k-Means聚类算法

> 聚类算法有很多种，但是大家最熟悉、用的最多、最经典的聚类，就是k-Means算法了。这个算法很简单，简单到几乎不需要任何数学知识，简单到几十行代码就能实现。之所以介绍这个方法，就是因为它太常用了，而且效果还很好，所以下面就开始代码复现吧！

算法介绍

整个算法可以描述成这个样子：

step1: 从训练集D中选择k个样本作为初始均值向量
step2: 计算每个样本与各个均值向量之间的距离
step3: 根据距离最近的均值向量，确定样本所属的簇
step4: 将样本划入相应的簇
step5: 计算簇均值向量
step6: 更新簇均值向量
step7: 更新后的均值向量是否和原始均值向量一致？若一致则结束，否则返回step2

这段算法，理解起来并不困难。算法的名字叫k-Means，k是指簇的个数，means是指均值向量。整个算法干了这样一件事，输入你想聚成的簇的个数，然后不停迭代直到找到最为“合理”的均值向量，也就是簇的中心。这里的合理，其实就是最小化簇划分的平方误差。我们聚类希望得到的结果是簇内所有点都能围绕于簇中心，这也就是最小平方误差的含义，所以k均值所寻找的就是这样的一些中心。对于k值的选择，也是有很多的讲究，比如手肘法、轮廓系数法这些，由于我是在做代码复现，这部分就不再详细讲述了，有兴趣的小伙伴可以自行Google。

代码复现

算法的实现极为简单，代码中有详细注释，大家可以注意看一下里面的函数train，这部分是最为核心的内容。至于实验，我用了最常用的UCI的鸢尾花数据集，并且作图大致可视化了一下，结果是很好的。

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 28 22:07:14 2019

@author: zmddzf
"""
import random
import numpy as np

class KMeans:
    """
    k-Means算法类
    attributes:
        train: 训练k-Means聚类器
        predict: 预测一个样本所属的簇
    """
    def __init__(self, k, D):
        """
        构造器，传入簇个数(k)，训练集(D)，初始化中心
        :param k: 簇个数
        :param D: 训练集
        """
        self.k = k
        self.D = D
        self.Mu = random.sample(self.D, k = self.k)
        self.cluster = [i for i in range(self.k)]

    def __dist(self, P1, P2):
        """
        私有属性，计算两点距离
        """
        dist = 0
```

```
        for p1, p2 in zip(P1, P2):
            dist += (p1 - p2)**2
        dist = dist**0.5
        return dist

    def __computeMu(self, c):
        """
        私有属性，计算簇的中心点
        """
        c = np.array(c)
        mu = c.mean(axis = 0).tolist()
        return mu

    def train(self):
        """
        训练聚类器
        :return histMu: 每一轮迭代的簇中心点
        :return Mu: 训练完成后的簇中心点
        """
        histMu = [] #历史中心点
        while True:
            Mu_ = self.Mu.copy() #将中心点拷贝为训练前的中心点
            histMu.append(Mu_) #历史中心点数组加入
            C = [[] for item in range(self.k)] #初始化簇点集合

            for d in self.D:
                dist = [] #初始化点与各个中心点的距离数组
                for mu in self.Mu:
                    dist.append(self.__dist(d, mu)) #计算距离
                C[dist.index(min(dist))].append(d) #找出点所属的簇

            for i in range(self.k):
                self.Mu[i] = self.__computeMu(C[i]) #更新簇中心点

            if Mu_ == self.Mu:
                #如果中心点数组不再更新 则停止迭代
                break

        return histMu, self.Mu

    def predict(self, p):
        """
        对样本进行簇的预测
        :param p: 样本点数组
        :return cluster: 样本所属的簇
        """
        dist = []
        for mu in self.Mu:
            dist.append(self.__dist(p, mu))
        cluster = self.cluster[dist.index(min(dist))]
        return cluster
```

下面这一段是实验测试代码。

```
if __name__ == '__main__':
    from sklearn.datasets import load_iris #导入iris数据集函数
    import matplotlib.pyplot as plt #导入可视化库

    D = load_iris()['data'].tolist() #加载数据集
    km = KMeans(3, D) #实例化对象
    histMu, Mu = km.train() #训练

    #保存聚类结果
    clusters = []
    for d in D:
        clusters.append(km.predict(d))

    #可视化聚类结果
    D = np.array(D).T
    plt.scatter(D[2], D[3], c = clusters)
    plt.show()

    #可视化聚类中心的变动
    for mu in histMu:
        plt.scatter(np.array(mu).T[2], np.array(mu).T[3], c = [0,1,2])
    plt.show()
```