T03 Generate Random Data

16 de outubro de 2019

Ricardo A. Fernandes
Matrícula: 2019105350 (PPGEC/CTEC/UFAL)

Universidade Federal de Alagoas - UFAL
Instituto de Computação - IC
Programa de Pós-Graduação em Informática - PPGI

PPGI017-10 - Tópicos Especiais em Computação Visual e Inteligente: Aprendizagem Profunda
Para: Professor Tiago F. Vieira

---

1 Gereção de Dados Randômicos

Objetivo: Gerar dados randômicos de duas classes "+" e "o" usando numpy e matplotlib conforme ilustrado na Figura 3.

- use np.random.seed(1) para fixar a semente randômica e gerar distribuições mais próximas possíveis da Figura 3.
- Classe "o" tem média [0, 0].
- Classe "+" tem média [3, 4].

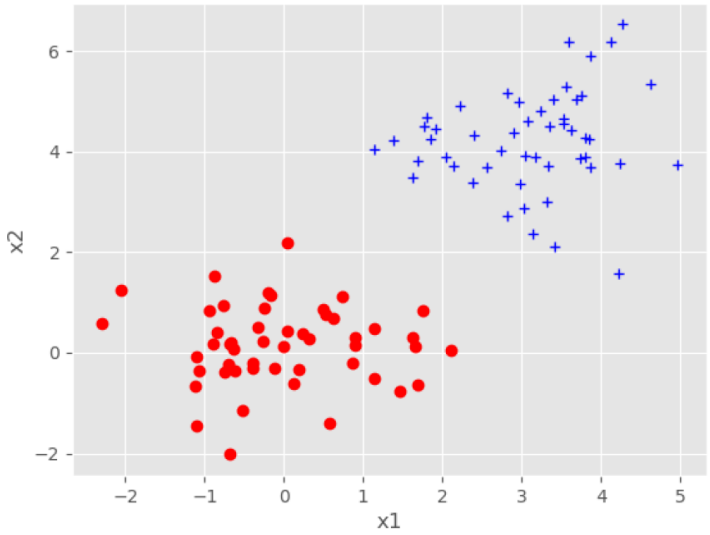

Figura 1: Ilustração das distribuições fornecidas para as classes "o" e "+".

1.1 Resolução

Desenvolveu-se a rotina generate_random_data.py (ver Listing 1) em que se define a classe "Data" que recebe em seu construtor: nome, média e símbolo. A classe "Data" implementa também o método "generate_random_sample" que recebe um número de amostras e retorna dois arrays: xs e ys. O método "random.normal" da numpy é utilizado para gerar os arrays xs e ys, distribuições normais referentes às médias recebidas em cada direção, considerando desvio padrão unitário e o número de amostras fornecido.

Dessa forma, são criados os objetos c e p, instâncias de "Data", representando as classes "+" e "o", respectivamente. Conforme orientação, utiliza-se a semente randômica 1 e chama-se o método "generate_random_sample" para ambos os objetos considerando 50 amostras. Os arrays retornados são plotados usando matplotlib, levando-se em consideração os nomes e símbolos previamente definidos.

Listing 1: Rotina em Python3: generate_random_data.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4
5  class Data:  # define Data class
6      def __init__(self, name, mean, mark):
7          self.name = name,
8          self.mean = mean
9          self.mark = mark
10
11     def generate_random_sample(self, n_samples):
12         xs = np.random.normal(self.mean[0], 1, n_samples)
13         ys = np.random.normal(self.mean[1], 1, n_samples)
14         return xs, ys
15
16
17 c = Data('o', [0, 0], 'ro')  # circle instance
18 p = Data('+', [3, 4], 'b+')  # plus instance
19
20 # lock random seed
21 np.random.seed(1)
22
23 # generate samples
24 xc, yc = c.generate_random_sample(50)
25 xp, yp = p.generate_random_sample(50)
26
27 # plot samples
28 plt.plot(xc, yc, c.mark, label=c.name)
29 plt.plot(xp, yp, p.mark, label=p.name)
30
31 # set options and show plot
32 plt.xlabel('x1'), plt.ylabel('x2'), plt.grid(True)
33 plt.legend(loc='lower right')
34 plt.show()
```

1.2 Comparação entre as distribuições

A Figura 2 ilustra as distribuições obtidas através da rotina generate_random_data.py utilizando o procedimento supracitado. Visualmente, observa-se boa concordância entre as distribuições geradas (ver Figura 2) e as fornecidas (ver Figura 3).

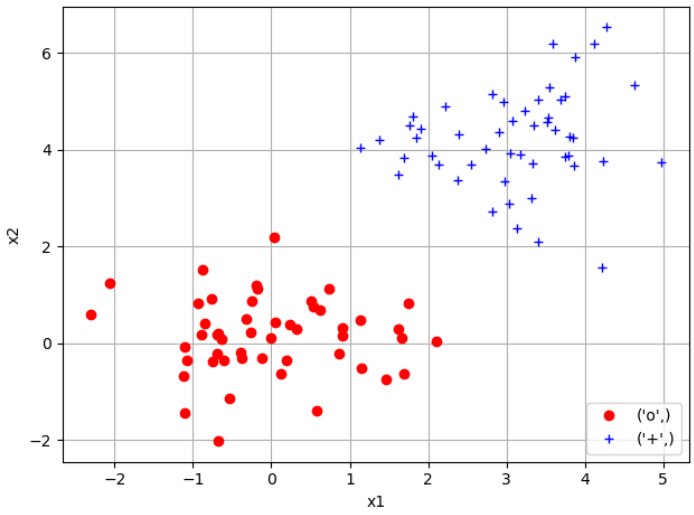

Figura 2: Ilustração das distribuições geradas para as classes "o" e "+".

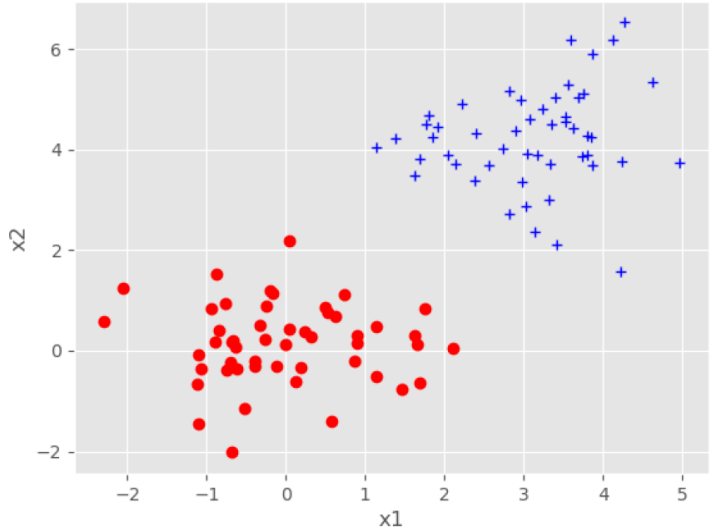

Figura 3: Ilustração das distribuições fornecidas para as classes "o" e "+".