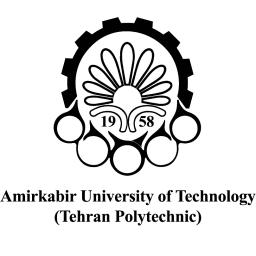
**گزارشکار آزمایش جلسه چهارم**



**الگوریتم clustering و پیاده سازی الگوریتم Kmean**

**اعضای گزارش**

**عباس بدیعی**

**محمد حسین طیب زاده**

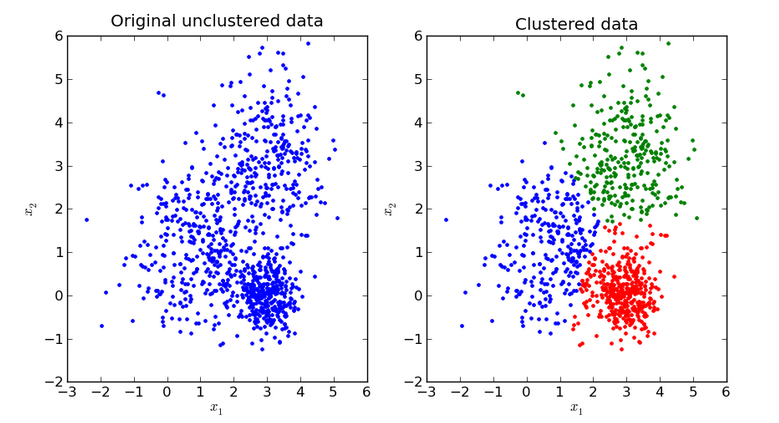
**نویسنده گزارش : عباس بدیعی**

بسم الله الرحمن الرحیم

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

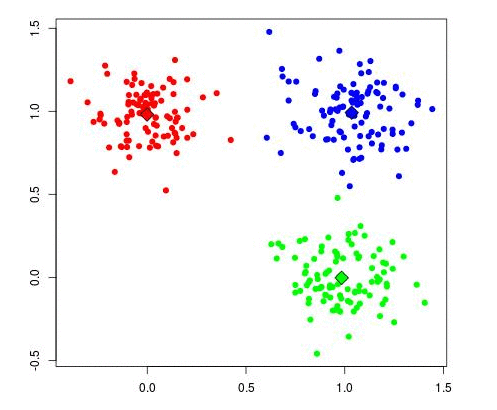
الگوریتم های Clustering برای خوشه بندی داده ها مورد استفاده قرار می گیرند ، این الگوریتم ها را می‌توان بر اساس مدل خوشه ای طبقه‌بندی کرد.  
یکی از الگوریتم های Clustering که در مبحث خوشه بندی مطرح می شود الگوریتم k-means می باشد ، این الگوریتم داده ها را به k ناحیه تقسیم می کند و براساس یک معیار شباهت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



الگوریتم K-Means، با فرض داشتن ورودی‌های xn ،… ،x3 ،x2 ،x1 به 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 xi 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**

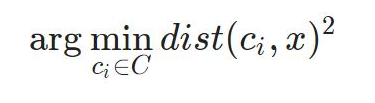
Kمرکز خوشه (مرکزوار) به طور تصادفی انتخاب می‌شوند. فرض می‌شود که این مرکزوارها ck،… ،c3،c2،c1 هستند. می‌توان گفت که:

C = c1, c2, …, ck

C مجموعه‌ای از همه مراکز خوشه هاست.

**گام دوم**

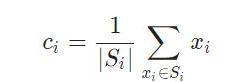
در این گام، هر مقدار ورودی به نزدیک‌ترین مرکز تخصیص داده می‌شود. این کار با محاسبه فاصله اقلیدسی (L2) بین نقطه و هر مرکزوار یا همان مرکز دسته انجام می‌شود.



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.



Si مجموعه‌ای از همه نقاط تخصیص داده شده به خوشه ith 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()