گزارشکار آزمایش جلسه چهارم

Amirkabir University of Technology
(Tehran Polytechnic)

الگوریتم clustering و پیاده سازی الگوریتم Kmean

اعضای گزارش

عباس بدیعی

محمد حسین طیب زاده

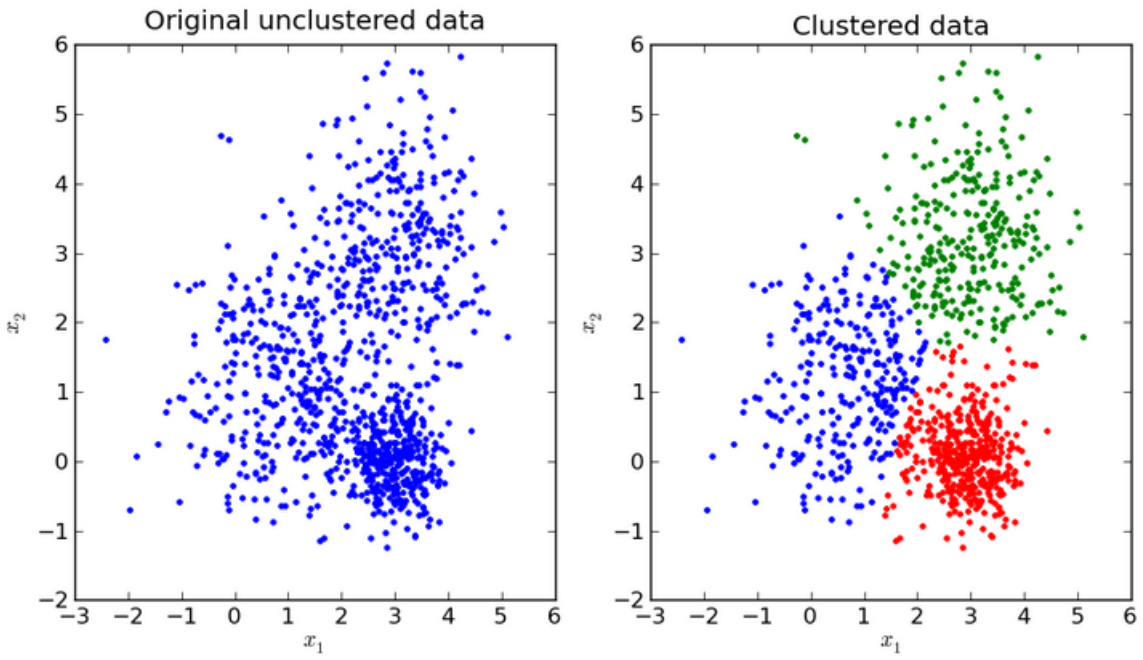
نویسنده گزارش : عباس بدیعی

بسم الله الرحمن الرحیم

در این آزمایش به الگوریتم clustering و پیاده سازی الگوریتم Kmean پرداخته ایم .

الگوریتم های Clustering برای خوشه بندی داده ها مورد استفاده قرار می گیرند ، این الگوریتم ها را می‌توان بر اساس مدل خوشه ای طبقه‌بندی کرد. یکی از الگوریتم های Clustering که در مبحث خوشه بندی مطرح می شود الگوریتم k-means می باشد ، این الگوریتم داده ها را به k ناحیه تقسیم می کند و براساس یک معیار شباهت آنها را طبقه بندی می کند.

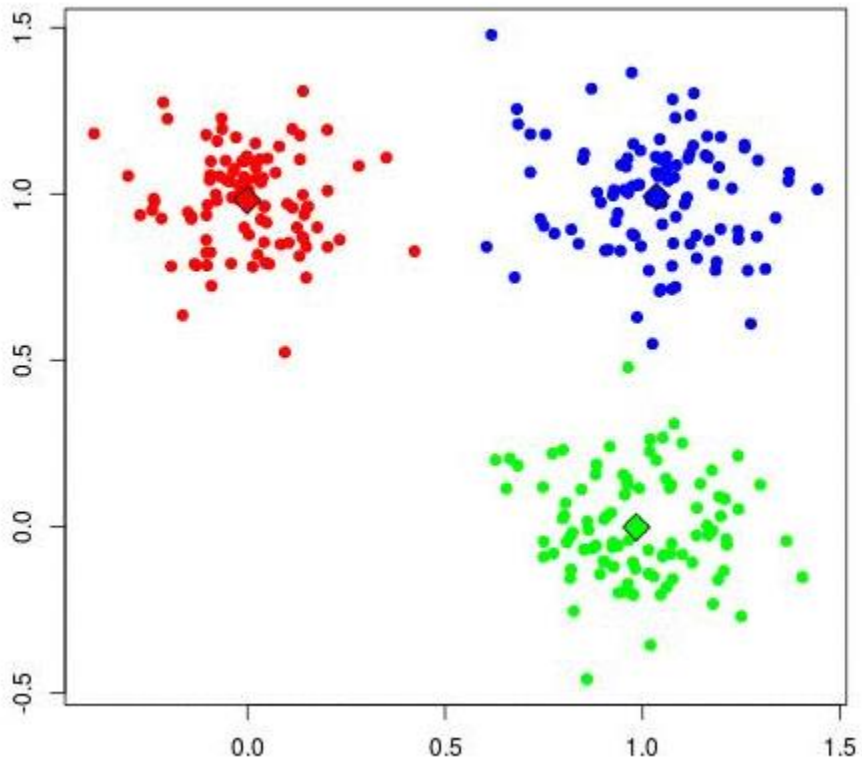
K-Means یک الگوریتم بسیار ساده است که داده‌ها را در K خوشه، گروه‌بندی می‌کند. نتیجه ای از خوشه‌بندی K-Means به شکل زیر است.



الگوریتمK-Means ، با فرض داشتن ورودی‌های x_n ، ...، x_3، x_2، x_1به شکل زیر کار می‌کند .

- گام اول: انتخاب K نقطه تصادفی به عنوان مرکز خوشه‌ها که به آن (مرکز دسته ها) گفته می‌شود.
- گام دوم: هر x_i به نزدیک‌ترین خوشه با محاسبه فاصله آن از هر مرکز تخصیص داده می‌شود.
- گام سوم: پیدا کردن مرکز خوشه‌های جدید با محاسبه میانگین نقاط تخصیص داده شده به یک خوشه
- گام ۴: تکرار گام ۲ و ۳ تا هنگامی که هیچ یک از نقاط تخصیص داده شده به خوشه‌ها تغییر نکنند.

(ارور از یک حد معین کاهش یابد).



گام اول

K مرکز خوشه (مرکزوار) به طور تصادفی انتخاب می‌شوند. فرض می‌شود که این مرکزوارها c_1 ،c_2 ،c_3 ،... ، c_k هستند. می‌توان گفت که:

$$C = c_1, c_2, \ldots, c_k$$

C مجموعه‌ای از همه مراکز خوشه هاست.

گام دوم

در این گام، هر مقدار ورودی به نزدیک‌ترین مرکز تخصیص داده می‌شود. این کار با محاسبه فاصله اقلیدسی (L2) بین نقطه و هر مرکزوار یا همان مرکز دسته انجام می‌شود.

$$\arg\min_{c_i \in C} dist(c_i, x)^2$$

که در آن dist() فاصله اقلیدسی است.

گام سوم

در این گام، مرکزخوشه های جدید با محاسبه میانگین کلیه نقاط اختصاص داده شده به هر خوشه محاسبه می‌شود.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

S_iمجموعه‌ای از همه نقاط تخصیص داده شده به خوشه i^{th} است.

گام چهارم

در این گام، مراحل ۲ و ۳ تکرار می‌شوند تا هیچ یک از نقاط تخصیص داده شده به خوشه‌ها تغییر نکنند. این یعنی تا هنگامی که خوشه‌ها پایدار شوند، الگوریتم تکرار می‌شود.

در کد مقدار $k=10$ می باشد. و کد مطابق تعریف های بالا به صورت زیر می باشد :

```
from MyKmean import Kmean

# from mpl_toolkits.mplot3d import Axes3D
# import matplotlib.pyplot as plt
# from matplotlib import cm
# from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

class Gaussian:
    def __init__(self ,x_mean , sigma):
        self.mean = x_mean
        self.sigma = sigma

    def dist(self , p1 , p2):
        dd = np.array(p1) - np.array(p2)
        return np. sqrt(dd[0]**2 + dd[1]**2)

    def eval(self, x):
        aa =  pow((1/(self.sigma*np.sqrt(2*np.pi)))*np.exp(-
0.5*(self.dist(x,self.mean))/(self.sigma)),2);
        # print(aa)
        return aa


class perceptron:
    def __init__(self , m):
        m = m+1
        # self.threshold = np.random.random()
        self.w =  np.random.random(m)
        self.eta = .25
        self.Err = 0

    def eval(self,x):
```

```
        x = np.array(x)
        x = np.append(x , -1)
        net = np.sum((x*self.w))
        return net

    def train__(self , x , d):
        x = np.array(x)
        y = np.append(x , -1)
        o = self.eval(x)
        self.w = self.w + 0.5*self.eta*(d - o)*y##*(1 - o**2)
        self.Err = self.Err + (d-o)**2;
        return self.Err

    def train(self , input_x , input_y , n):
        Emax = 3.1
        while(1):
            # print("1111")
            err = 0
            for i in range(n):
                x = input_x[i]
                y = input_y[i]
                err = self.train__(x,y)

            print(err)
            if err < Emax:
                break
            else:
                self.Error_reset()

    def Error_reset(self):
        self.Err = 0


class RBF:
    def __init__(self, RBF_Func_number ):
        self.RBF_Func_number = RBF_Func_number
        self.kmean = Kmean(RBF_Func_number,RBF_Func_number)
        self.m = RBF_Func_number
        self.perceptron = perceptron(self.m)
        # self.bias = np.random.random()

    def fit(self , input_data):
        self.kmean.train(input_data , iter_num = 10)
        self.centerOfClusters = self.kmean.getCenterOfCluster()
        print(self.centerOfClusters)
```

```
        c = self.centerOfClusters
        diff  = c[1]-c[0]
        d = np.sqrt((diff[0]**2 + diff[1]**2))

        # d = abs(max(self.centerOfClusters) - min(self.centerOfClusters))
        self.sigma = d/(np.sqrt(2*self.m))
        self.GaussianFunctions = []
        self.centerOfClusters = np.array([[3,2] , [8,7] , [9,3] , [2,8]])
        for c in self.centerOfClusters:
            self.GaussianFunctions.append(Gaussian(c , self.sigma))

    def train(self , x , d):
        input_data_new = []
        # print(x)
        for input_data in x:
            x_new = []
            # print(input_data[0])
            a1 = self.GaussianFunctions[0].eval(input_data)
            a2 = self.GaussianFunctions[1].eval(input_data)
            a3 = self.GaussianFunctions[2].eval(input_data)
            a4 = self.GaussianFunctions[3].eval(input_data)
            # a5 = self.GaussianFunctions[4].eval(input_data)

            # for i in range(self.m):
            #     # F = F + self.GaussianFunctions[i](self.w[i] * self.input_data
[i])
                # x_new.append(self.GaussianFunctions[i].eval(input_data[i]))
            input_data_new.append([a1,a2,a3,a4])
            # input_data_new.append([a1,a2,a3,a4,a5])
            # print([a1,a2])
        # print(input_data_new)

        x = np.array(x)
        self.perceptron.train(input_data_new , d, x.shape[0])

    def eval(self , data):
        F = []
        for input_data in data:
            a1 = self.GaussianFunctions[0].eval(input_data)
            a2 = self.GaussianFunctions[1].eval(input_data)
            a3 = self.GaussianFunctions[2].eval(input_data)
            a4 = self.GaussianFunctions[3].eval(input_data)
            x_new = [a1,a2,a3,a4]
            f = self.perceptron.eval(x_new)
```

```
            F.append(f)
        return np.array(F)

if __name__=="__main__":
    print("RBF ...")
    x1 = [[1,8],[2,9],[3,8],[2,7],[3,6],[4,8],[6,4],[7,1],[7,3],[7,4],[7,5],[8,1]
,[8,2],[8,3],[9,1],[9,2],[9,4],[10,2],[10,3],[10,4]]
    d1 = [1]*20
    x2 = [[1,1],[1,3],[2,1],[2,2],[3,1],[3,2],[3,3],[4,2],[4,3],[4,4],[5,4],[5,5]
,[6,7],[7,6],[8,8],[8,9],[9,7],[9,8],[9,9],[10,10]]
    d2 = [0]*20

    x = x1+x2
    d = d1+d2
    x = np.array(x)
    d = np.array(d)
    rbf = RBF(4)
    rbf.fit(x)
    rbf.train(x,d)
    print("end")
    print("x = [1.1,8.1]" + str(rbf.eval([[1.1,8.1]]))) ## 1
    print("x = [3.1,2.5]" + str(rbf.eval([[13.1,2.5]]))) ## 0
```

در ادامه می خواهیم با استفاده از این الگوریتم یک عکس رنگی به عکسی با حجم کمتر تبدیل کنیم :

```
# Python 2/3 compatibility
from __future__ import print_function

import numpy as np
import cv2 as cv

from gaussian_mix import make_gaussians

def main():
    cluster_n = 5
    img_size = 512

    # generating bright palette
    colors = np.zeros((1, cluster_n, 3), np.uint8)
    colors[0,:] = 255
    colors[0,:,0] = np.arange(0, 180, 180.0/cluster_n)
    colors = cv.cvtColor(colors, cv.COLOR_HSV2BGR)[0]

    while True:
        print('sampling distributions...')
        points, _ = make_gaussians(cluster_n, img_size)

        term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
        _ret, labels, _centers = cv.kmeans(points, cluster_n, None, term_crit, 10
, 0)

        img = np.zeros((img_size, img_size, 3), np.uint8)
        for (x, y), label in zip(np.int32(points), labels.ravel()):
            c = list(map(int, colors[label]))

            cv.circle(img, (x, y), 1, c, -1)

        cv.imshow('kmeans', img)
        ch = cv.waitKey(0)
        if ch == 27:
            break

    print('Done')


if __name__ == '__main__':
    print(__doc__)
    main()
    cv.destroyAllWindows()
```