segundo gráfico
plt.savefig('subplot.png') # salva os gráficos
plt.show() # exibe os gráficos em apenas uma janela

```

Além destes três tipos de gráficos, é possível também gerar gráficos em 3D utilizando o Matplotlib. Da mesma forma que o 2D, é necessário utilizar o `pyplot`, além do `Axes3D` contido em `mpl_toolkits.mplot3d`. O exemplo a seguir plota o resultado da classificação em relação às variáveis normalizadas da amostra e a figura 3.6 apresenta o resultado dessa classificação.

```

#importa o K-Means implementado pelo Scikit-Learn
from sklearn.cluster import KMeans
from sklearn import preprocessing # normalização
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D # gráficos em 3D

#instancia um objeto do K-Means utilizando 3 núcleos (k=3)
kmeans = KMeans(k=3)

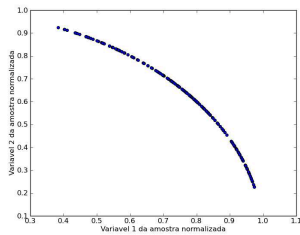
#considerando que os dados foram carregados e
#formatados corretamente e que encontram-se
#na variável "amostra"

#realiza a classificação da amostra e
#armazena o resultado para plotagem
resultados = kmeans.fit_predict(amostra)

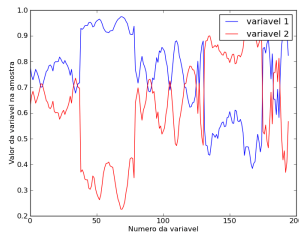
```

```
X = amostra.T[0]
Y = amostra.T[1]

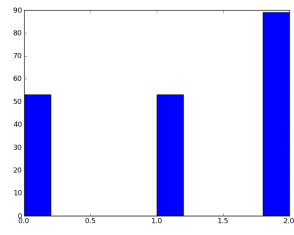
#gera o gráfico 3D com a amostra e classificação
fig = plt.figure() # cria uma nova figura
ax = fig.gca(projection='3d') # para fazer o gráfico 3D
#plota os resultados utilizando dispersão
ax.scatter(X, Y, resultados)
plt.savefig('plot3d.png')
plt.show()
```



(a) Dispersão



(b) Linha



(c) Histograma

Figura 3.4: Resultados do exemplo de gráficos do Matplotlib

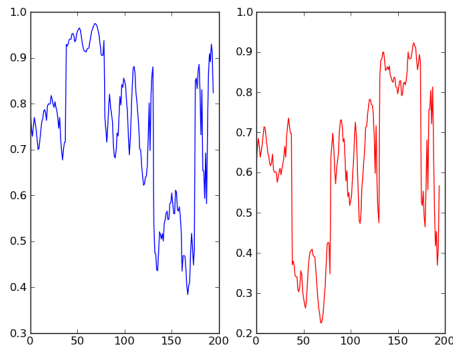


Figura 3.5: Resultado do plotagem de dois gráficos em uma única janela

Os gráficos apresentados até o momento são figuras estáticas que podem ser utilizados para produção de textos e apresentações. Entretanto, o Matplotlib também permite que sejam criadas animações nas quais os dados são plotados ao longo do tempo.

3.9.2. Gráficos Animados

Utilizando o mesmo princípio da geração de gráficos em 2D e 3D, o Matplotlib permite a plotagem de gráficos animados utilizando a biblioteca `matplotlib.animation`.

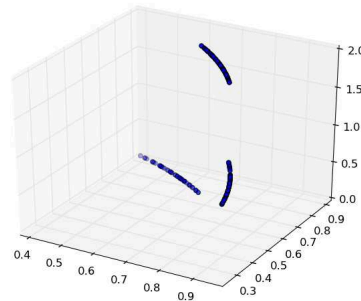


Figura 3.6: Resultado da classificação feita pelo K-Means com $k = 3$

Neste caso, os dados que serão plotados necessitam de atualização a cada passo da animação, de forma que ao longo do tempo o Matplotlib produzirá uma animação que consiste na plotagem sequencial destes dados.

O exemplo a seguir apresenta um código que gera uma animação a partir dos mesmos dados utilizados nos exemplos anteriores e a exportação desse resultado como um vídeo no formato mp4.

```
#importa o K-Means implementado pelo Scikit-Learn
from sklearn import preprocessing # normalização
from matplotlib import pyplot as plt

#a biblioteca animation contém as funções necessárias
#para o gráfico animado
from matplotlib import animation

#considerando que os dados foram carregados e
#formatados corretamente e que encontram-se
#na variável "amostra"

X = amostra.T[0]
Y = amostra.T[1]

#gera o gráfico animado

#função para atualizar os dados que serão plotados
fig = plt.figure()
ax = fig.add_subplot(111)
linha = ax.scatter([], []) # gera uma figura vazia
ax.set_xlim(-1.5, 1.5) # limita os valores em X
ax.set_ylim(-1.5, 1.5) # limita os valores em Y
x, y = [], [] # dados que serão plotados
```

```
indice = 0

def gera_dados():
    cont = 0
    while cont < len(X) - 1:
        cont += 1
        yield X[cont], Y[cont]

def atualiza_dados(data):
    x0, y0 = data # adiciona aos dados que serão plotados
    x.append(x0)
    y.append(y0)

    #determina o conjunto de pontos
    linha = ax.scatter(x, y)

    return linha

ani = animation.FuncAnimation(fig, atualiza_dados,
                              gera_dados, interval=10, repeat=False)

ani.save('animacao.mp4')
plt.show()
```

Nesta seção foram apresentadas algumas funções básicas para a produção de gráficos utilizando o Matplotlib, porém existem diversas outras opções disponíveis no Matplotlib, como gráficos de pizza, apresentação de imagens e gráficos com coordenadas polares e boxplot.

Referências

- Aquino Junior, P. T. (2008) “Picap: padrões e personas para expressão da diversidade de usuários no projeto de interação”, Biblioteca Digital de Teses e Dissertações da USP. <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-15092008-144412/>, Abril
- Aquino Junior, P. T. (2010) “Papéis do docente de ihc: do conhecimento ao mercado”, In: IX Simpósio sobre Fatores Humanos em Sistemas Computacionais, IHC '10, Brazil, Brazilian Computer Society. p. 79–82.
- Aquino Junior, P. T. (2012) “Personas alunos e papéis do docente no atendimento da diversidade de perfis”
- Aquino Junior, P. T. and Filgueiras, L. V. L. (2005) “User modeling with personas”, In: Proceedings of the 2005 Latin American conference on Human-computer interaction, CLIHC '05, USA, ACM, p. 277–282.

- Aquino Junior, P. T. and Filgueiras, L. V. L. (2008) “A expressão da diversidade de usuários no projeto de interação com padrões e personas”, In: Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems, IHC '08, Brazilian Computer Society, Brazil, p. 1–10.
- Bagnall, P. (2007) “Using personas effectively”, In: Proceedings of the 21st British HCI Group Annual Conference on HCI 2008: People and Computers, volume 2.
- Borges, L. E. “Python para desenvolvedores 2ª edição”.
- Cirelli, M.; Aquino, P. T., and Filgueiras, L. V. L. (2010) “Personas celulares do mercado brasileiro”, In: Proceedings of the IX Symposium on Human Factors in Computing Systems, IHC '10, Brazilian Computer Society, Brazil, p. 129–138
- Cooper, A. (1999) “The Inmates Are Running the Asylum”, Macmillan Publishing Co., Inc., 1999.
- Cooper, A.; Reimann, R., and Cronin, D. (2007) “About face 3: the essentials of interaction design”, Wiley-India.
- Deitel, H. M.; Deitel, P. J.; Liperi, J. P., and Weidemann, B. (2002) “Python: how to program”, Prentice Hall.
- Ester, M.; Kriegel, H.; Sander, J., and Xu, X. (1996) “A density-based algorithm for discovering clusters in large spatial databases with noise”, In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining, AAAI Press, p. 226–231.
- Filgueiras, L. V. L.; Aquino Junior, P. T.; Sakai, R.; Filho, A. G.; Torres, C., and Barbarian, I. (2005) “Personas como modelo de usuários de serviços de governo eletrônico”, In: Proceedings of the 2005 Latin American conference on Human-computer interaction, CLIHC '05, USA, ACM, p. 319–324.
- Frey, B. J. and Dueck, D. (2007) “Clustering by passing messages between data points”,
- Hair, J. F. (2007) *Análise multivariada de dados* Bookman
- Hunter, J. D. (2007) “Matplotlib: A 2d graphics environment”, *Computing In Science & Engineering*, p. 90–95
- Jones, E.; Oliphant, T.; Peterson, P., and others, . (2001) “SciPy: Open source scientific tools for Python”, <http://www.scipy.org/>
- Jung, C. G. (1975) “The archetypes and the collective unconscious”, volume 2, Bollingen.
- Jung, C. G. and do Amaral, V. (1986) “O desenvolvimento da personalidade”, volume 7, Brazil, Vozes.
- Jung, C. G. and Rocha, M. R. (1990) “AION-Estudos sobre o simbolismo do si-mesmo”, Brazil, Vozes.

- Lattin, J.; Carrol, D. J. D., and Green, P. E. (2011) “Análise de dados multivariados”, Brazil, Cengage Learning.
- Lloyd, S. P. (1982) “Least squares quantization in pcm”, In: IEEE Transactions on Information Theory 28, p. 129–137
- MacQueen, J. and others, (1967) “Some methods for classification and analysis of multivariate observations”, In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability”, volume 1, USA, p. 281–297.
- Marsland, S. (2009) “Machine Learning: An Algorithmic Introduction”, USA, CRC Press.
- Masiero, A. A.; Leite, M. G.; Filgueiras, L. V. L., and Aquino Junior, P. T. (2011) “Multi-directional knowledge extraction process for creating behavioral personas”, In: Proceedings of the 10th Brazilian Symposium on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction, IHC+CLIHC '11, Brazil, Brazilian Computer Society, p. 91–99.
- McKay, A. T. (1932) “Distribution of the Coefficient of Variation and the Extended t Distribution” Journal of the Royal Statistical Society, JSTOR, p. 695–698
- Pedregosa, F.; Varoquaux, G.; Gramfort, A., and others. (2011) “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research 12
- Pruitt, J. and Adlin, T. (2006) “The persona lifecycle: keeping people in mind throughout product design”, USA, Morgan Kaufmann Publishers.
- Pruitt, J. and Grudin, J. (2003) “Personas: practice and theory”, In: Proceedings of the 2003 conference on Designing for user experiences, DUX '03, USA, ACM, p. 1–15.
- Summerfield, M. (2009) “Programming in Python 3: a complete introduction to the Python language”, USA, Addison-Wesley Professional.
- Verrill, S. P. (2003) “Confidence bounds for normal and lognormal distribution coefficients of variation” US Department of Agriculture, Forest Service, Forest Products Laboratory