Capítulo

6

Introdução à Ciência de Dados: Uma Visão Pragmática utilizando Python, Aplicações e Oportunidades em Redes de Computadores

Giovanni Comarela (UFV), Gabriel Franco (UFV), Celio Trois (UFSM), Alextian Liberato (IFES), Magnos Martinello (UFES), João Henrique Corrêa (UFES) e Rodolfo Villaça (UFES)

Abstract

Internet of Things, smart cities, crowdsourcing, and other neotechnologies are making the world and people more and more connected, causing colossal amounts of information to be transmitted through the network. In order to extract new knowledge and relevant information from these data, Data Science*, which covers several disciplines, such as statistics, data mining, machine learning, and artificial intelligence, is becoming popular and gaining space in the most diverse areas of knowledge. In this context, this course presents, in a pragmatic way, through the Python language and specific libraries, the methods most used by data scientists to manipulate and analyze information, applying them in the resolution of relevant problems in the area of computer networks.*

Resumo

Internet das Coisas, cidades inteligentes, crowdsourcing e outros neotecnologismos estão tornando o mundo e as pessoas cada vez mais conectados, fazendo que quantidades colossais de informação sejam transmitidas pela rede. No intuito de extrair novos conhecimentos e informações relevantes desses dados, a Ciência de Dados*, que abrange várias disciplinas, tais como estatística, mineração de dados, aprendizado de máquina e inteligência artificial, está se popularizando e ganhando espaço nas mais diversas áreas de conhecimento. Nesse contexto, esse minicurso apresenta, de forma pragmática, através da linguagem Python e bibliotecas específicas, os métodos mais usados para manipular e analisar dados e extrair informações, aplicando-os na resolução de problemas relevantes da área de redes de computadores.*

Introdução

Recentemente o termo "Dataist" foi cunhado no vale do silício como uma espécie de religião dos dados. O objetivo é maximizar o fluxo de dados conectando mais e mais mídias, produzindo e consumindo cada vez mais informação; os profetas do "Dataism" conscientemente usam a linguagem tradicional messiânica[1]. Como toda a religião de sucesso, o "Dataism" é também missionário e tem como seu mandamento ligar tudo ao sistema, incluindo os hereges. E tudo significa mais do que apenas conectar humanos, isso quer dizer conectar todas as coisas. Os carros nas ruas, refrigeradores nas cozinhas, e árvores nas florestas - tudo deve estar conectado à Internet-de-todas-as-coisas. Carros vão se comunicar entre si, as árvores na selva vão relatar os níveis de dióxido de carbono. Em tese, não deve-se deixar qualquer parte do universo desconectado da grande rede.

O "Dataism" [Harari, 2016] apoia a liberdade de informação como o maior bem de todos. Entretanto, "Dataists" acreditam que humanos não conseguem lidar com o imenso fluxo de dados, e não distinguem mais dados de informação. O trabalho de processamento de dados deveria assim ser passado aos algoritmos, cuja capacidade excede de longe o cérebro humano. Na prática, isso significa que os "Dataists" são céticos em relação ao processamento de grandes volumes de dados pelo cérebro humano, e preferem confiar em algoritmos. Como exemplo da efetividade dos algoritmos, o Facebook é capaz de apontar com precisão a intenção de votos nas eleições, identificar os eleitores que ainda não se decidiram e determinar o que cada candidato precisa dizer para conquistar mais votos [2].

Como obter esses dados como a matéria-prima dourada? Muitas pessoas proveem dados gratuitamente para ter acesso aos serviços oferecidos na Internet. De fato, a popularidade atingida pela Internet e pela Web nas últimas décadas faz com que uma grande quantidade de dados seja gerada diariamente. Esses dados podem estar relacionados tanto aos usuários da rede (e.g., páginas Web visitadas, produtos comprados, pesquisas em máquinas de busca, amizades em redes sociais e filmes assistidos) quanto a própria rede em si (e.g., utilização e topologia da rede, falhas em roteadores e *links*, disponibilidade e desempenho de serviços e dinâmica de roteamento). De qualquer forma, analisar essa quantidade massiva de dados é essencial para que empresas possam oferecer melhores serviços e, consequentemente, possam sobreviver em um mercado que é tão competitivo.

Esta crescente necessidade de analisar dados fez surgir a demanda por profissionais bem versados em estatística, aprendizado de máquina, mineração de dados, inteligência artificial, econometria e várias outras áreas correlatas. Informalmente[3], tal profissional é referido como *cientista de dados* e a área que trabalha, *ciência de dados*.

A abordagem adotada nesse texto é apresentar um resumo de diversas competências básicas de um profissional da área Ciência de Dados. Outro aspecto considerado é reunir alguns exemplos de trabalhos científicos que mostram a importância da análise de dados em redes de computadores. A partir de exemplos, pretende-se motivar o leitor sobre a importância do tema para a área de redes, para então mostrar que, apesar de complexas, muitas técnicas podem ser utilizadas de forma simples por meio da linguagem de

[1] Por exemplo, no livro de profecias de "The Singularity is Near [Kurzweil, 2010]".

[2] https://bdtechtalks.com/2018/01/31/yuval-harari-wef-ai-big-data-digital-dictatorship/

[3] Apesar de esforços, ainda não há um consenso na academia e indústria, nem mesmo regulamentação governamental, sobre a profissão cientista de dados.

programação Python e várias de suas bibliotecas.

Há bastante tempo, a comunidade acadêmica de redes vem estudando técnicas de manipulação e análise de dados de forma a extrair conhecimento para apoiar a tomada de decisão [Boutaba et al., 2018]. Mais especificamente no contexto do Simpósio Brasileiro de Redes de Computadores, minicursos relacionados, mas não equivalentes, ao tema deste texto e à Ciência de Dados já foram apresentados. Por exemplo, em [Carvalho et al., 2014] foi explorado o advento da e-Ciência e aplicações de *Big Data*, que trouxe consigo a necessidade de se trabalhar com volumes cada vez maiores de dados oriundos de sensores, equipamentos especializados e centros de computação de alto desempenho. Este trabalho visou apresentar os desafios impostos pelas aplicações de e-Ciência com o foco em um modelo de rede denominado DMZ Científica, bem como as tecnologias existentes e futuras para a sua implantação e utilização. Outro minicurso ministrado no contexto de análise de dados foi [Celes et al., 2017], intitulado "Big Data Analytics no Projeto de Redes Móveis: Modelos, Protocolos e Aplicações". Este, por sua vez, investigou a extração de conhecimento a partir dados com informações espaço-temporais de diferentes entidades, tais como pessoas, veículos e objetos no domínio de redes móveis. Nesse sentido, o minicurso proposto usou dados históricos dos usuários para obter características importantes que permitiram detectar padrões e realizar predições, tanto da mobilidade desses usuários como do tráfego de dados no âmbito espaço-temporal.

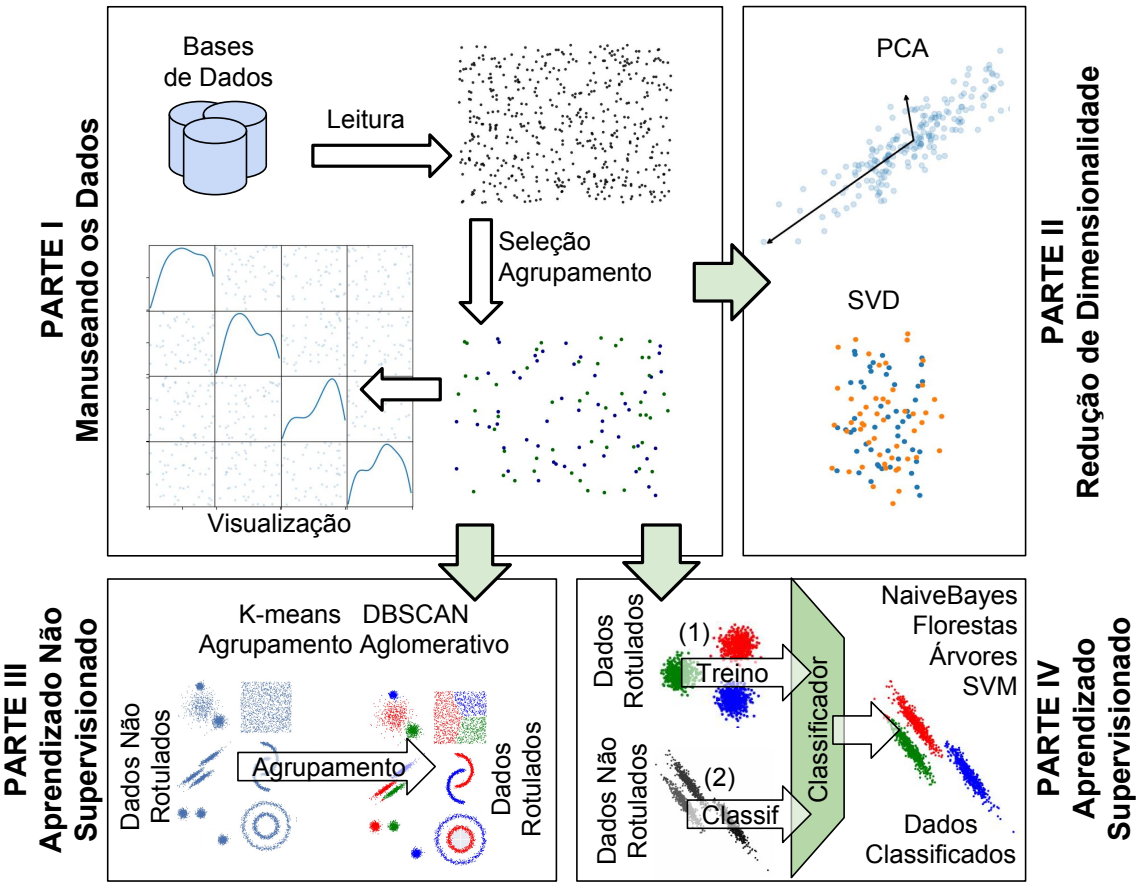


Figura 6.1: Visão geral dos tópicos abordados no minicurso.

O conteúdo deste texto está dividido em quatro partes principais, conforme apresentado na Figura 6.1 e explicado a seguir:

- A Parte I apresenta recursos para **leitura e manipulação de dados** por meio da biblioteca *Pandas* do Python.
- Na Parte II, mostra-se como efetuar **redução de dimensionalidade** via análise de componentes principais e decomposição de valores singulares. Os conceitos apresentados são então utilizados para mostrar como é possível detectar anomalias em matrizes de tráfego, assim como proposto em [Lakhina et al., 2005].
- A Parte III parte tem foco em **aprendizado não supervisionado**, onde são apresentados diferentes paradigmas para realizar agrupamento de dados: K-Means, agrupamento aglomerativo, agrupamento baseado em densidade e agrupamento espectral. Além disso, mostra-se, como exemplo de aplicação, como algumas dessas técnicas podem ser utilizadas para detectar comunidades em redes.
- A parte IV tem como objetivo introduzir algumas **técnicas de classificação**, i.e., aprendizado supervisionado, mostrando, além dos principais classificadores, **metodologias para treino, avaliação, validação e teste** de modelos. Nessa parte, apresenta-se um exemplo referente às comunicações de aplicações científicas de alto desempenho em um centro de dados. São analisadas as imagens das matrizes de tráfego de aplicações científicas, visando identificar, através de aprendizado supervisionado, qual aplicação está sendo executada [Trois et al., 2016, Trois et al., 2018].

PARTE I - Leitura e Manipulação de dados

O primeiro passo para que dados sejam analisados é a sua aquisição. Muitas vezes os dados lidos contém valores indesejáveis, sendo necessário efetuar operações tais como seleção, ordenação e agrupamento, antes que a análise propriamente dita seja feita. Então, nessa parte do minicurso serão apresentadas algumas funcionalidades da biblioteca *Pandas*[4] do Python, que auxilia na leitura e manipulação de dados. Os exemplos citados são baseados em [McKinney, 2013] e na documentação da biblioteca.

DataFrame

O conceito mais importante da biblioteca *Pandas* é o de `DataFrame`, que consiste em uma estrutura bi-dimensional com elementos rotulados. Os rótulos das linhas estão em um atributo `index`, e os das colunas em `columns`. Essa estrutura pode ser abstraída como uma planilha de dados ou como uma tabela de um banco de dados relacional.

Um `DataFrame` pode ser criado de várias formas. Por exemplo, é possível inicializá-lo com os valores de uma matriz ou importá-lo de um arquivo de extensão `.csv`. O primeiro caso é ilustrado no código a seguir.

[4] `https://pandas.pydata.org`

```
In [1]: import numpy as np
In [2]: import pandas as pd
In [3]: dates = pd.date_range('20130101', periods=6)
In [4]: df = pd.DataFrame(np.random.randn(6,4), index=dates,
   ....:                                  columns=list('ABCD'))
In [5]: df
Out[5]:
                   A         B         C         D
2013-01-01 -0.684768  0.857024  0.040317 -2.954732
2013-01-02  1.015421  0.182357  0.483999 -0.649455
2013-01-03  0.212769 -1.153668 -1.006612 -0.642244
2013-01-04 -0.683321  0.297379  0.693017  0.021572
2013-01-05  0.376206  0.054179 -0.373296 -0.907636
2013-01-06  1.416673 -0.825072  1.505570 -2.277256
```

Para importar um `DataFrame` de um arquivo de extensão `.csv`, a função `read_csv` é utilizada.

```
In [6]: df = pd.read_csv('example.csv')
In [7]: df
Out[7]:
        foo       bar
0  1.274366  0.593544
1 -0.409548  0.999609
```

Se for necessário utilizar alguma coluna do arquivo como `index`, é preciso utilizar o parâmetro `index_col`, passando a posição da coluna que será utilizada.

```
In [8]: df = pd.read_csv('example.csv', index_col=0)
In [9]: df
Out[10]:
                bar
foo
 1.274366  0.593544
-0.409548  0.999609
```

Manipulação Básica

Há varias formas de se manimpular um `DataFrame` de forma simples e eficiente. Alguns exemplos são dados a seguir.

Abaixo, um novo `DataFrame` é criado passando uma matriz de tamanho 8×4, criada aleatoriamente, com os índices sendo dados pelo arranjo de datas e as colunas por uma lista de caracteres.

```
In [10]: dates = pd.date_range('20130101', periods=8)
In [11]: df = pd.DataFrame(np.random.randn(8,4), index=dates,
   ....:                                  columns=list('ABCD'))
```

É possível ver apenas as 5 primeiras (últimas) linhas de um `DataFrame` com o método `head` (`tail`).

```
In [12]: df.head()
Out[12]:
                   A         B         C         D
2013-01-01  1.040374  0.264662 -0.906420 -1.368221
2013-01-02 -0.588482 -1.739829  1.581274 -0.488375
2013-01-03  0.116974 -0.550166  0.489298 -1.298768
2013-01-04 -0.301473 -0.373992  0.132532  1.560100
2013-01-05 -1.713864  0.949441 -1.035164  2.080529
```

A matriz de valores correspondente ao `DataFrame` pode ser acessada pelo atributo `values`.

```
In [14]: df.values
Out[14]:
array([[ 1.04037406,  0.26466216, -0.90642008, -1.36822133],
       [-0.58848233, -1.73982906,  1.58127391, -0.48837542],
       [ 0.11697361, -0.55016634,  0.48929796, -1.29876785],
       [-0.30147253, -0.37399189,  0.13253181,  1.56009999],
       [-1.71386375,  0.94944098, -1.03516444,  2.0805291 ],
       [ 0.70227858,  0.05009946, -0.91479705,  0.44931671],
       [ 0.28091816,  1.76048814, -1.46628267,  1.03673774],
       [-0.22356422, -0.31369806, -1.57005912,  0.48789771]])
```

O `DataFrame` transposto é obtido pelo atributo `T`.

```
In [15]: df.T
Out[15]:
   2013-01-01  2013-01-02     ...      2013-01-07  2013-01-08
A    1.040374   -0.588482     ...        0.280918   -0.223564
B    0.264662   -1.739829     ...        1.760488   -0.313698
C   -0.906420    1.581274     ...       -1.466283   -1.570059
D   -1.368221   -0.488375     ...        1.036738    0.487898
[4 rows x 8 columns]
```

Existem outros métodos mais sofisticados para descrever e manipular os dados. Um deles é o `describe`, que gera estatísticas descritivas sobre os dados, excluindo atributos não numéricos.

```
In [16]: df.describe()
Out[16]:
              A         B         C         D
count  8.000000  8.000000  8.000000  8.000000
mean  -0.002739  0.374372  0.319950  0.317294
std    1.148546  0.695346  0.926048  1.161585
min   -1.245660 -1.033110 -0.999356 -1.458622
25%   -1.208835  0.237243 -0.367379 -0.257647
50%    0.150425  0.473167  0.686695  0.173904
75%    0.721654  0.758711  0.945877  1.009124
max    1.746989  1.142517  1.250873  2.285539
```

É possível ordenar um `DataFrame` pelos valores dos índices com o método `sort_index`. Neste caso, de maneira decrescente, utilizando o parâmetro `ascending`.

```
In [17]: df.sort_index(axis = 0, ascending = False)
Out[17]:
                   A         B         C         D
2013-01-08  0.961354 -0.130800  0.542649  1.199216
2013-01-07  0.539913  1.047744 -0.157967 -0.656732
2013-01-06 -1.245660  1.142517  0.830741  2.285539
2013-01-05 -1.223127  0.662367  0.847614  0.945761
2013-01-04 -0.239063  0.359925 -0.999356 -0.124618
2013-01-03 -1.204071 -1.033110 -0.995615 -1.458622
2013-01-02  1.746989  0.440088  1.250873  0.001847
2013-01-01  0.641754  0.506245  1.240665  0.345961
```

O `DataFrame` oferece um método para ordenar por valores, chamado `sort_values`. Neste caso, é necessário passar uma lista de colunas para ser feita a ordenação. Os valores são ordenados pela primeira coluna da lista. Se houver outra coluna, os valores são ordenados por esta, mas respeitando a ordenação da primeira coluna, e assim sucessivamente. Para exemplificar esse método, abaixo é criado um `DataFrame` de exemplo passando um dicionário para o construtor.

```
In [18]: df = pd.DataFrame({
   ....:       'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],
   ....:       'col2' : [2, 1, 9, 8, 7, 4],
   ....:       'col3': [0, 1, 9, 4, 2, 3],
   ....:  })
In [19]: df
Out[19]:
  col1  col2  col3
0    A     2     0
1    A     1     1
2    B     9     9
3  NaN     8     4
4    D     7     2
5    C     4     3
```

O `DataFrame` é ordenado pelos valores da primeira coluna.

```
In [20]: df.sort_values(by = ['col1'])
Out[20]:
  col1  col2  col3
0    A     2     0
1    A     1     1
2    B     9     9
5    C     4     3
4    D     7     2
3  NaN     8     4
```

O mesmo `DataFrame` é ordenado pelos valores da primeira coluna, e depois pelos valores da segunda coluna respeitando a ordenação feita pela primeira coluna.

```
In [21]: df.sort_values(by = ['col1', 'col2'])
Out[21]:
```

```
  col1  col2  col3
1    A     1     1
0    A     2     0
2    B     9     9
5    C     4     3
4    D     7     2
3  NaN     8     4
```

Seleção em DataFrames

Há várias formas de selecionar dados em um `DataFrame`. É permitido selecionar áreas de interesse por índices ou colunas de forma direta.

Um novo `DataFrame` é criado passando uma matriz de tamanho 6×4 criada aleatoriamente, com os índices sendo dados pelo arranjo de datas e as colunas por uma lista de caracteres.

```
In [22]: dates = pd.date_range('20130101', periods=6)
In [23]: df = pd.DataFrame(np.random.randn(6,4), index=dates,
   ....:                                columns=list('ABCD'))
In [24]: df
Out[24]:
                   A         B         C         D
2013-01-01  1.795895  0.428990 -0.911703  1.536664
2013-01-02  0.729177  1.448289 -0.302455 -0.079658
2013-01-03  0.276477 -1.167711 -0.788572 -0.609279
2013-01-04  0.088234  0.607176  1.080679 -1.163709
2013-01-05 -1.780621  0.293291  0.727409  1.817065
2013-01-06 -0.499000 -2.246464 -0.911982 -0.118192
```

Para acessar o valor de uma coluna é utilizado o operador `[]` ou o atributo com o nome da coluna.

```
In [25]: df['A'] #Ou df.A
Out[25]:
2013-01-01    1.795895
2013-01-02    0.729177
2013-01-03    0.276477
2013-01-04    0.088234
2013-01-05   -1.780621
2013-01-06   -0.499000
Freq: D, Name: A, dtype: float64
```

Para selecionar linhas pelo índice, é possível utilizar o operador `[:]`. São selecionadas as três primeiras linhas do `DataFrame`.

```
In [26]: df[0:3]
Out[26]:
                   A         B         C         D
2013-01-01  1.795895  0.428990 -0.911703  1.536664
2013-01-02  0.729177  1.448289 -0.302455 -0.079658
2013-01-03  0.276477 -1.167711 -0.788572 -0.609279
```

As linhas também podem ser selecionadas por seus rótulos. Abaixo, são selecionadas as linhas do índice `20130102` ao `20130104`.

```
In [27]: df['20130102':'20130104']
Out[27]:
                   A         B         C         D
2013-01-02  1.161427 -1.634133 -2.517285 -1.120083
2013-01-03 -1.409665  0.516400  0.241388  0.064734
2013-01-04 -0.293723  0.722081  0.033310  0.797433
```

Existem dois métodos muito úteis de seleção que permitem localizar linhas (ou colunas) satisfazendo um critério de busca (`loc`) ou especificando suas posições (`iloc`). Com o `loc` são utilizados os rótulos das linhas e colunas. Abaixo, selecionam-se todas as linhas relativas as colunas `A` e `B`.

```
In [28]: df.loc[:,['A','B']]
Out[28]:
                   A         B
2013-01-01  1.795895  0.428990
2013-01-02  0.729177  1.448289
2013-01-03  0.276477 -1.167711
2013-01-04  0.088234  0.607176
2013-01-05 -1.780621  0.293291
2013-01-06 -0.499000 -2.246464
```

Analogamente, é possível selecionar um subconjunto de linhas e um subconjunto de colunas.

```
In [29]: df.loc['20130102':'20130104',['A','B']]
Out[29]:
                   A         B
2013-01-02  0.729177  1.448289
2013-01-03  0.276477 -1.167711
2013-01-04  0.088234  0.607176
```

Já com o `iloc`, os índices são usados para a seleção. Neste caso, são selecionadas as linhas 3 e 4 e as colunas 0 e 1.

```
In [30]: df.iloc[3:5,0:2]
Out[30]:
                   A         B
2013-01-04  0.088234  0.607176
2013-01-05 -1.780621  0.293291
```

O método `at` é utilizado para selecionar rapidamente uma célula do `DataFrame`. Por exemplo, abaixo é selecionado o elemento da primeira linha na coluna `A`.

```
In [32]: df.at[df.index[0], 'A']
Out[32]: 1.7958951465197248
```

Combinando DataFrames

A biblioteca *Pandas* possuí vários métodos para combinar `DataFrames` que são similares a operações em um banco de dados por meio de SQL. Um deles é o `merge`, que possibilita realizar a *junção* de `Dataframes`. Considere os `Dataframes` `df1` e `df2` construídos abaixo.

```
In [33]: df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
...                           'value': [1, 2, 3, 5]})
In [34]: df1
Out[34]:
  lkey  value
0  foo      1
1  bar      2
2  baz      3
3  foo      5
In [35]: df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
...                           'value': [5, 6, 7, 8]})
In [36]: df2
Out[36]:
  rkey  value
0  foo      5
1  bar      6
2  baz      7
3  foo      8
```

A junção de `df1` e `df2` nas colunas `lkey` e `rkey` pode ser feita da seguinte forma:

```
In [37]: df1.merge(df2, left_on='lkey', right_on='rkey')
Out[37]:
  lkey  value_x rkey  value_y
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7
```

Pandas também possui a função `groupby`, a qual pode ser usada para uma ou mais das seguintes tarefas:

- Dividir os dados com base em algum critério.
- Aplicar a função para cada grupo.
- Combinar o resultado em um `DataFrame`.

O `groupby` suporta o agrupamento por uma lista de colunas, podendo ser aplicada uma função para cada agrupamento. Considere para um exemplo o seguinte `DataFrame`.

```
In [40]: df = pd.DataFrame({
   ...: 'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
   ...: 'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
   ...: 'C' : np.random.randn(8),
   ...: 'D' : np.random.randn(8)})
In [41]: df
Out[41]:
     A      B         C         D
0  foo    one  0.699437 -1.241271
1  bar    one -0.478602 -1.282819
2  foo    two -0.798638 -0.814409
3  bar  three -0.934389  1.347837
4  foo    two -1.078957 -1.107251
5  bar    two -1.041093 -1.617251
6  foo    one -1.942070  0.910410
7  foo  three  0.095535 -0.223301
```

Então, o DataFrame é agrupado pelas colunas `A` e `B`, sendo aplicada a função `sum` em cada grupo resultante.

```
In [42]: df.groupby(['A', 'B']).sum()
Out[42]:
                  C         D
A   B
bar one   -0.478602 -1.282819
    three -0.934389  1.347837
    two   -1.041093 -1.617251
foo one   -1.242633 -0.330861
    three  0.095535 -0.223301
    two   -1.877595 -1.921660
```

Visualização de Dados em DataFrames

A biblioteca *Pandas* também possui uma série de facilidades para visualização de dados. A seguir, alguns exemplos são apresentados.

Primeiro, é necessário importar o módulo *pyplot*, da biblioteca *matplotlib*.

```
In [43]: import matplotlib.pyplot as plt
```

Abaixo, cria-se um *DataFrame* de exemplo com valores aleatórios.

```
In [44]: df = pd.DataFrame(np.random.randn(10, 4), index=
   ....: pd.date_range('1/1/2000', periods=10),
   ....: columns=['A', 'B', 'C', 'D'])
```

Então chama-se o método `plot`. A saída desse método é mostrada na Figura 6.2. Repare que de forma simples gerou-se um gráfico com vários elementos complexos, incluindo: várias linhas de cores diferentes, legenda (com os nomes das colunas do *DataFrame*) e eixo horizontal com rótulos temporais (provenientes da coluna de índices do *DataFrame*).

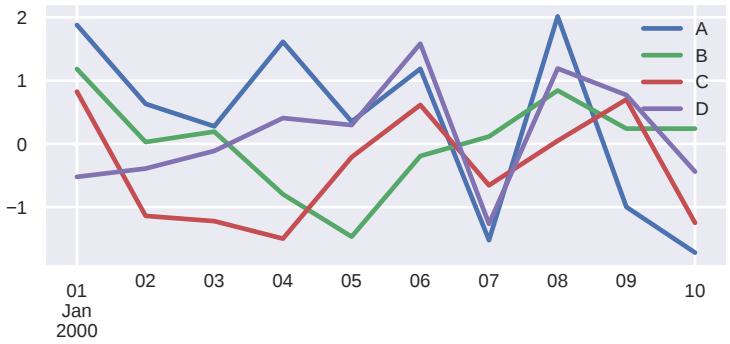


Figura 6.2: Exemplo de gráfico produzido com o método `plot`

.

```
In [45]: df.plot()
```

Por motivos de espaço, está fora do escopo deste texto discutir todos os aspectos da função plot. No entanto, salienta-se que vários outros tipos de gráficos podem ser criados. Entre eles: gráficos de barra, histogramas, *boxplots*, gráficos de dispersão e gráficos de pizza.

Nesta parte do minicurso, abordou-se *Pandas*, biblioteca desenvolvida na linguagem Python para facilitar o processo de leitura, manipulação e visualização de dados. A próxima parte relata algoritmos usados para reduzir a dimensionalidade desses dados, visando otimizar os processos de análise e extração de informação.

PARTE II - Redução de Dimensionalidade

Técnicas de aprendizado de máquina são utilizados para encontrar padrões ou tendências em dados. Em geral, esses padrões ou tendências são identificados por apresentar um conjunto de características semelhantes. Esse conjunto é representado computacionalmente através de um vetor de valores (inteiros, reais, lógicos, textos). O termo dimensionalidade é atribuído ao número de características de uma representação de padrões.

O objetivo desta parte é apresentar uma das tarefas mais recorrentes em análise de dados, a redução de dimensionalidade. Duas técnicas são discutidas e exemplificadas. Por fim, será mostrado como elas podem ser utilizadas para encontrar anomalias em matrizes de tráfego de redes.

Assume-se que os dados estão organizados em uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$. É comum, mas não obrigatório, interpretar as linhas de $\mathbf{X}$ como sendo *objetos* de interesse e as colunas representando *atributos*.

Da álgebra linear, sabe-se que o *rank* de $\mathbf{X}$ é o número de colunas (ou linhas) linearmente independentes. Por exemplo, se o *rank* de $\mathbf{X}$ for $r < \min\{n,d\}$, então é possível remover as colunas (ou linhas) que são formadas de combinações lineares das demais. A matriz resultante é menor, mas contém a mesma "informação" que $\mathbf{X}$.

Na prática, raramente encontra-se uma matriz de dados cujo *rank* seja menor que suas dimensões. No entanto, é comum encontrar matrizes com baixo *rank efetivo* [Roy and Vetterli, 2007]. Nesse caso, o *rank* real da matriz pode até ser máximo, mas tal

matriz pode ser "bem" aproximada por uma matriz cujo *rank r* seja muito menor que $\min\{n,d\}$. Para encontrar essa aproximação, técnicas de redução de dimensionalidade podem ser empregadas. Tais técnicas são interessantes, ou muitas vezes necessárias, por vários motivos. Entre eles:

1. Em muitas situações, os objetos de interesse são descritos por uma grande quantidade de atributos, quando na verdade um pequeno subconjunto desses atributos (ou uma combinação linear deles) seria suficiente para descrevê-los de maneira apropriada. Nesse caso, a redução de dimensionalidade auxilia na eficiência de armazenamento e no desempenho de alguns algoritmos.

2. Ao lidar com objetos em dimensionalidades muito altas, a intuição que muitas vezes é aplicada para lidar com objetos de uma, duas ou três dimensões não é mais válida. Esse fenômeno é conhecido como *a maldição da dimensionalidade* (sugere-se a leitura do Capítulo 6 de [Zaki and Jr, 2014]).

3. Há varias aplicações interessantes (dentro e fora da área de redes) baseadas na teoria *rank* efetivo de matrizes. Por exemplo, sistemas de recomendação e detecção de anomalias. Esse último será discutido mais adiante.

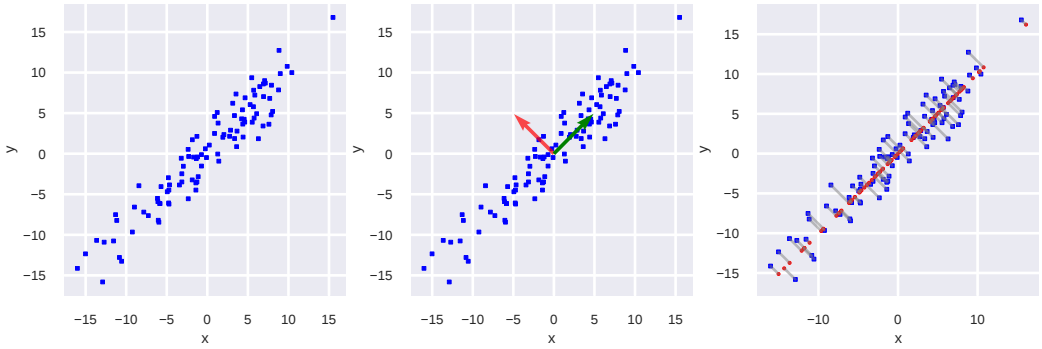
Duas técnicas populares, e de certa forma relacionadas, para redução de dimensionalidade são a análise de componentes principais e a decomposição em valores singulares, as quais são conhecidas como PCA (*Principal Component Analysis*) e SVD (*Singular Value Decomposition*), respectivamente.

A seguir, as técnicas PCA e SVD serão brevemente descritas, assim como utilizá-las de forma simples em Python. Em seguida, será apresentado como elas podem ser utilizadas para detectar anomalias em tráfego de uma rede, como proposto em [Lakhina et al., 2005]. Ao leitor interessado em mais detalhes e demonstrações matemáticas sobre os conceitos aqui abordados, sugere-se a leitura do Capítulo 7 de [Zaki and Jr, 2014].

PCA – Análise de Componentes Principais

A Figura 6.3a apresenta um conjunto de dados composto por uma coleção de pontos pertencentes ao $\mathbb{R}^2$. Pode-se perceber que os pontos estão basicamente distribuídos na vizinhança da reta $y = x$. A Figura 6.3b sobrepõe ao conjunto de pontos dois vetores ortogonais. Um deles, em verde, acompanha a direção em que a variância do conjunto de pontos é maior. Já a Figura 6.3c apresenta a projeção de cada ponto sobre a reta formada ao longo da direção de maior variância. Uma vez que a aproximação por uma reta parece razoável, pode-se pensar em substituir o conjunto de dados original pelas coordenadas de cada ponto no subespaço gerado pela reta. Dessa forma, a representação dos dados será simplificada, com uma pequena perda de informação. Para cada ponto, essa perda, ou erro, é representada pela linha cinza na Figura 6.3c, representando a distância entre o ponto original (em azul) e o ponto projetado no subespaço (em vermelho).

A técnica de Análise de Componentes Principais permite, de forma geral, executar os passos ilustrados pela Figura 6.3. Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, representando n objetos do espaço d-dimensional (assumindo que $n \geq d$ e que o *rank* de $\mathbf{X}$ seja d), PCA permite:



(a) (b) (c)

Figura 6.3: Intuição da técnica de PCA: (a) conjunto de pontos no plano; (b) direções ortogonais de maiores variâncias; (c) projeções dos pontos no subespaço gerado na direção de maior variância.

1. encontrar d direções ortogonais que capturam a variância máxima do conjunto de dados, assim como na Figura 6.3b;

2. encontrar a variância dos dados associada a cada uma das direções ortogonais. A direção contendo a maior variância é denominada *primeira componente principal*. De forma geral, a direção associada à i-ésima maior variância é denominada i-ésima componente principal.

3. projetar os n objetos no subespaço r-dimensional ($r \leq d$) gerado pelas r primeiras componentes principais, assim como na Figura 6.3c. Além disso, é possível obter as coordenadas de cada objeto neste novo subespaço, permitindo substituir o conjunto de dados original por uma representação aproximada e mais compacta.

A biblioteca *Scikit-learn* do Python permite utilizar PCA de uma forma bem simples e conveniente.[5] A seguir, será ilustrado como aplicar PCA ao conjunto de dados da Figura 6.3. Primeiro, é necessário gerar o conjunto de dados, o que é feito no código abaixo utilizando a biblioteca *numpy*.[6] Em tal código, 100 observações de uma distribuição normal bivariada são geradas.

```
In [1]: import numpy as np
In [2]: np.random.seed(0)
In [3]: sigma = np.array([[41, 39], [39, 41]])
In [4]: x = np.random.multivariate_normal([0, 0], sigma, 100)
```

A seguir, é necessário importar a biblioteca e criar um objeto da classe `PCA`, passando como argumento r, i.e., o número de componentes principais que tem-se interesse.

```
In [5]: from sklearn.decomposition import PCA
In [6]: pca = PCA(n_components=1)
```

[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[6] http://www.numpy.org

Para aplicar PCA aos dados gerados, `x`, é necessário utilizar o método `fit`.

```
In [7]: pca.fit(x)
```

Agora, é possível ter acesso às *r* primeiras componentes principais e às variâncias associadas a cada uma delas, como indicado a seguir.

```
In [8]: pca.components_
Out[8]: array([[0.70436626, 0.70983672]])

In [9]: pca.explained_variance_
Out[9]: array([84.3970249])

In [10]: pca.explained_variance_ratio_
Out[10]: array([0.97590927])
```

No código acima, o atributo `components_` contém os vetores (direções) de cada componente principal. O atributo `explained_variance_` informa a variância associada a cada componente. Já o atributo `explained_variance_ratio_` apresenta a porcentagem da variância total dos dados que é explicada por cada componente. No caso do exemplo, a primeira componente captura cerca de 98% da variância total dos dados, e por isso, a aproximação dos dados por uma reta é razoável.

Por fim, para efetivamente reduzir dimensionalidade de `x`, pode-se utilizar o método `transform`, o qual faz a projeção dos pontos originais no subespaço gerado pelas *r* primeiras componentes principais. Esse recurso é exemplificado no código abaixo. Repare que a matriz original possui duas colunas, ao passo que a matriz de dados reduziada possui apenas uma.

```
In [12]: pca.transform(x)
Out[12]:
array([[-15.78520352],
       [ -8.7511212 ],
       [-16.71851443],
       [ -8.50796033],
       [  0.91610997],
       ...

In [13]: x
Out[13]:
array([[-11.55700386, -10.75668944],
       [ -8.43097572,  -3.94918933],
       [-10.83419594, -12.7887517 ],
       [ -5.85752955,  -6.16024396],
       [  0.24221484,   1.06341184],
       ...
```

SVD – Decomposição em Valores Singulares

Considere uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, e assuma que o *rank* de $\mathbf{X}$ seja l. A decomposição em valores singulares de $\mathbf{X}$ consiste em reescrever $\mathbf{X}$ como o produto de três matrizes da

seguinte maneira:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{1}$$

onde:

1. $\mathbf{U} \in \mathbb{R}^{n \times l}$ é uma matriz ortogonal. As colunas de $\mathbf{U}$ são denominadas de vetores singulares à esquerda.
2. $\mathbf{S} \in \mathbb{R}^{l \times l}$ é uma matriz diagonal, cujos valores da diagonal são números reais positivos e ordenados de maneira decrescente. Tais valores são denominados valores singulares.
3. $\mathbf{V} \in \mathbb{R}^{d \times l}$ é também uma matriz ortogonal. As colunas de $\mathbf{V}$ são denominadas vetores singulares à direita.

Por que esta decomposição é interessante? Dentre os vários motivos, dois destacam-se. Primeiro, pode-se demonstrar que é possível (e eficiente) fazer PCA utilizando SVD. Segundo, dado $r \leq l$, considere as matrizes:

1. $\mathbf{U}_r \in \mathbb{R}^{n \times r}$ sendo a matriz formada pelas primeiras r colunas de $\mathbf{U}$;
2. $\mathbf{S}_r \in \mathbb{R}^{r \times r}$ sendo a matriz quadrada formada pelos elementos das r primeiras linhas e r primeiras colunas de $\mathbf{S}$;
3. $\mathbf{V}_r \in \mathbb{R}^{d \times r}$ sendo a matriz formada pelas r primeiras colunas de $\mathbf{V}$; e
4. $\mathbf{X}_r = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^T$.

É possível mostrar que $\mathbf{X}_r$ é a melhor aproximação, com *rank* r, de $\mathbf{X}$, ou seja, pode-se mostrar que $\mathbf{X}_r$ minimiza

$$f(x) = ||x - \mathbf{X}||_F, \tag{2}$$

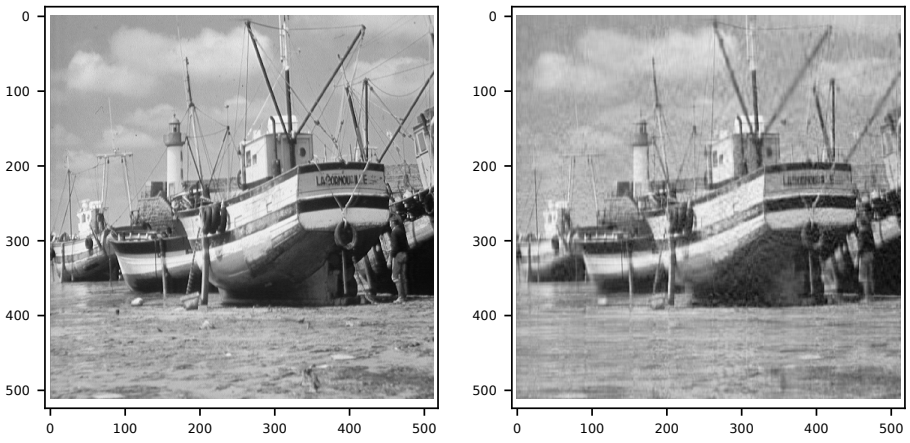
no espaço de todas as matrizes com n linhas, d colunas e *rank* r. Na Equação (2), $||\cdot||_F$ denota a norma de *Frobenius*.

O poder da técnica SVD será ilustrado por meio de um exemplo de compressão de imagens. A Figura 6.4a apresenta a imagem, em escala de cinza, de um barco. A imagem tem 512 *pixels* de altura e largura. Uma vez que a imagem está em escala de cinza, esta pode ser interpretada como um matriz com 512 linhas e 512 colunas, onde cada célula da matriz indica a intensidade de cinza. Originalmente, são necessários 512×512 valores para representar a imagem.

Ao aplicar SVD e computar a matriz $\mathbf{X}_{40}$, a imagem da Figura 6.4b é obtida. Observe que a qualidade da imagem diminui, mas agora são necessário apenas $2 \times 40 \times 512 + 40$ valores para armazenar a imagem, i.e., 6.4 vezes menos em comparação com a imagem original.

Python permite aplicar SVD de uma maneira muito conveniente, por meio da biblioteca *numpy*.[7] No exemplo abaixo, uma matriz de números aleatórios é gerada. Após isso, os vetores e valores singulares são obtidos.

[7] `https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html`



(a) (b)

Figura 6.4: Utilização de SVD para compressão de imagens: (a) imagem original; (b) approximação da imagem utilizando uma matriz de *rank* 40.

```
In [1]: import numpy as np
In [2]: np.random.seed(0)
In [3]: X = np.random.random((4, 3))
In [4]: X
Out[5]:
array([[0.5488135 , 0.71518937, 0.60276338],
       [0.54488318, 0.4236548 , 0.64589411],
       [0.43758721, 0.891773  , 0.96366276],
       [0.38344152, 0.79172504, 0.52889492]])
In [6]: U, S, Vt = np.linalg.svd(X, full_matrices = False)

In [6]: U
Out[6]:
array([[-0.48665747,  0.08953811, -0.54332309],
       [-0.41275168,  0.81483266, -0.00407301],
       [-0.62010377, -0.2250203 ,  0.73183668],
       [-0.45636813, -0.52668447, -0.41133746]])

In [7]: S
Out[7]: array([2.21425506, 0.29626672, 0.21874633])

In [8]: Vt
Out[8]:
array([[-0.42376582, -0.64907895, -0.63175869],
       [ 0.65045995, -0.70346017,  0.2864361 ],
       [-0.63033672, -0.28955189,  0.72030224]])
```

É importante observar que no retorno do método da biblioteca *numpy* apenas os valores singulares são retornados, i.e., a diagonal da matriz **S**. Além disso, observe que o método retorna a matriz transposta de **V** e não a matriz **V** em si.

Exemplo: Minerando Anomalias de Tráfego

A redução da dimensionalidade foi aplicada por [Lakhina et al., 2005] para identificar anomalias de tráfego na rede. O exemplo apresentado a seguir é uma adaptação deste trabalho.

O problema de detecção de anomalias pode ser visto da seguinte forma: dada uma população P, tem-se o objetivo em dividir os elementos de P em um conjunto N de elementos "normais" e outro O, de indivíduos "anômalos". Uma maneira usual de atacar tal problema é utilizar aprendizado supervisionado para construir um modelo que separe N de O. O problema dessa abordagem é que ela requer dados rotulados para treinar o modelo, o que nem sempre está disponível. No entanto, se o fenômeno da baixa dimensionalidade puder ser observado (i.e., se os dados puderem ser organizado em uma matriz com baixo *rank* efetivo), é possível detectar anomalias de forma apropriada sem a necessidade de dados previamente rotulados.

Para este fim, assume-se que na população P, a maioria das observações seguem o comportamento "normal". Então, pode-se usar o fenômeno de baixa dimensionalidade para separar as anomalias das não anomalias. Para tal, é necessário obter um modelo (linear) de baixa dimensão para os dados. Assim, todo comportamento que não puder ser bem descrito pelo modelo, provavelmente foge da normalidade e portanto, pode ser enquadrado como uma anomalia. Na prática, dada uma matriz de dados $\mathbf{X}$, os seguintes passos devem ser seguidos:

1. Obtenha a decomposição em valores singulares de $\mathbf{X}$

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T; \tag{3}$$

2. Obtenha uma aproximação de $\mathbf{X}$ com baixo *rank* efetivo, r

$$\mathbf{X}_r = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^T; \tag{4}$$

3. Compute a porção dos dados que não pode ser explicada por $\mathbf{X}_r$

$$\mathbf{O} = \mathbf{X} - \mathbf{X}_r; \tag{5}$$

4. Identifique as linhas (ou colunas) de $\mathbf{O}$ com as maiores normas: essas linhas (colunas) correspondem às anomalias.

Para aplicar essa metodologia à detecção de anomalias no tráfego de uma rede de computadores, primeiro, é necessário modelar tal tráfego como uma matriz. Usualmente, isso é feito da seguinte forma: o tráfego entre cada par origem-destino (OD) é medido em intervalos fixos de tempo. Após isso, uma matriz $\mathbf{X}$ é construída, onde o elemento (i, j) da matriz é referente ao tráfego do par OD j no intervalo de tempo i. Como caso de estudo, considere o tráfego observado na rede *Abilene*[8] durante uma semana de setembro de 2003. A rede contém 120 pares origem-destino, para os quais o tráfego da origem para

[8] `https://uit.stanford.edu/service/network/internet2/abilene`

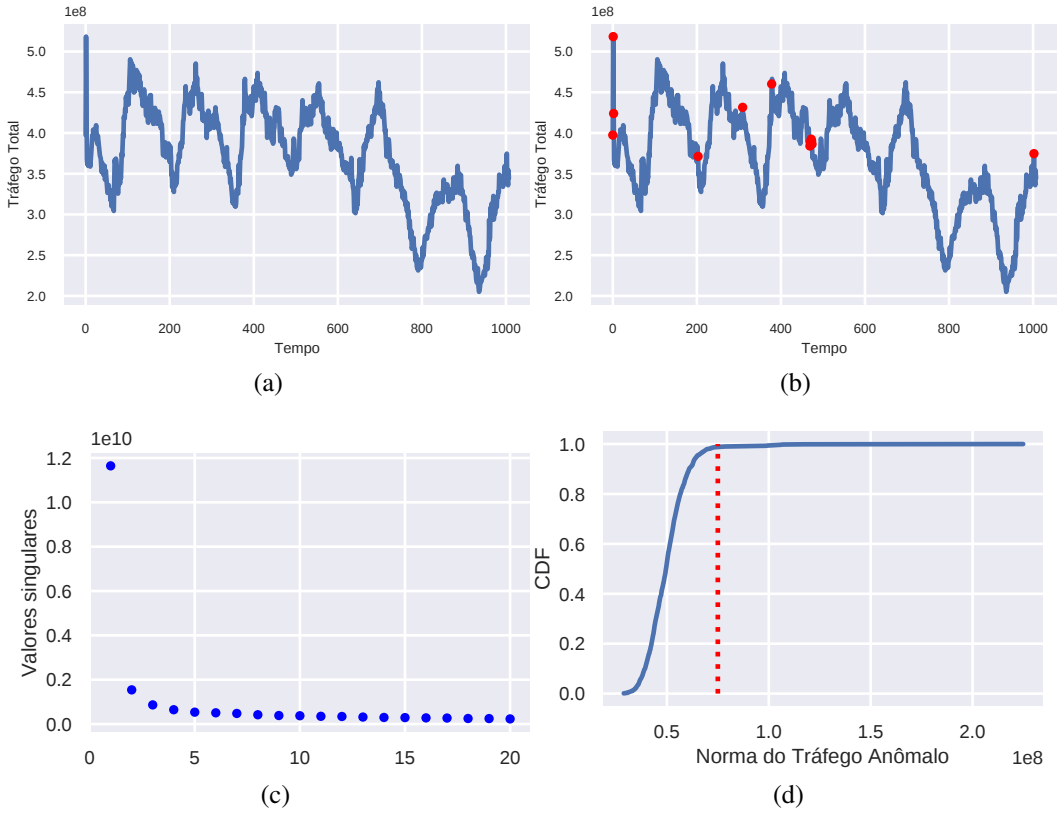


Figura 6.5: (a) Tráfego total na rede; (b) Tráfego total na rede, onde os pontos em vermelho são considerados anômalos; (c) Norma ℓ_2 das linhas da matriz de $\mathbf{O}$; (d) Distribuição acumulada (CDF – *Cumulative Distribution Function*) das normas ℓ_2 das linhas de $\mathbf{O}$.

o destino foi medido em intervalos de 10 minutos. A Figura 6.5a apresenta o tráfego total medido em cada intervalo, i.e., a soma dos elementos de cada linha da matriz de tráfego. A pergunta que deseja-se responder é: há algum ponto de tempo em que é possível afirmar que o tráfego na rede é anômalo? Responder essa pergunta é uma tarefa desafiadora, uma vez que os dados contêm padrões de variação diários e semanais. Além disso, uma anomalia pode surgir por diferentes motivos, e.g., um ataque à rede, falha de equipamento ou uma demanda repentina devido a um evento externo.

De acordo com a metodologia sumarizada acima, é preciso encontrar uma representação de baixa dimensionalidade para $\mathbf{X}$. Mas como definir r? Uma heurística usual consiste em analisar o gráfico dos valores singulares de $\mathbf{X}$ e definir r como sendo a abscissa do "joelho" da curva, i.e., o ponto em que a curva para de decrescer de forma brusca. Para o conjunto de dados em questão, o gráfico é apresentado na Figura 6.5c, o qual indica que $r = 6$ é uma escolha razoável e conservadora.

A Figura 6.5d apresenta a distribuição das normas ℓ_2 das linhas de $\mathbf{O} = \mathbf{X} - \mathbf{X}_6$. Na figura, pode-se perceber que valores acima de 0.75×10^8 são infrequentes. Por isso, escolheu-se esse valor como limiar para indicar se um dado intervalo de tempo apresenta comportamento anômalo ou não. Há na matriz $\mathbf{O}$ 13 linhas (cada uma referente a um intervalo de tempo) cujas normas excedem esse limiar. Os intervalos de tempo associ-

ados a cada uma dessas linhas são destacadas junto ao tráfego total da rede na Figura 6.5b. Pode-se perceber que, em alguns dos casos, o tráfego anômalo é referente a uma observação globalmente discrepante. No entanto, este fato não é uma regra. Há também observações discrepantes locais e outras que são consideradas anômalas por não poderem ser explicadas por um modelo linear de baixa dimensão.

Nesta parte do minicurso foram apresentadas PCA (*Principal Component Analysis*) e SVD (*Singular Value Decomposition*), duas formas de reduzir a dimensionalidade dos vetores de características usados para representar padrões nos dados. Um exemplo onde essa técnica foi aplicada na área de redes também foi relatado. Na parte seguinte, serão apresentados algoritmos que conseguem identificar automaticamente padrões nos dados.

PARTE III - Aprendizado Não Supervisionado

Aprendizado não supervisionado é uma área de aprendizado de máquina que tem por objetivo *aprender* a partir de dados não rotulados. A tarefa mais comum em aprendizado não supervisionado, e que será o foco desse texto, é a de agrupamento (ou *clustering*).

Nesse contexto, o objeto de estudo tipicamente é uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, onde n é o número de objetos a serem agrupados e d é a dimensionalidade de cada objeto. Informalmente, o problema de interesse pode ser descrito como dividir as n linhas de $\mathbf{X}$ em k grupos naturais, ou seja, de forma que:

- elementos do mesmo grupo sejam "similares"; e
- elementos de grupos diferentes não sejam "similares".

Há na literatura uma variedade de algoritmos para resolver o problema de agrupamento, com base em heurísticas ou de forma exata. Cada algoritmo possui características específicas com relação à eficiência (tamanho do problema que consegue-se resolver em um tempo razoável), aos tipos de grupos naturais que conseguem identificar e às diferentes noções de "similaridade" entre objetos.

Considere como exemplo a Figura 6.6. Um ser humano é capaz de identificar diferentes grupos naturais em cada uma das subfiguras sem grandes dificuldades. Apesar do exemplo da figura parecer trivial, um olhar mais cuidadoso revela vários desafios para abordagens automatizadas. Entre eles: grupos de tamanhos não uniformes, grupos não convexos e grupos com densidades diferentes. Ao longo desse texto, estes exemplos serão utilizados para mostrar que poucos algoritmos conseguem aprender "bem" em todas essas situações.

Dessa forma, pretende-se dar uma visão geral sobre o problema de agrupamento por meio da apresentação das características de alguns dos métodos populares, a saber: K-Means; Agrupamento aglomerativo; Agrupamento baseado em densidade; e Agrupamento espectral. Além disso, serão mostrados recursos da biblioteca *Scikit-learn*, do Python, para a utilização de tais algoritmos.

[9] `https://scikit-learn.org/stable/modules/clustering.html`

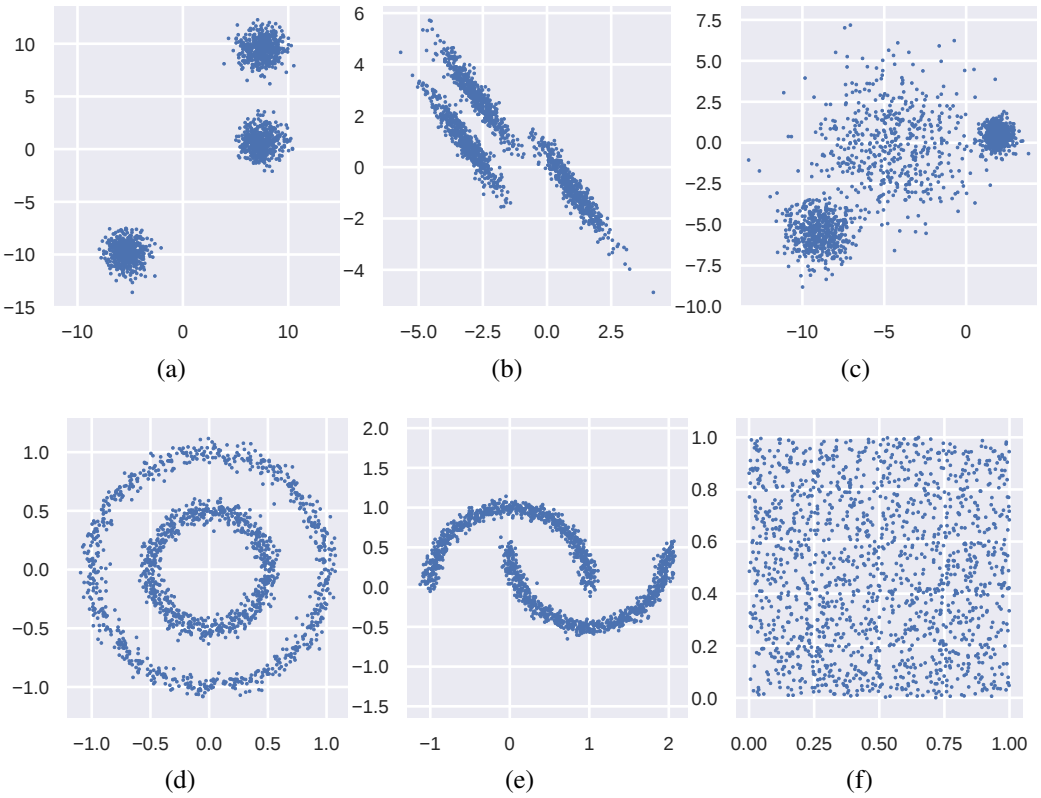


Figura 6.6: Exemplos de vários tipos de grupos naturais: (a) grupos circulares; (b) grupos alongados e de mesmo tamanho; (c) grupos com densidades diferentes; (d) grupos circulares e concêntricos; (e) grupos não convexos; (f) sem estrutura de grupos. Exemplos baseados na documentação da biblioteca *Scikit-learn*.[9]

Por fim, o contexto de redes de computadores será abordado por meio de um exemplo, onde algoritmos de agrupamento serão utilizados para detectar "comunidades" em topologias de rede.

K-means

O algoritmo K-means é uma das abordagens mais populares para tarefa de agrupamento de dados. Atualmente, o termo K-means é utilizado para uma série de heurísticas baseadas no trabalho de [Forgy, 1965], as quais consistem em variações e melhoramentos do algoritmo proposto em 1965. Formalmente, dados $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ ($\mathbf{x}_i \in \mathbb{R}^d$) e um inteiro $k \geq 2$, o K-means tem o objetivo de minimizar

$$f(\mathbf{c}_1, \ldots, \mathbf{c}_k) = \sum_{i=1}^{n} \min_{1 \leq j \leq k} ||\mathbf{x}_i - \mathbf{c}_j||^2, \tag{6}$$

onde $||\cdot||$ representa a norma ℓ_2 e $\mathbf{c}_j \in \mathbb{R}^d$ ($1 \leq j \leq k$). Intuitivamente, a otimização dessa função objetivo consiste em encontrar k pontos no espaço d-dimensional, denominados centroides, que "representem" bem o conjunto de pontos. Na Figura 6.6a, por exemplo, para $k = 3$, uma boa solução é aquela que coloca cada centroide no centro de um aglomerado.

O problema de minimização descrito acima é NP-difícil. No entanto, as heurísticas conhecidas na literatura funcionam razoavelmente bem para conjuntos de dados não muito grandes. A biblioteca *Scikit-learn*, do Python, traz por padrão uma das variações mais populares, o *K-means++*. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `KMeans`. No caso do exemplo, é informado que deseja-se agrupar os pontos em 3 grupos. Então, o método `fit` é utilizado para executar o algoritmo como os parâmetros especificados no construtor.

```
In [5]: kmeans = cluster.KMeans(n_clusters = 3)
In [6]: kmeans.fit(X)
```

Após a execução, é possível identificar o grupo de cada ponto do conjunto de dados e o valor da função objetivo (soma dos quadrados das distâncias de cada ponto para o respectivo centroide) por meio dos atributos `labels_` e `inertia_`.

```
In [7]: kmeans.labels_
Out[7]: array([2, 2, 2, ..., 2, 1, 1], dtype=int32)
In [8]: kmeans.inertia_
Out[8]: 2959.7867290413565
```

Repare que a implementação da biblioteca *Scikit-learn* exige o que o número de grupos seja fornecido. No caso do exemplo, a escolha de $k = 3$ é imediata, mas o que fazer quando os dados não podem ser facilmente visualizados ou quando não há uma intuição sobre o número de grupos naturais?

Uma técnica popular para tal tarefa consiste em observar o valor da função objetivo para diferentes valores de k. Aumentar k implica em diminuir o valor da função; no entanto, ao se incrementar o valor de k, se o decremento da função objetivo não for significativo, pode-se entender que não há necessidade de realizar o incremento. A Figura 6.7 apresenta a aplicação dessa técnica para o conjunto de pontos da Figura 6.6a.

Por fim, a Figura 6.8 apresenta o resultado da aplicação do K-means para os conjuntos de pontos da Figura 6.6. A técnica não é indicada para situações em que os grupos naturais possuem geometria não convexa, densidade variável ou grupos alongados (e.g., Figuras 6.8b, 6.8c, 6.8d e 6.8e).

Agrupamento Aglomerativo

O agrupamento aglomerativo usa uma abordagem *bottom-up*, criando uma sequência de partições aninhadas, que pode ser vista como uma hierarquia de grupos, também chamada de dendrograma. A Figura 6.9 mostra um exemplo de dendrograma.

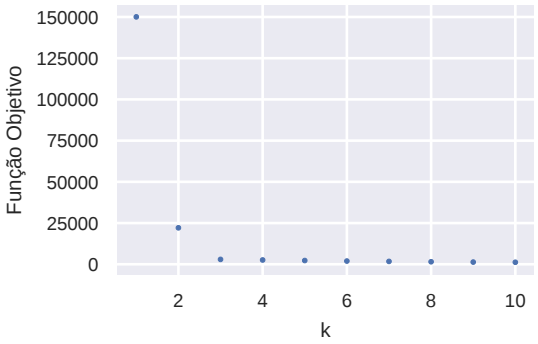


Figura 6.7: Valor da função objetivo para diferentes valores de k. Pode-se observar que o decremento do valor da função para $k > 3$ é quase nulo. Esse resultado sugere que $k = 3$ é uma escolha razoável para o número de grupos para o conjunto de dados em questão.

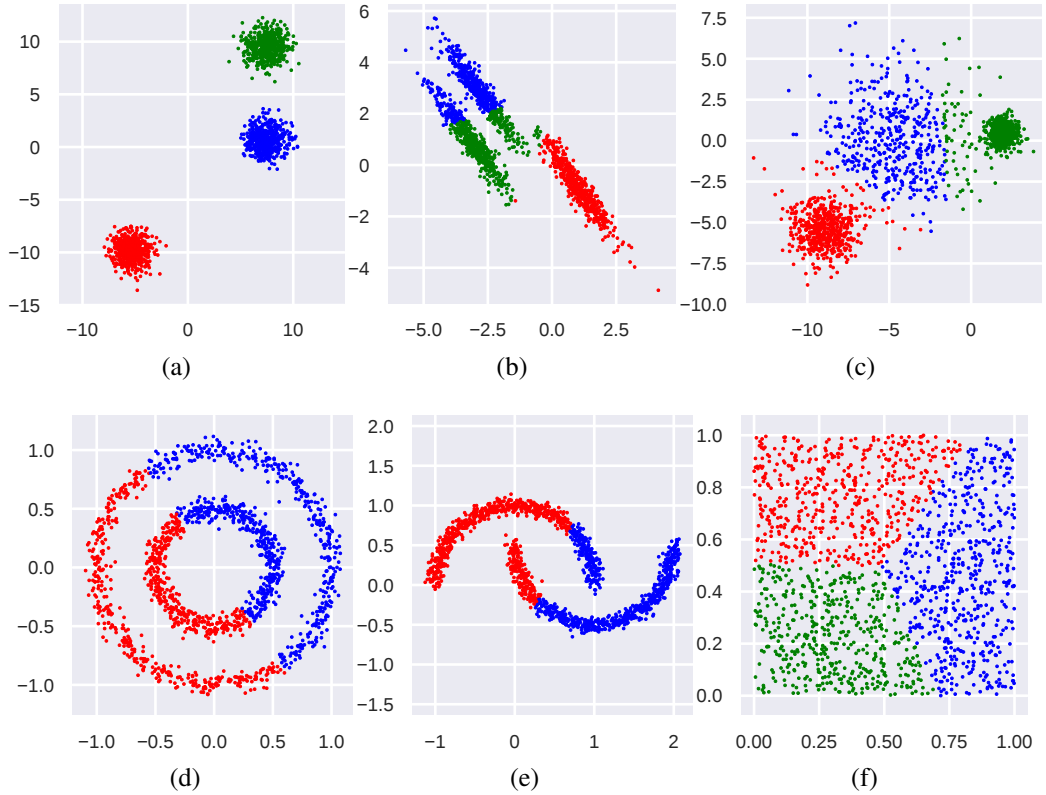


Figura 6.8: Aplicação do algoritmo K-means para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3.

Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, e um inteiro k, o agrupamento aglomerativo funciona da seguinte forma:

1. Na inicialização, cada $\mathbf{x}_i$ se torna um grupo $\{\mathbf{x}_i\}$.
2. O próximo passo consiste de um laço onde os dois grupos mais similares são aglomerados em um só até que restem apenas k grupos.

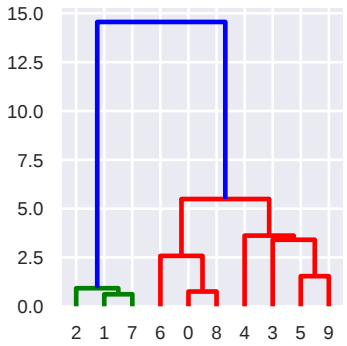


Figura 6.9: Exemplo de um dendrograma

Note que, dependendo de como é calculada a distância entre os grupos, o resultado do agrupamento pode ser diferente. Algumas variantes de como esse cálculo de distância pode ser feito são listadas a seguir. Em todos os casos, $\delta(\mathbf{x},\mathbf{y})$ denota a distância Euclidiana entre os pontos $\mathbf{x}$ e $\mathbf{y}$.

- *Single Link*: A distância entre dois grupos C_1 e C_2 é dada pela distância mínima entre um ponto em C_1 e outro em C_2

$$\delta(C_1,C_2) = min\{\delta(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in C_1, \mathbf{y} \in C_2\}.$$

- *Complete Link*: A distância entre dois grupos C_1 e C_2 é dada pela distância máxima entre um ponto em C_1 e outro em C_2

$$\delta(C_1,C_2) = max\{\delta(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in C_1, \mathbf{y} \in C_2\}.$$

- *Group Average*: A distância entre dois grupos C_1 e C_2 é dada pela distância média par a par entre pontos em C_1 e C_2

$$\delta(C_1,C_2) = \frac{\sum_{\mathbf{x}\in C_1}\sum_{\mathbf{y}\in C_2}\delta(\mathbf{x},\mathbf{y})}{n_1 n_2},$$

  onde n_1 e n_2 são o número de pontos em C_1 e C_2, respectivamente.

- *Ward's Method*: A distância entre dois grupos C_1 e C_2 é dada pela aumento na soma dos erros quadrados, de cada ponto para o centroide do grupo, quando dois grupos são unidos

$$\delta(C_1,C_2) = \left(\frac{n_1 n_2}{n_1+n_2}\right)\|\mu_1-\mu_2\|^2,$$

  onde $\mu_1 = \frac{1}{n_1}\sum_{\mathbf{x}\in C_1}\mathbf{x}$ e $\mu_2 = \frac{1}{n_2}\sum_{\mathbf{x}\in C_2}\mathbf{x}$

A biblioteca *Scikit-learn*, do Python, traz a implementação do Agrupamento Aglomerativo com todas essas variações do cálculo da distância entre grupos. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `AgglomerativeClustering`. No caso do exemplo, é informado que deseja-se agrupar os pontos em 3 grupos com o cálculo da distância entre os grupos sendo feito com o método de *Ward*. Então, o método `fit` é utilizado para executar o algoritmo com os parâmetros especificados no construtor.

```
In [5]: agg = cluster.AgglomerativeClustering(n_clusters = 3,
   ...:        linkage = 'ward')
In [6]: agg.fit(X)
```

Após a execução, é possível identificar o grupo de cada ponto do conjunto de dados por meio do atributo `labels_`.

```
In [7]: agg.labels_
Out[7]: array([2, 2, 2, ..., 2, 1, 1], dtype=int32)
```

Por fim, a Figura 6.10 apresenta o resultado da aplicação do Agrupamento Aglomerativo para os conjuntos de pontos da Figura 6.6.

Agrupamento Baseado em Densidade

O Agrupamento Baseado em Densidade mais popular, chamado DBSCAN, tem o objetivo de encontrar grupos de alta densidade (pontos por região) que sejam isolados uns dos outros.

Considere uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, um número real ε e um inteiro η. Em alto nível, o DBSCAN funciona da seguinte forma:

1. Para cada ponto $\mathbf{x}$, encontre todos os vizinhos numa distância máxima ε. Os pontos que tiverem mais de η vizinhos dentro desta distância são denominados *core points*;
2. Encontre conjuntos maximais de *core points* tais que cada *core point* esteja em uma ε-vizinhança de ao menos um outro *core point*;
3. Associe a cada ponto que não é um *core points* ao grupo mais próximo, se a distância for inferior a ε;
4. Todos os outros pontos são rotulados como ruído.

A vantagem do Agrupamento Baseado em Densidade é a facilidade de encontrar grupos não convexos. Mas sua aplicação é um pouco mais complicada, já que não há definição do número de grupos. O resultado é baseado nos valores de ε e η, os quais devem ser fornecidos pelo usuário.

A biblioteca *Scikit-learn*, do Python, traz a implementação do DBSCAN. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

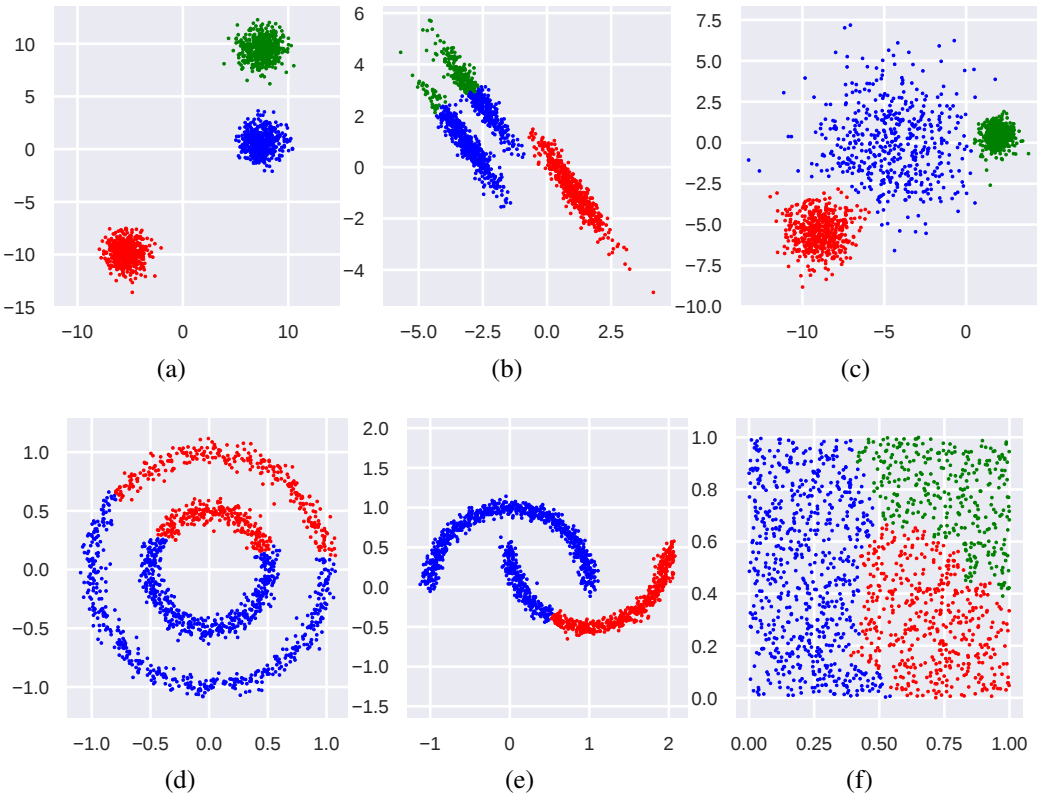


Figura 6.10: Aplicação do Agrupamento Aglomerativo usando o método de *Ward* para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `DBSCAN`. No caso do exemplo, é informado que deseja-se agrupar os pontos utilizando $\varepsilon = 1.5$ e $\eta = 5$. Então, o método `fit` é utilizado para executar o algoritmo como os parâmetros especificados no construtor.

```
In [5]: dbs = cluster.DBSCAN(eps = 1.5, min_samples = 5)
In [6]: dbs.fit(X)
```

Então, é possível identificar o grupo de cada ponto do conjunto de dados e o índice dos *core-points* por meio dos atributos `labels_` e `core_sample_indices_`.

```
In [7]: dbs.labels_
Out[7]: array([0, 0, 0, ..., 0, 2, 2], dtype=int32)
In [8]: dbs.core_sample_indices_
Out[8]: array([   0,    1,    2, ..., 1497, 1498, 1499], dtype=int32)
```

A Figura 6.11 apresenta o resultado da aplicação do DBSCAN para os conjuntos de pontos da Figura 6.6.

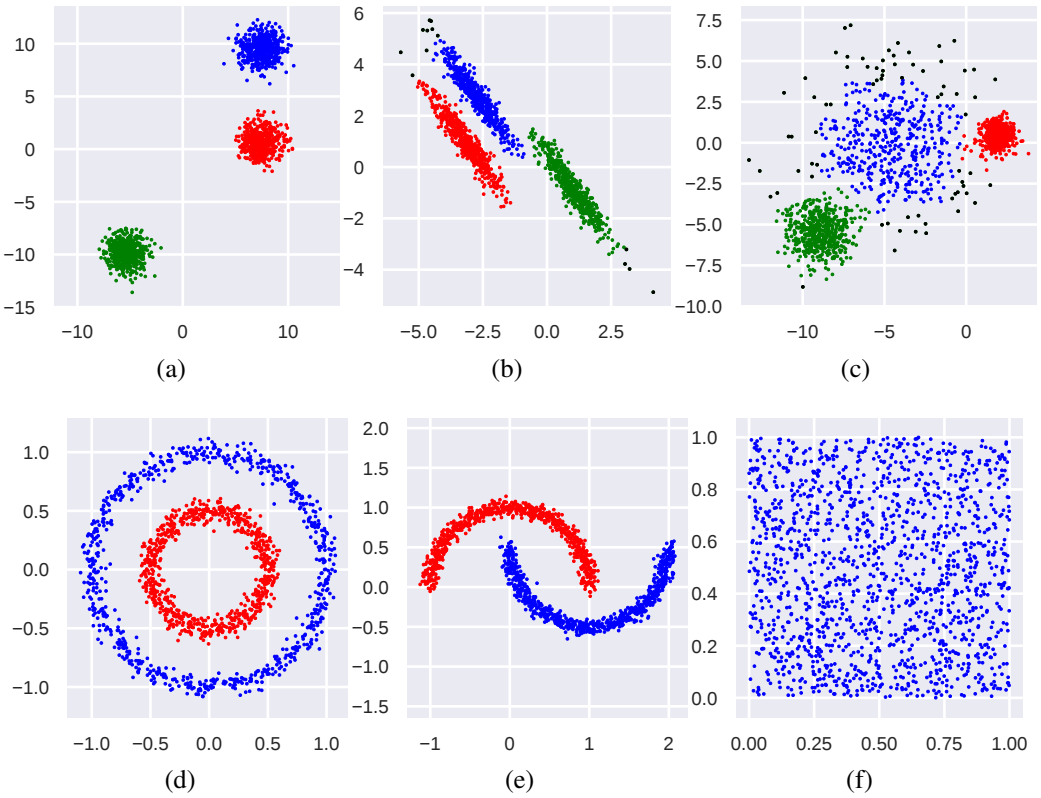


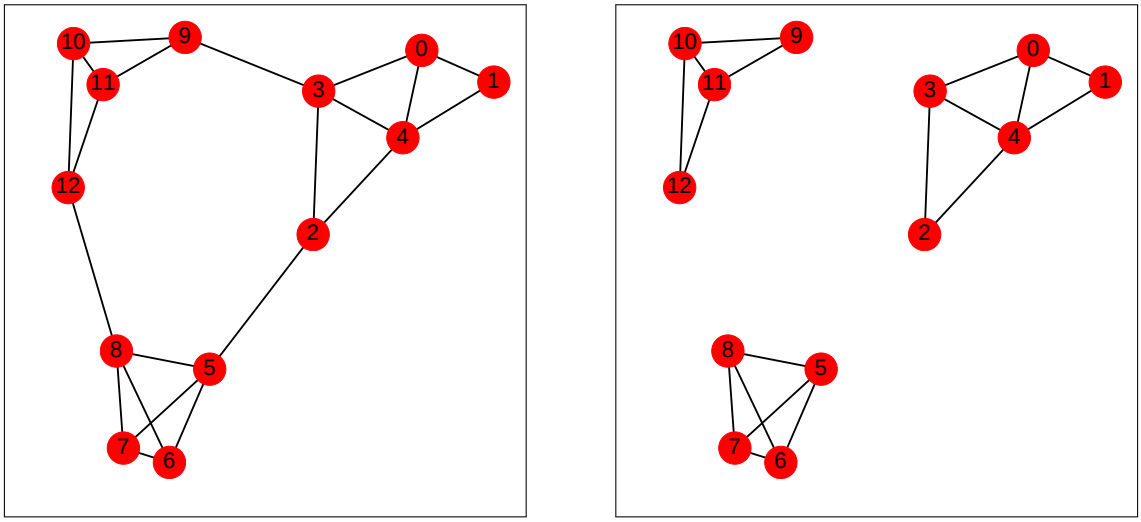
Figura 6.11: Aplicação do DBSCAN para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Pontos em preto são considerados ruídos pelo algoritmo. Valores de ε, em ordem, são: 1.50, 0.35, 0.80, 0.23, 0.20, 0.30. Valores de η, em ordem, são: 5, 5, 10, 50, 50, 5.

Uma limitação do DBSCAN é a sensibilidade aos parâmetros que devem ser fornecidos, especialmente ε, cuja uma pequena variação pode mudar drasticamente a quantidade e forma dos grupos. Não há um procedimento único e bem definido para a escolha desses parâmetros. Na prática, recorre-se a experimentação, tentativa e erro.

Agrupamento Espectral

O método de agrupamento espectral foi projetado para detectar grupos naturais em grafos. Apesar de ter sua fundamentação teórica voltada para tais objetos, é possível utilizá-lo para dados em uma organização matricial, assim como para os métodos descritos anteriormente. Inicialmente, será apresentada uma intuição do porquê do nome *espectral*, e então será discutido como a técnica pode ser utilizada quando o objeto de estudo não é um grafo. Por fim, será demonstrado como pode-se utilizar a técnica por meio da biblioteca *Scikit-learn* do Python.

Considere o exemplo da Figura 6.12a. Pode-se perceber que, apesar de ser conexo, o grafo em questão possui três "grupos" distintos: $\{0,1,2,3,4\}$, $\{5,6,7,8\}$ e $\{9,10,11,12\}$. Cada nó do grafo está mais conectado com outros nós dentro do que fora do grupo a que pertence. O objetivo do agrupamento espectral é justamente capturar essa intuição, ou seja, particionar um grafo em subgrafos (ou comunidades) que sejam "bem" conectados.



(a) (b)

Figura 6.12: (a) grafo conexo com estrutura em comunidades. (b) grafo não conexo.

Para motivar como a técnica funciona, considere o grafo da Figura 6.12b. O grafo consiste do mesmo exemplo da Figura 6.12a, apenas não contendo as arestas que conectam os diferentes grupos. Seja $\mathbf{A}$ a matriz de adjacências e $\mathbf{D}$ a matriz de graus do grafo em questão. A matriz Laplaciana do grafo é definida como sendo $\mathbf{L} = \mathbf{D} - \mathbf{A}$. No caso do exemplo, tem-se que que a matriz $\mathbf{L}$ é dada por:

$$\begin{pmatrix}
3 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & -1 & -1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2
\end{pmatrix}$$

Pode-se perceber que tal matriz tem uma estrutura bloco diagonal. Além disso, tem-se que os três seguintes vetores

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,$$
$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \text{ e}$$
$$\mathbf{z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

são autovetores de $\mathbf{L}$ associados ao autovalor 0.

De forma geral, se a matriz $\mathbf{L}$ possuir k autovetores (linearmente independentes) associados ao autovalor 0, então o grafo refente a tal matriz possui k componentes conexas. Esse fato em si não é útil na prática, pois raramente os grafos de interesse possuem a estrutura de comunidades formada por subgrafos desconexos (como na Figura 6.12b). No entanto, o exemplo acima apresenta uma intuição de que o espectro da matriz Laplaciana está fortemente relacionado com a estrutura de comunidades do grafo (se tal estrutura existir).

Os detalhes de como a técnica funciona estão além do escopo deste texto. O leitor interessado pode consultar o texto de [Zaki and Jr, 2014]. É importante ressaltar que a matriz Laplaciana (e sua versão normalizada) também pode ser construída para grafos cujas arestas têm peso. Esse fato permite que a técnica de agrupamento espectral seja aplicada a bases de dados arbitrárias, bastando para isso considerar que a matriz de afinidades entre os objetos do conjunto de dados seja interpretada como a matriz de adjacências de um grafo. Essa abordagem é seguida pela biblioteca *Scikit-learn*[10] cuja documentação também possui mais detalhes sobre a parte teórica do algoritmo. A aplicação do agrupamento espectral pela biblioteca *Scikit-learn* é muito similar aos casos anteriores. O código para importar as bibliotecas necessárias e gerar um conjunto de dados de exemplo é mostrado abaixo.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Para construir um modelo de agrupamento espectral, fornecendo o número de grupos desejados, e para aplicar utilizar o modelo em um conjunto de dados, tem-se:

```
In [5]: esp = cluster.SpectralClustering(n_clusters=3)
In [6]: esp.fit(X)
```

Por fim, os rótulos referentes às linhas da matriz $\mathbf{X}$ podem ser acessados da mesma forma que nos exemplos anteriores, por meio do atributo `labels_`.

```
In [7]: esp.labels_
Out[7]: array([1, 1, 1, ..., 1, 0, 0], dtype=int32)
```

[10] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

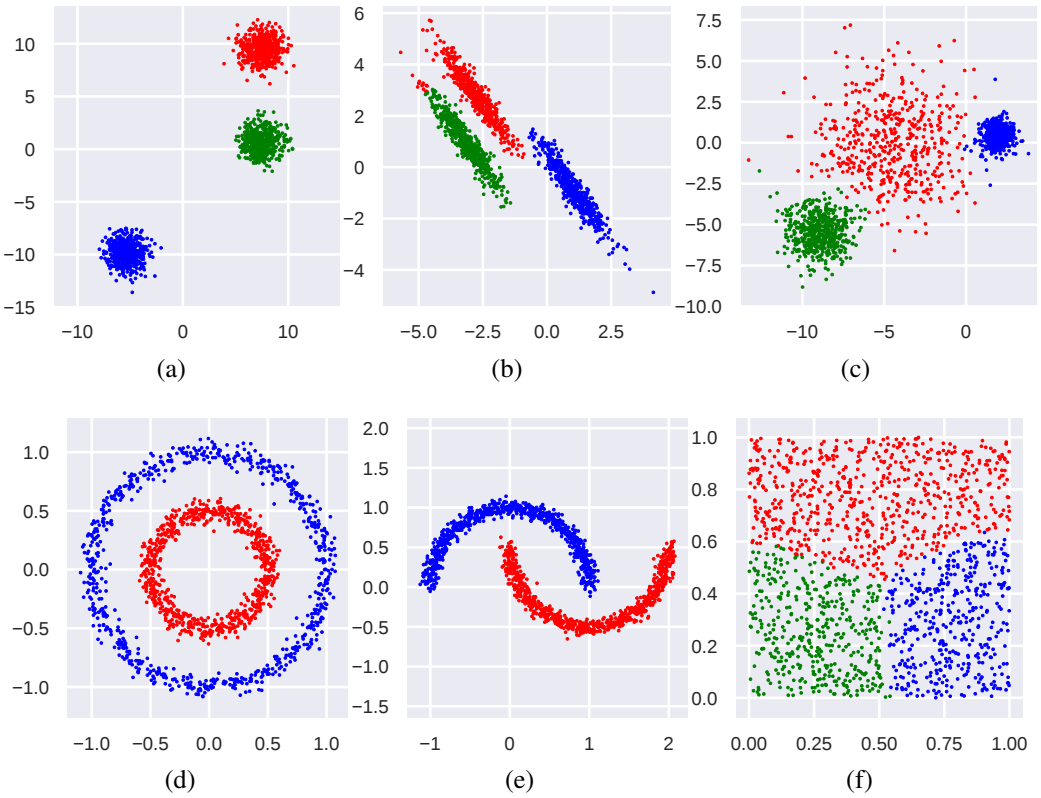


Figura 6.13: Aplicação do agrupamento espectral para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3. Além disso, o parâmetro `affinity='nearest_neighbors'` foi utilizado.

O resultado da aplicação do agrupamento espectral nos exemplos da Figura 6.6 é apresentado na Figura 6.13. Pode-se perceber que entre todas as técnicas exemplificadas, esta foi a que proporcionou melhores resultados.

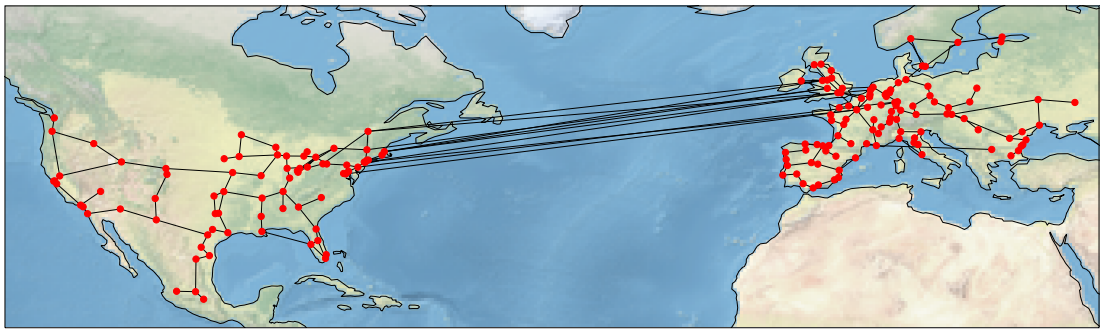
De forma geral, o agrupamento espectral é capaz de encontrar grupos significativos e próximos dos grupos naturais dos dados. Por outro lado, a técnica tem um custo computacional maior e também possui mais dificuldades para determinar o valor correto de k, i.e., o número de grupos desejado.

Exemplo: Detectando Comunidades em Redes

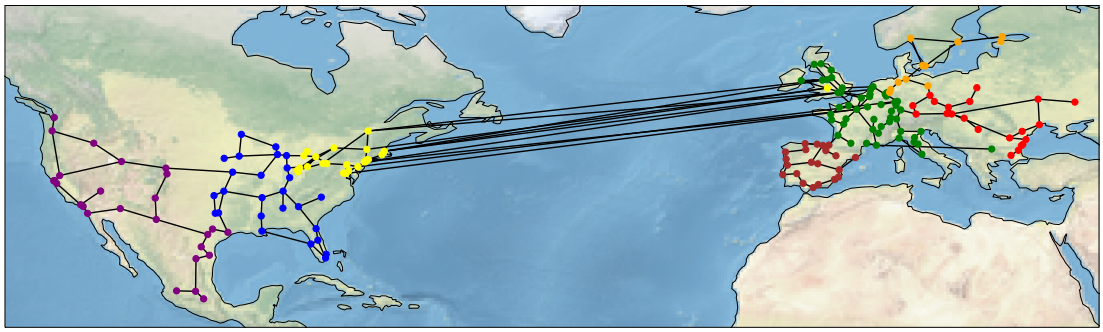
Em várias situações, dados provenientes de redes de computadores podem ser representados por meio de grafos. Exemplos são: topologias de redes (em diferentes camadas) e redes sociais. Neste contexto, uma tarefa comum de interesse consiste em encontrar partes da rede cujos elementos dentro de cada parte sejam "bem" conectados entre si, mas que sejam "fracamente" conectados com elementos de partes diferentes; em outras palavras, tem-se interesse em detectar comunidades na rede.

De forma geral, o problema de detecção de comunidades em grafos é desafiador, uma vez que até mesmo o conceito de "comunidade" é muito subjetivo. A seguir, tem-se

o objetivo de mostrar, por meio de um exemplo, que algumas das técnicas de agrupamento apresentadas nas seções anteriores podem ser utilizadas para o problema de detecção de comunidades em grafos, com resultados razoáveis.



(a)



(b)

Figura 6.14: (a) Grafo representando a rede da empresa Cogent em agosto de 2010. Cada nó do grafo representa um PoP da rede e cada aresta representa uma conexão entre PoPs (ao nível de IP). (b) Resultado da aplicação do agrupamento espectral no grafo anterior para $k = 6$. Cada cor representa um grupo diferente identificado pelo algoritmo.

A Figura 6.14a apresenta um exemplo de grafo baseado nos dados obtidos do projeto *The Internet Topology Zoo* [Knight et al., 2011]. Cada nó do grafo representa um ponto de presença (PoP) da rede da empresa Cogent (referente ao mês de Agosto de 2010) e cada aresta representa uma conexão entre PoPs (ao nível de IP). Além das informações topológicas, os dados contêm a geolocalização de cada PoP, o que permite estimar o custo de comunicação entre dois PoPs por meio da distância entre eles.

Como discutido anteriormente, o método de agrupamento espectral é fundamentado em teoria de grafos, podendo ser utilizado para encontrar comunidades em grafos de forma natural. A Figura 6.14b apresenta o resultado do agrupamento do grafo da Figura 6.14a utilizando $k = 6$ (i.e., seis grupos). Neste caso, a biblioteca *Scikit-learn* foi utilizada considerando a estrutura do grafo em si e também a distância física entre os nós do grafo. Pode-se perceber, por meio de uma inspeção visual, que os grupos identificados pelo algoritmo são correlacionados com regiões do mapa em comunidades distintas do grafo.

Esta parte explicou métodos que identificam padrões em dados não rotulados, foram reportados os algoritmos K-means, agrupamento aglomerativo, agrupamento baseado

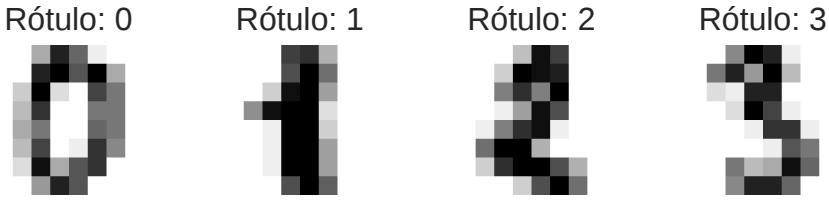


Figura 6.15: Exemplo de problema de classificação. Como associar cada imagem, referente a um dígito escrito à mão, ao número que ela representa? Nesse caso, cada imagem é representada por uma matriz 8×8, cujos valores definem uma escala de cinza. Essas imagens podem então ser representadas por vetores de 64 dimensões. Esse problema possui um total de 10 classes, uma vez que há 10 dígitos distintos. Exemplo baseado na documentação da biblioteca *Scikit-learn*.[11]

em densidade e agrupamento espectral, bem como um exemplo da aplicação deste último método para identificar diferentes regiões no globo terrestre, a partir das informações georreferenciadas dos PoPs e links de uma empresa de telecomunicações. Na sequência, serão apresentados métodos que aprendem a partir de dados previamente rotulados.

PARTE IV - Aprendizado Supervisionado

Aprendizado supervisionado é uma área de aprendizado de máquina que tem o objetivo de *aprender* (ou treinar) a partir de dados (vetor de características) rotulados, para então classificar novos dados em função do conhecimento adquirido no treinamento. Há duas tarefas principais: regressão e classificação. Nesse contexto, o objeto de estudo pode ser representado por meio de uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$ e por um vetor $\mathbf{y}$, onde:

- n é o número de objetos;
- d é a dimensionalidade de cada objeto; e
- y_i é o rótulo (ou valor) associado a $\mathbf{x}_i$ (i-ésima linha de $\mathbf{X}$).

Um exemplo do problema de classificação é ilustrado na Figura 6.15, o qual consiste da tarefa de associar imagens, referentes a dígitos escritos à mão, aos números que representam.

Dados $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, a tarefa de classificação, foco deste texto, consiste em encontrar um modelo f que aprenda, a partir dos vetores de características, os rótulos. No entanto, há requisitos importantes com relação ao modelo desejado; os dois principais são:

1. se $(\mathbf{x}, y) \in D$, deseja-se que $f(\mathbf{x}) = y$; e
2. a função aprendida deve ser *generalizável*. Se um novo $\mathbf{x}$ for observado (i.e, um $\mathbf{x}$ que não esteja em D), deseja-se que $f(\mathbf{x})$ seja o rótulo real de $\mathbf{x}$.

[11] `https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html`

Satisfazer apenas o requisito 1 é fácil. Satisfazer 1 e 2 simultaneamente, por outro lado, é uma tarefa difícil. Para ver que apenas o requisito 1 é fácil de ser satisfeito, considere o exemplo a seguir. Dados $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ e um vetor não rotulado que deseja-se classificar, $\mathbf{w}$, defina o seguinte classificador:

1. se $\mathbf{w}$ estiver em D, retorne a classe associada;

2. se $\mathbf{w}$ não estiver em D, retorne um rótulo aleatório (entre os valores válidos).

Observe que o requisito 1 é totalmente satisfeito, mas f generaliza de forma pobre, pois a chance de se acertar o rótulo real de $\mathbf{w}$ (quando este não estiver em D) é $\frac{1}{m}$, onde m é o número de classes. Esse exemplo também ilustra um dos maiores problemas em aprendizado de máquina: *overfitting*. Tal fenômeno ocorre quando o modelo explica tão bem o conjunto de dados usado durante o aprendizado, que é incapaz de generalizar para dados não vistos.

No restante do texto, são apresentados alguns dos classificadores mais populares e como utilizá-los por meio da biblioteca *Scikit-learn* do Python. Além disso, são mostradas técnicas de como evitar o problema de *overfitting* e uma aplicação no contexto de classificação de matrizes de tráfego, a qual é baseada no trabalho de [Trois et al., 2018].

Naive Bayes

A ideia do classificador Naive Bayes é criar um modelo probabilístico para encontrar a classe de um dado $\mathbf{x} \in \mathbb{R}^d$. Mais especificamente, tem-se interesse em calcular $P(C_k \mid \mathbf{x})$, onde C_k é a k-ésima possível classe ($k = 1, \ldots, K$) relativa ao problema de classificação. A quantidade $P(C_k \mid \mathbf{x})$ define a probabilidade de C_k ser a classe associada a $\mathbf{x}$. Assim, o modelo probabilístico é utilizado para determinar a classe de $\mathbf{x}$ como sendo aquela que maximize tal probabilidade.

Da teoria de probabilidade, e de uma maneira informal, sabe-se que a probabilidade condicional da classe C_k dado um vetor $\mathbf{X}$ é dada por

$$P(C_k \mid \mathbf{x}) = \frac{P(C_k, x_1, \ldots, x_d)}{P(\mathbf{x})}. \tag{7}$$

Na prática, tem-se interesse apenas no numerador dessa equação, uma vez que o denominador não depende de C_k. Daí,

$$\begin{aligned} P(C_k, x_1, \ldots, x_d) &= P(x_1, \ldots, x_d, C_k) \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2, \ldots, x_d, C_k) \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2 \mid x_3, \ldots, x_d, C_k) P(x_3, \ldots, x_d, C_k) \\ &= \ldots \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2 \mid x_3, \ldots, x_d, C_k) \ldots P(x_{d-1} \mid x_d, C_k) P(x_d \mid C_k) P(C_k) \end{aligned}$$

A hipótese ingênua do classificador, e daí o nome *Naive Bayes*, consiste de assumir que $P(x_i \mid x_{i+1},\ldots,x_d,C_k) = P(x_i \mid C_k)$. Dessa forma:

$$P(C_k \mid x_1,\ldots,x_d) \propto P(C_k,x_1,\ldots,x_d) = P(C_k)\prod_{i=1}^{d} P(x_i \mid C_k).$$

Para estimar o valor de $P(C_k \mid \mathbf{x})$, faz-se necessário saber apenas como calcular $P(x_i \mid C_k)$. Há várias formas de realizar tal cálculo. Geralmente, assume-se que os dados são provenientes de alguma distribuição de probabilidade (e.g., Normal, Multinomial e Bernoulli), e então, técnicas de inferência estatística são utilizadas para estimar os parâmetros de tal modelo. Após isso, o modelo pode ser utilizado para obter a classe de qualquer $\mathbf{x}$. O trabalho de [Metsis et al., 2006] apresenta detalhes de como esse procedimento pode ser realizado para diferentes distribuições de probabilidade, assim como uma aplicação do método para detecção de *spam* em *e-mails*.

Por fim, para estimar a classe de $\mathbf{x}$, denotada por $\hat{y}$, tem-se:

$$\hat{y} = \underset{z\in\{C_1,\ldots,C_K\}}{\operatorname{argmax}} \; P(z)\prod_{i=1}^{d} P(x_i \mid z)$$

A biblioteca *Scikit-learn* permite a utilização do classificador Naive Bayes de uma forma bem simples. O exemplo a seguir, mostra como treinar um modelo e utilizá-lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.naive_bayes import GaussianNB
```

Abaixo, o conjunto de dados referente ao conjunto de dados usado na Figura 6.15 é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: X
Out[4]:
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
In [5]: y
Out[5]: array([0, 1, 2, ..., 8, 9, 8])
```

Então, pode-se construir o classificador. Neste caso, utilizando uma versão que faz uso da distribuição Normal.

```
In [6]: model = GaussianNB()
In [7]: model.fit(X, y)
```

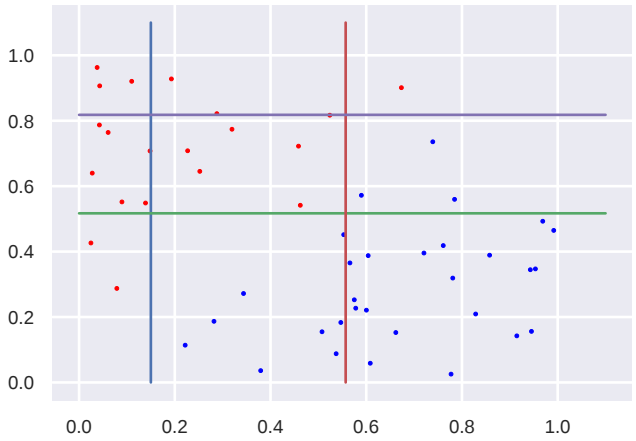


Figura 6.16: Aplicação de uma árvore de decisão em pontos no plano. Os pontos em azul são da classe 1, e os em vermelho da classe -1. As retas são as divisões feitas pela Árvore de Decisão treinada com esses pontos, mostrada na Figura 6.17.

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 8, ..., 8, 9, 8])
```

No exemplo acima, repare que a qualidade do classificador não foi avaliada. Além disso, apenas como título de exemplo, o método `predict` foi aplicado aos mesmos dados utilizados para a construção do modelo. Mais adiante, técnicas para avaliação e teste de modelos são apresentadas.

Árvores de Decisão

A ideia geral de uma Árvores de Decisão é criar uma árvore onde os nós internos são divisões no espaço de dados e os nós folha são rotulados com a classe mais frequente na região representada por esse nó. Como mostrado na Figura 6.16, o algoritmo de construção de uma Árvore de Decisão faz as divisões mais "puras" possíveis no espaço, de tal forma que os pontos estejam bem separados de acordo com suas respectivas classes. A Figura 6.17 mostra a Árvore de Decisão treinada com os pontos da Figura 6.16. Cada nó da Árvore de Decisão é um teste em algum atributo, e cada aresta é a saída daquele teste (verdadeiro ou falso). Note que, para classificar um novo ponto, basta caminhar na Árvore de Decisão verificando os condicionais até chegar em um nó folha, onde é dada a classe do ponto.

A forma mais utilizada para particionar o espaço de forma pura utiliza o GINI. Esse índice mede o quanto os dados são heterogêneos. O GINI é dado por: $1 - \sum_{i=1}^{d} p_i^2$, onde p_i é a frequência relativa de cada classe em cada nó. Quanto mais o GINI se aproxima de zero, mais o nó é considerado puro. Se o GINI se aproxima de 1, o nó é considerado impuro.

A biblioteca *Scikit-learn* permite a utilização do classificador Árvore de Decisão de uma forma bem simples. O exemplo a seguir, mostra como treinar um modelo e utilizá-

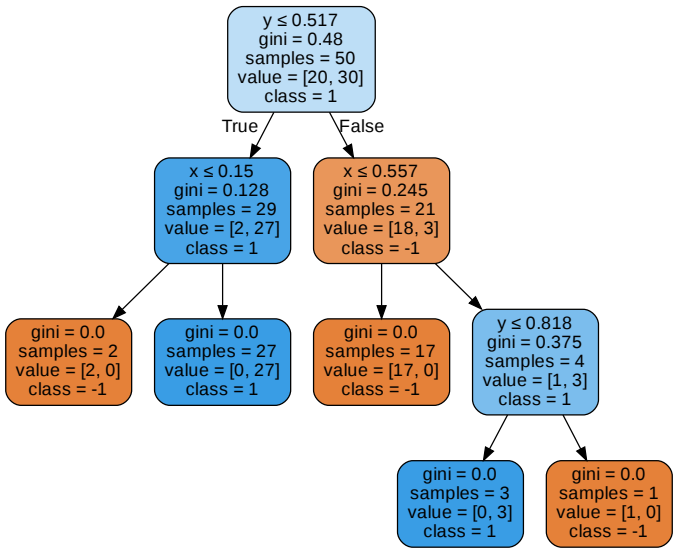


Figura 6.17: Árvore de Decisão construída com os pontos da Figura 6.16

lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.tree import DecisionTreeClassifier
```

Abaixo, o conjunto de dados referente ao conjunto de dados usado na Figura 6.15 é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: X
Out[4]:
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
In [5]: y
Out[5]: array([0, 1, 2, ..., 8, 9, 8])
```

Então, pode-se construir o classificador e treinar o modelo com base nos dados.

```
In [6]: model = DecisionTreeClassifier()
In [7]: model.fit(X, y)
```

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 2, ..., 8, 9, 8])
```

Florestas Aleatórias

Uma técnica bastante utilizada em classificação é a combinação de classificadores. É possível mostrar que, a combinação de n classificadores independentes utilizando o voto de maioria (a classe escolhida por pelo menos $\frac{n}{2}+1$ classificadores) melhora a acurácia da classificação. Na prática, é difícil ter uma quantidade grande de classificadores independentes, por isso são utilizados classificadores com baixa correlação.

A técnica de Florestas Aleatórias faz uso dessa filosofia, combinando uma sequência de Árvores de Decisão com baixa correlação. Para conseguir essa baixa correlação, aleatoriedade é introduzida no processo de criação das árvores, tanto na seleção dos objetos que serão utilizados para treinamento, como na seleção de atributos a serem utilizados como candidatos a critério de divisão em cada nó interno da árvore.

A biblioteca *Scikit-learn* permite a utilização do classificador Florestas Aleatórias. O exemplo a seguir mostra como treinar um modelo e utilizá-lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.ensemble import RandomForestClassifier
```

Abaixo, o conjunto de dados é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
```

Então, pode-se construir o classificador; neste caso, 100 árvores de decisão serão utilizadas.

```
In [6]: model = RandomForestClassifier(n_estimators = 100)
In [7]: model.fit(X, y)
```

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 2, ..., 8, 9, 8])
```

SVM – Support Vector Machine

O SVM (*Support Vector Machine*) é um dos classificadores mais populares da literatura. Tal popularidade se deve, principalmente, à boa qualidade dos resultados proporcionados pelo classificador em diversos tipos de problemas de classificação, até mesmo em problemas difíceis e não lineares.

Há duas versões principais do SVM, uma que gera um modelo linear, i.e., que divide o conjunto de pontos por meio um plano (ou hiperplano), e uma que gera um

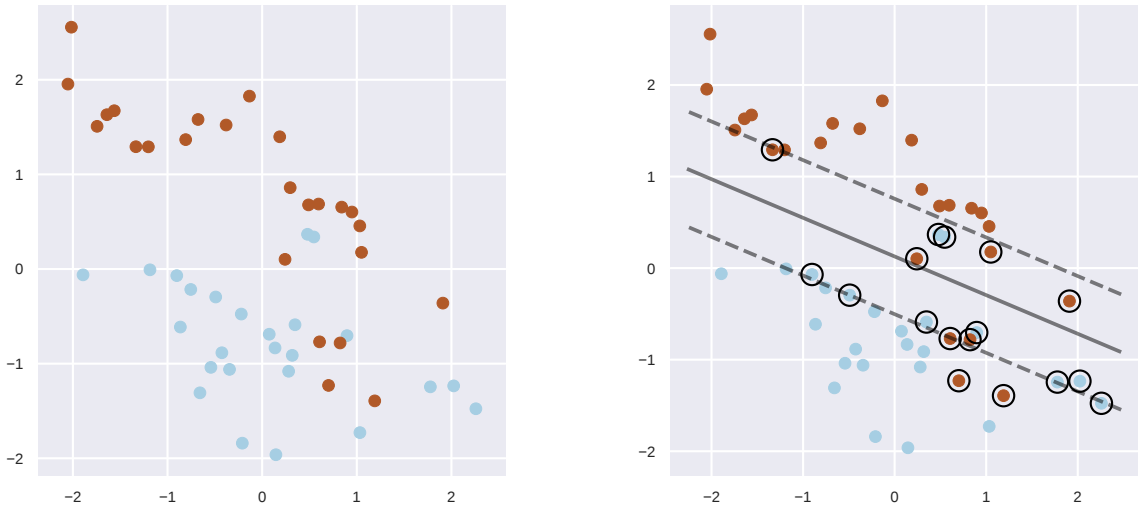


Figura 6.18: (a) conjunto de pontos a serem separados (i.e., classificados); (b) aplicação do classificador SVM ao conjunto de pontos. A linha sólida representa o plano de separação, as linhas tracejadas representam as margens do classificador e os pontos circulados são os vetores de suporte.

modelo não linear, a qual é necessária para problemas mais complexos. Aqui, apenas a versão linear será discutida, uma vez que a versão não linear é mais complexa e está fora do escopo do texto.

A Figura 6.18a mostra um conjunto de pontos pertencentes a duas classes distintas (indicadas pelas cores) a serem separados. Observe que os pontos podem ser separados por uma reta de forma quase perfeita. De fato, o objetivo do SVM é encontrar uma reta (ou um hiperplano no caso geral) que "melhor" separe o conjunto de pontos de acordo com as duas classes.

Mas o que significa "melhor" separar? No caso de um problema que pode perfeitamente ser separado por um modelo linear, o melhor modelo, ou hiperplano, é aquele que tem a maior margem, ou seja, que fica o mais distante possível dos pontos pertencentes as duas classes. Quando o problema não é linearmente separável, o melhor hiperplano é aquele que possui a maior margem, mesmo que a separação não seja perfeita, e que também minimize penalidades advindas de violações das margens. A ideia principal é que uma margem muito pequena implica na existência de vários modelos, dos quais, muitos são passíveis de generalização pobre. Por outro lado, uma margem muito grande pode penalizar bons modelos de uma forma desnecessária. Os conceitos de margem, hiperplano de separação e vetores de suporte são ilustrados na Figura 6.18b.

A dinâmica mencionada no parágrafo anterior é capturada pelo problema de otimização que o SVM pretende resolver. Dado um conjunto de treinamento $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$, com cada $\mathbf{x}_i \in \mathbb{R}^d$ $y_i \in \{-1, 1\}$, o objetivo é encontrar o hiperplano de separação $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, cujos parâmetros são obtidos resolvendo o seguinte problema de otimização:

$$\min_{\mathbf{w},b,\zeta} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\zeta_i,$$

com restrições

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \zeta_i \text{ e } \zeta_i \geq 0,\ i = 1,\ldots,n.$$

No problema acima, C é uma constante e deve ser fornecida pelo usuário. Tal constante influencia diretamente no tamanho da margem do classificador, e portanto, na qualidade do modelo resultante da solução do problema de otimização. Um valor grande de C tende a diminuir o tamanho da margem, ao passo que um valor pequeno de C tende a aumentar a margem. Qual o valor de C adequado? A resposta desta pergunta está diretamente relacionado ao fenômeno de *overfitting* e será abordada na próxima seção.

Na biblioteca *Scikit-learn*, no mesmo molde dos classificadores apresentados anteriormente, o SVM pode ser utilizado de forma conveniente. De início, é necessário importar as bibliotecas, como mostrado abaixo.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.svm import SVC
```

Após isso, deve-se criar o modelo, como os devidos parâmetros e ajustar tal modelo aos dados. No caso abaixo, cria-se um modelo *linear* com constante $C = 1$. A escolha deste C foi feita de forma arbitrária. Mais adiante, será apresentado como o valor correto de C deve ser selecionado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: model = SVC(kernel = 'linear', C = 1)
In [5]: model.fit(X, y)
```

Uma vez que o modelo é treinado, a função `predict` pode ser usada para predizer a classe de outros objetos.

```
In [6]: model.predict(X)
Out[6]: array([0, 1, 2, ..., 8, 9, 8])
```

Repare que no exemplo acima, o SVM foi utilizado para um problema de classificação não binária (há 10 classes nesse problema). Essa generalização não foi abordada neste texto. O leitor interessado pode consultar [Zaki and Jr, 2014].

Avaliação, Treino e Teste

No contexto de classificação, três perguntas importantes, e relacionadas, são:

1. Como avaliar um classificador?
2. Como obter os hiperparâmetros e um classificador (e.g., C do SVM ou o número de estimadores em uma floresta aleatória)?

3. Como evitar o problema de *overfitting*?

Quantificar o desempenho de um classificador é uma tarefa importante em. Além de possibilitar afirmar se um modelo é razoável ou não, tal tarefa é importante para a comparação de modelos diferentes. Neste contexto, considere $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ como sendo um conjunto no qual se pretende avaliar um classificador, denotado por μ, em um problema de classificação binária (i.e., $y_i \in \{0,1\}$).[12] De acordo com a classe real de cada $\mathbf{x}_i$, i.e., y_i, e a classe estimada de $\mathbf{x}_i$, i.e., $\mu(\mathbf{x}_i)$, definem-se as quatro seguintes quantidades:

- $TP = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 1 \text{ e } y_i = 1\}|$;
- $FP = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 1 \text{ e } y_i = 0\}|$;
- $FN = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 0 \text{ e } y_i = 1\}|$;
- $TN = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 0 \text{ e } y_i = 0\}|$.

Com essas quatro quantidades, é possível construir a *Matriz de Confusão* do classificador, assim como mostrado abaixo.

Predição \ Classe Real	Positiva	Negativa
Positiva	TP	FP
Negativa	FN	TN

Tabela 6.1: Estrutura de uma Matriz de Confusão. As colunas se referem as classes e as linhas as predições.

Tomando com base a Matriz de Confusão e as definições anteriores, há quatro métricas principais para avaliar ou comparar o desempenho de classificadores:

1. *Accuracy*:
$$\frac{TP+TN}{TP+TN+FP+FN},$$
i.e., do total de amostras, a fração que teve a classe corretamente predita;

2. *Precision*:
$$\frac{TP}{TP+FP},$$
i.e., dos que são preditos como pertencentes à classe positiva, a fração que realmente é pertencente à classe positiva;

3. *Recall*:
$$\frac{TP}{TP+FN},$$
i.e., dos que são pertencentes à classe positiva, a fração que é predita como pertencente à classe positiva;

[12] Assume-se que 1 é a classe positiva e 0 é a classe negativa.

4. *F-score*:
$$2\frac{pr}{p+r},$$
que é a média harmônica entre *Precision* (p) e *Recall* (r).

Como mencionado anteriormente, um bom classificador é aquele capaz de generalizar, ou seja, é esperado que tal classificador tenha um bom desempenho (de acordo com as métricas descritas) quando for utilizado em observações que não estiverem presentes no conjunto de treinamento. Mas dado um conjunto de treinamento $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, como garantir que um classificador treinado em D seja generalizável? Uma estratégia simples e bastante utilizada é "fingir" que uma parte de D não é conhecida! Em outras palavras, seleciona-se aleatoriamente uma fração dos pares em D (comumente por volta de 20%); denote essa porção D'. Após isso, treine o classificador escolhido em $D \setminus D'$. Use o conjunto não utilizado para treinamento (i.e., D') para testes. Se o classificador não tiver *overfitting*, então, ele deverá ter desempenho similar em D' e $D \setminus D'$.

Uma pergunta que ainda deve ser respondida é referente aos hiperparâmetros de um classificador. Por exemplo, como definir o C para o SVM ou o número de estimadores para uma floresta aleatória? Comumente, para evitar *overfitting*, faz-se uso de validação cruzada. Seja D o conjunto de treinamento (suponha que conjunto de testes já foi devidamente separado) e considere Θ como sendo um vetor de valores candidatos para os hiperparâmetros de um classificador escolhido. Então, proceda da seguinte maneira:

1. Particione, aleatoriamente, D em k partes: $D_1, \ldots, D_k$
2. Para cada $\theta \in \Theta$
   (a) Para cada $i \in \{1, \ldots, k\}$
      i. Treine o classificador em $D \setminus D_i$ e θ.
      ii. Avalie o classificador em D_i.
   (b) Calcule a média do desempenho dos k classificadores obtidos
3. Escolha o θ que proporcione os melhores resultados médios.

Repare que no procedimento acima, o particionamento de treino e teste é repetido novamente dentro do conjunto de treino. O objetivo desta etapa é, novamente, evitar *overfitting*.

A biblioteca *Scikit-learn* permite que todos os passos acima sejam de forma simples e com poucas linhas de código. A seguir, um exemplo mostrando um conjunto mínimo de etapas que devem ser seguidas para treinar um classificador de forma apropriada. O exemplo consiste de utilizar o SVM em um conjunto de dados de exemplos da própria *Scikit-learn*. Inicialmente, deve-se importar as bibliotecas necessárias, como mostrado abaixo.

```
In [1]: import numpy as np
In [2]: from sklearn.svm import SVC
In [3]: from sklearn.model_selection import GridSearchCV,
```

```
   ...:                                        train_test_split
In [4]: from sklearn import datasets
In [5]: from sklearn.preprocessing import MinMaxScaler
In [6]: from sklearn.metrics import confusion_matrix, accuracy_score,
   ...:                               precision_score, recall_score,
   ...:                               f1_score
```

O conjunto de dados de exemplo contém 569 observações, 30 atributos e duas classes. O método `train_test_split` permite dividir o conjunto de dados em porções distintas para treino e teste. Neste caso, 20% dos dados foram selecionados para teste.

```
In [7]: X, y = datasets.load_breast_cancer(return_X_y=True)
In [8]: XTrain, XTest, yTrain, yTest = train_test_split(X, y,
   ...:                                                 test_size = 0.2)
```

Uma prática comum em classificação consiste em normalizar a matriz de dados. Isso garante que todas as colunas da matriz de dados estejam na mesma ordem de grandeza. Há várias formas de conduzir esse processo. Abaixo, o `MinMaxScaler` é utilizado para colocar os valores de cada coluna da matriz de dados entre 0 e 1. Repare que objeto `scaler` é ajustado ao conjunto de treinamento.

```
In [9]: scaler = MinMaxScaler()
In [10]: scaler.fit(XTrain)
In [11]: XTrain = scaler.transform(XTrain)
```

Abaixo, 10 valores candidatos para o hiperparâmetro C são estabelecidos, os quais variam de 10^{-10} até 10^{10} em uma escala logarítmica.

```
In [12]: model = SVC(kernel = 'linear')
In [13]: cValues = np.logspace(-10, 10, 10)
In [14]: hParameters = {'C': cValues}
```

A classe `GridSearchCV` permite encontrar a melhor combinação de hiperparâmetros, neste caso apenas para C, por meio de busca exaustiva e validação cruzada.

```
In [15]: clf = GridSearchCV(model, hParameters, cv = 3)
In [16]: clf.fit(XTrain, yTrain)
```

Uma vez que a fase de treinamento é finalizada, é necessário testar o modelo no conjunto de testes. Para isso, é necessário primeiro aplicar a mesma transformação que foi aplicada ao conjunto de treinamento. Após isso, o método `predict` é utilizado para predizer a classe dos elementos do conjunto.

```
In [17]: XTest = scaler.transform(XTest)
In [18]: yPredicted = clf.predict(XTest)
```

Por fim, pode-se comparar a classe real dos objetos no conjunto de testes com as classes preditas. A seguir, um exemplo de como obter a matriz de confusão[13], *accuracy*, *precision*, *recall* e *F-Score*.

```
In [19]: confusion_matrix(yTest, yPredicted)
Out[19]:
array([[45,  2],
       [ 2, 65]])

In [20]: accuracy_score(yTest, yPredicted)
Out[20]: 0.9649122807017544

In [21]: precision_score(yTest, yPredicted)
Out[21]: 0.9701492537313433

In [22]: recall_score(yTest, yPredicted)
Out[22]: 0.9701492537313433

In [23]: f1_score(yTest, yPredicted)
Out[23]: 0.9701492537313433
```

Observe que o resultado final pode sofrer uma variação pequena, uma vez que o processo inicial de separar os conjuntos de treino e teste, bem como a validação cruzada, são feitos de forma aleatória. Alguns autores sugerem que todo o processo acima deve ser repetido algumas vezes para que a média e intervalos de confiança do desempenho do classificador no conjunto de teste também possam ser analisados.

Exemplo: Classificando Aplicações de Data Centers

Cada vez mais *data centers* e grandes *clusters* são utilizados para executar diversos tipos de aplicativos. Eles variam desde redes sociais e jogos até aplicativos com uso intensivo de computação, como indexação de conteúdo da Web, simulação de projetos de produtos, análise de dados provenientes de *crowdsourcing*. Outro tipo de software que demanda alto poder computacional são as aplicações científicas [Benson et al., 2010], como, por exemplo, previsão do tempo, previsão de desastres naturais, perfil bacteriano e genotipagem animal. Um aspecto interessante dessas aplicações é que a grande maioria apresenta padrões similares de computação e comunicação [Asanovic et al., 2006], o que significa que elas tendem a transmitir a mesma quantidade de informação entre os mesmos nós computacionais, independentemente dos dados de entrada.

Para melhorar a qualidade dos resultados, essas aplicações estão exigindo cada vez mais poder computacional, sendo executadas em sistemas de computacionais dedicados, demorando muitas horas para completar suas execuções e movimentando grandes volumes de dados, fazendo com que a rede se torne um gargalo. Então, entender as demandas de comunicação e classificar os aplicativos nos *data centers* é um desafio relevante [Srivastava et al., 2016]. A identificação correta de quais aplicativos estão usando os recursos do *data center* é essencial em várias atividades de gerenciamento, como planejamento de expansão, engenharia de tráfego, detecção de anomalias, monitoramento, e economia de energia [Trois et al., 2018].

[13]Na biblioteca *Scikit-learn*, as classes reais são as linhas e as preditas são as colunas

Existem três formas de classificar as aplicações. A primeira usa as portas de comunicação TCP para realizar essa tarefa; porém, como muitos aplicativos estão adotando a numeração dinâmica de portas, essa abordagem acaba sendo imprecisa. A segunda consiste em examinar o *payload* dos pacotes (*Deep Packet Inspection*) para classificar o tráfego. Essa abordagem não apenas impõe uma complexidade computacional significativamente maior, mas também requer conhecimento específico dos protocolos de aplicativos [Bujlow et al., 2013].

A terceira categoria aplica técnicas de aprendizado de máquina para classificar o tráfego. Algumas propostas exploram informações intrínsecas e estatísticas dos fluxo da rede para alimentar os classificadores [Eerman et al., 2006, Wang et al., 2010, Zhang et al., 2012, Fahad et al., 2014]; por exemplo, média e variação do tamanho do pacote, número total de pacotes ou bytes, duração do fluxo, números de porta do servidor e do cliente e muitos outros. Como aplicações de *data centers* apresentam padrões de comunicação [Asanovic et al., 2006], outra abordagem propõe extrair características de imagens geradas a partir da matriz de tráfego dos nodos computacionais [Trois et al., 2018].

Uma matriz de tráfego pode ser representada como uma matriz M_B, onde a entrada (i, j) de M_B contém o número de pacotes e/ou bytes transmitidos do nó i para o nó j. As matrizes de tráfego são normalizadas, formando uma imagem em tons de cinza, onde as células em preto são os pares de nós mais comunicativos e as células brancas indicam que nenhuma comunicação aconteceu. A Figura 6.19 mostra quatro exemplos de matrizes de tráfego coletadas de quatro aplicações científicas executadas em 16 nós computacionais.

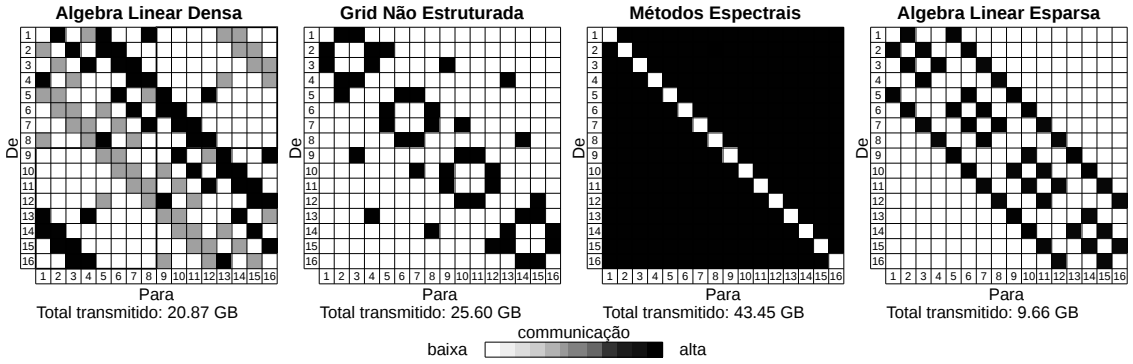


Figura 6.19: Matrizes de tráfego coletadas de 16 nós computacionais.

Para simplificar a implementação, as matrizes de tráfego serão transformadas em um vetor que será usado como vetor de característica do classificador[14]. As seções seguintes apresentam a leitura das imagens das matrizes de tráfego e a aplicação das funções de aprendizagem de máquina para reconhecer as aplicações.

[14]Será aplicada uma abordagem semelhante a este exemplo: `https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html`

Lendo as Matrizes de Tráfego

As matrizes de tráfego usadas no exemplo apresentado nessa seção foram coletadas do BlueCrystal Fase 3[15], um ambiente computacional de alta performance pertencente à Universidade de Bristol. Ele é composto de 223 *blades*, onde cada *blade* tem processador SandyBridge de 2,6GHz com 16 núcleos, 64GB de RAM e um disco SATA de 1TB. Além desses, há também 100 *blades* com GPGPUs duplos e 18 com maior capacidade de memória, onde cada *blade* contém 256GB.

As aplicações científicas foram executadas usando 128 nós computacionais, sendo capturada uma matriz de tráfego por segundo, durante 30 minutos. Serão usadas matrizes de tráfego de quatro aplicações científicas[16] que simulam o escoamento de fluidos por meio do método de Lattice Boltzmann, implementadas usando a biblioteca OpenLB [Heuveline and Latt, 2007]. A Figura 6.20 mostra exemplos das matrizes coletadas.

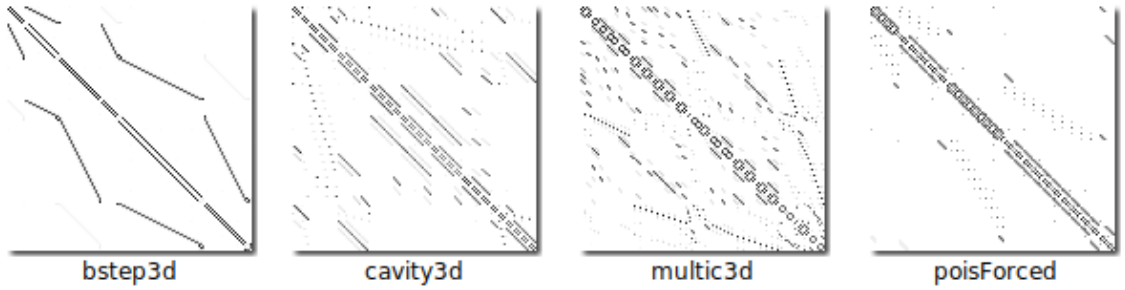


Figura 6.20: Matrizes de tráfego das aplicações científicas a serem classificadas.

É importante salientar que a maioria das aplicações possui várias fases de execução, por exemplo, fase de distribuição dos dados, fase de processamento e fase de sincronização. A Figura 6.21 mostra as matrizes de tráfego da aplicação científica `bstep3d`. Observa-se que nos tempos 380, 382 e 383, as matrizes de tráfego são semelhantes, pois representam a fase de processamento da aplicação. Por outro lado, no tempo 381 é feita a comunicação de sincronismo dos nodos, onde observa-se uma matriz bem distinta das demais.

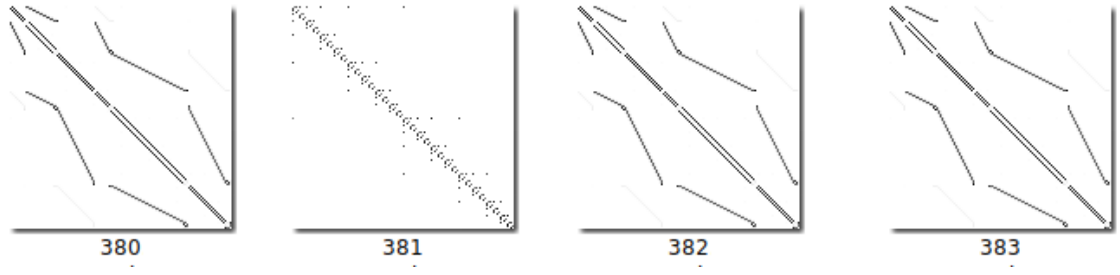


Figura 6.21: Matrizes de tráfego das fases de processamento e sincronização de uma aplicação científica.

[15]BlueCrystal Fase 3: `https://www.acrc.bris.ac.uk/acrc/phase3.htm`

[16]O dataset original, que apresenta 12 aplicações científicas além de diferentes fases de processamento do MapReduce, está disponível em: `www.inf.ufpr.br/ctrois/dctracs/dataset/`.

Para facilitar a implementação, as matrizes disponibilizadas neste exemplo foram filtradas, retirando-se as imagens que não correspondiam ao tráfego de processamento das aplicações. Primeiramente é criada uma lista com as aplicações que serão classificadas (linha 5); observe que o nome da aplicação deve ser igual ao diretório onde as fotos estão armazenadas.

```
In [1]: import glob # ler as imagens do diretorio
In [2]: import cv2 # converter a imagem em uma matriz python
In [3]: import numpy as np # converter a matriz em um vetor (reshape)

In [4]: # lista das aplicacoes que serão lidas
In [5]: aplicacoes=["bstep3d", "cavity3d", "multiComponent3d",
  ....:             "poiseuille3d"]
In [6]: # variavel auxiliar para nomear as classes
In [7]: classe=1

In [8]: # listas que armazenarão todas as imagens e respectivas classes
In [9]: matrizes=[]
In [10]: classes=[]
```

Em seguida, é feita a leitura das imagens. Foi usada a biblioteca glob, que retorna uma lista de todos os arquivos que estão dentro de um diretório. As imagens são lidas e transformadas para uma matriz Python pela função `cv2.imread`. Para usar os valores da matriz como vetor de características para o classificador, é necessário transformá-la em um vetor; isso é feito através da função `np.reshape` (linha 16).

```
In [11]: # percorre todas as aplicações
In [12]: for a in aplicacoes:
In [13]:     # percorre todas as imagens do diretorio
In [14]:     for i in glob.glob("./imagens/"+a+"/*"):

In [15]:         # le a imagem, converte para vetor e adiciona na lista
In [16]:         matrizes.append(np.reshape(cv2.imread(i,0),-1))
In [17]:         # inclui o numero da classe na lista de classes
In [18]:         classes.append(classe)

In [19]:     # proxima aplicacao: incrementa o numero da classe
In [20]:     classe+=1
```

Classificando as Aplicações

Uma vez que as imagens foram lidas e transformadas em um vetor (linha 16) e suas respectivas classes foram armazenadas (linha 18), serão aplicadas as funções de aprendizagem de máquina. Primeiramente será aplicada a função `train_test_split` para dividir as imagens aleatoriamente em dois conjuntos, 50% para treino e 50% para teste (parâmetro `test_size=0.5`).

```
In [21]:  from sklearn.model_selection import train_test_split
In [22]:  X_train, X_test, y_train, y_test = train_test_split(matrizes,
   ....:                                          classes, test_size=0.5)
```

Então, é criada uma instância do classificador `RandomForestClassifier` (linha 24) e o seu treinamento é realizado (linha 26).

```
In [23]: from sklearn.ensemble import RandomForestClassifier
In [24]: classifier = RandomForestClassifier(n_estimators=100,
   ....:                                     max_depth=2,random_state=0)
In [25]: # treina o classificador
In [26]: classifier.fit(X_train, y_train)
```

Finalmente é feita a classificação dos dados de teste `X_test` (linha 29) e o relatório de classificação e a matriz de confusão são apresentados (linhas 31 e 33).

```
In [27]: from sklearn import metrics

In [28]: expected = y_test
In [29]: predicted = classifier.predict(X_test)

In [30]: print("Classification report:\n%s\n"%
   ....:         (metrics.classification_report(expected, predicted)))
Out[31]:
Classification report:
             precision    recall  f1-score   support

          1       0.00      0.00      0.00       748
          2       1.00      1.00      1.00       836
          3       0.48      1.00      0.65       704
          4       1.00      1.00      1.00       857

avg / total       0.65      0.76      0.68      3145

In [32]: print("Confusion matrix:\n%s" %
   ....:       metrics.confusion_matrix(expected, predicted))
Out[33]:
Confusion matrix:
[[  0   0 748   0]
 [  1 835   0   0]
 [  0   1 703   0]
 [  0   0   0 857]]
```

Neste caso de estudo, foi apresentada uma forma de classificar aplicações usando suas matrizes de tráfego como entrada para algoritmos de aprendizagem de máquina. Conforme assertado por Trois et al. [Trois et al., 2018], os métodos existentes que aplicam aprendizagem de máquina para classificar aplicações usam o mesmo tipo de representação (estatísticas de fluxos), então existe espaço de pesquisa para investigar outras informações que permitam discriminar os diferentes tipos de tráfego de rede.

Algoritmos de aprendizagem supervisionada foram apresentados nessa parte do minicurso. Como exemplo, foi reportado uma forma de classificar as aplicações de centro de dados através da identificação dos padrões expressos nas matrizes de comunicação de seus nodos computacionais.

Conclusões e Perspectivas

O mercado profissional sofre modificações constantemente em função principalmente das novas demandas surgidas na sociedade e das novas tecnologias que vão surgindo. Esse contexto fez surgir uma nova profissão: o cientista de dados. A ciência de dados, também chamada de *data science*, é um reflexo do ambiente interconectado no qual vivemos e da enorme quantidade de dados disponibilizados e trafegados nas redes sociais e nas redes de computadores. A ciência de dados é uma área interdisciplinar que aborda as áreas de ciências exatas e engenharia, tais como programação, matemática e estatística.

No contexto das redes de computadores, o conhecimento da área de ciência de dados pode ajudar os administradores e gerentes de rede na tomada de decisões como implementação de soluções de engenharia de tráfego, planejamento de capacidade, detecção de tendências e anormalidades. Este minicurso teve como objetivo principal habilitar o profissional de redes com as competências básicas de um profissional da área de ciência de dados a partir de exemplos práticos contextualizados na área.

A abordagem adotada neste minicurso serviu para mostrar que, apesar de complexas, muitas técnicas podem ser utilizadas de forma simples por meio da linguagem de programação Python e algumas de suas bibliotecas. Foram abordados aspectos relacionados à análise exploratória de dados, com conteúdos de leitura, manipulação e visualização de dados e redução de dimensionalidade. Em seguida foi apresentado o conceito de aprendizado não supervisionado com exemplos de técnicas de agrupamento e detecção de comunidades. Por fim, o aprendizado supervisionado foi apresentado por meio de exemplos de diversas técnicas de classificação. A metodologia tradicional, que consiste em treino, validação e teste foi explicitada e contextualizada com exemplos práticos que podem ser aplicado no dia a dia do profissional de redes de computadores. Como resultado, espera-se que este minicurso tenha contribuído para capacitar os administradores, gerentes e pesquisadores de redes de computadores com o ferramental básico para realização de análise de dados a partir de suas redes e, com isso, essa informação auxilie suas tomadas de decisão e pesquisas.

Em se tratando de pesquisas acadêmicas e perspectivas futuras, a literatura traz diversos exemplos de como a ciência de dados pode ser aplicada em redes. Com o objetivo de mostrar como os resultados desse "casamento" pode auxiliar tomadas de decisão, reduzir custos e melhorar o desempenho das redes de computadores, em [Boutaba et al., 2018] é apresentado um apanhado bastante completo de como as técnicas apresentadas neste minicurso, aliadas às técnicas de aprendizado de máquina podem vêm sendo utilizadas na solução de diferentes desafios relacionados às redes de computadores.

Para motivar o leitor interessado em aprofundar-se na área e treinar seus conhecimentos com bases de dados reconhecidas e referenciadas, recomenda-se visitar o sítio eletrônico do CAIDA[17] (*Center for Applied Internet Data Analysis*) que realiza a coleta, curadoria e compartilhamento de dados para análise científica de tráfego na Internet, topologia, roteamento, desempenho e eventos relacionados à segurança. Todos os dados disponíveis no CAIDA, possuem *links* para descrições, formulários de solicitação de dados restritos, relatórios em tempo real e outros metadados.

[17] https://www.caida.org/data/

Agradecimentos

Este trabalho recebeu financiamento parcial do projeto Horizon 2020 (União Européia) sob no. 688941 (FUTEBOL), assim como do MCTI por meio da RNP e do CTIC. Além disso, este estudo foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, além de financiamento parcial proveniente de recursos de bolsas e projetos CNPq, FAPES e FAPEMIG. Os autores agradecem também à UFSM/FATEC pelo finaciamento parcial, através do projeto 041250 - 9.07.0025 (100548).

Referências

[Asanovic et al., 2006] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., et al. (2006). The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

[Benson et al., 2010] Benson, T., Anand, A., Akella, A., and Zhang, M. (2010). Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

[Bujlow et al., 2013] Bujlow, T., Carela-Español, V., and Barlet-Ros, P. (2013). Comparison of deep packet inspection (dpi) tools for traffic classification. Technical report, Universitat Politècnica de Catalunya.

[Carvalho et al., 2014] Carvalho, T., Junior, Marcos Simplicio, R. F. R. M., and Magri, D. (2014). Dmz científica: Desafios e modelos de gerenciamento de aplicações de alto volume de dados no contexto de e-ciência. In *SBRC 2014 - Minicursos*.

[Celes et al., 2017] Celes, C., De Oliveira Nunes, I., Borges, J., Silva, F., De Abreu COtta, L., Vaz de Melo, P., Ramos, H., Andrade, R., and Loureiro, A. (2017). *Big Data Analytics no Projeto de Redes Móveis: Modelos, Protocolos e Aplicações*, pages 1–58.

[Eerman et al., 2006] Eerman, J., Mahanti, A., and Arlitt, M. (2006). Internet traffic identification using machine learning techniques. In *Proc. of the 49th IEEE Global Telecomm. Conf.(GLOBECOM)*, pages 1–6.

[Fahad et al., 2014] Fahad, A., Tari, Z., Khalil, I., Almalawi, A., and Zomaya, A. Y. (2014). An optimal and stable feature selection approach for traffic classification based on multi-criterion fusion. *Future Generation Computer Systems*, 36:156–169.

[Forgy, 1965] Forgy, E. W. (1965). Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications. *Biometrics*, 21:768–769.

[Harari, 2016] Harari, Y. N. (2016). *Homo Deus: A brief history of tomorrow*. Random House.

[Heuveline and Latt, 2007] Heuveline, V. and Latt, J. (2007). The openlb project: an open source and object oriented implementation of lattice boltzmann methods. *International Journal of Modern Physics C*, 18(04):627–634.

[Knight et al., 2011] Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., and Roughan, M. (2011). The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775.

[Kurzweil, 2010] Kurzweil, R. (2010). *The singularity is near*. Gerald Duckworth & Co.

[Lakhina et al., 2005] Lakhina, A., Crovella, M., and Diot, C. (2005). Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228.

[McKinney, 2013] McKinney, W. (2013). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly, Beijing, first edition.

[Metsis et al., 2006] Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes?

[Roy and Vetterli, 2007] Roy, O. and Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610.

[Srivastava et al., 2016] Srivastava, G., Singh, M., Kumar, P., and Singh, J. (2016). Internet traffic classification: A survey. In *Recent Advances in Mathematics, Statistics and Computer Science*, pages 611–620. World Scientific.

[Trois et al., 2018] Trois, C., Bona, L. C., Oliveira, L. S., Martinello, M., Harewood-Gill, D., Fabro, M. D. D., Nejabati, R., Simeonidou, D., Lima, J. C. D., and Stein, B. (2018). Exploring textures in traffic matrices to classify data center communications. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 1123–1130.

[Trois et al., 2016] Trois, C., Fabro, M. D. D., de Bona, L. C. E., and Martinello, M. (2016). A survey on sdn programming languages: Toward a taxonomy. *IEEE Communications Surveys Tutorials*, 18(4):2687–2712.

[Wang et al., 2010] Wang, Y., Xiang, Y., and Yu, S. Z. (2010). Automatic application signature construction from unknown traffic. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 1115–1120.

[Zaki and Jr, 2014] Zaki, M. J. and Jr, W. M. (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, New York, NY, USA.

[Zhang et al., 2012] Zhang, H., Lu, G., Qassrawi, M. T., Zhang, Y., and Yu, X. (2012). Feature selection for optimizing traffic classification. *Computer Communications*, 35(12):1457–1471.