Kmeans算法简单理论和实现

原创 马雪峰 开源拾椹 2019-08-24

本文算是开源拾椹第一篇关于机器学习的文章，笔者水平有限，所以文章深度会比较搁浅。那今天给大家捋一捋机器学习的简单算法理论和实现。

机器学习主要是用来设计和分析一些可以让机器自动学习的算法，这些算法是从数据中自动分析获取规律，并且利用已经学习的规律来对未知数据进行预测，产生预测结果。而机器学习又分为监督学习，半监督学习，无监督学习和强化学习。笔者今天介绍一种无监督学习的一种简单算法Kmeans。Kmeans的算法理论是聚类中最常见的，也易于理解，运算速度快。但是只能应用于连续型的数据，并且在聚类前一定要声明分为几类。Kmeans的算法思想是人以群分，物以类聚。具体步骤如下。

1 输入k的值（k代表种类数量），即我们希望把数据划分为几类。
2 从数据中随机选择k个数据作为质心（每一个数据当做一个种类数据的代表）。
3 对于集合中的每一个数据，计算与各个质心之间的距离，跟哪一个质心越近，就划分为指定质心的种类。
4 此时每一个质心下面聚集了很多数据，此时每个种类的数据重新计算质心，即计算新的质心。
5 如果新的质心和旧的质心之间的距离小于设定的阈值，表示重新计算得出的质心位置变化不大，趋于稳定（收敛）可以认为我们的聚类已经达到了我们所预期的效果，算法终止。
6 如果新质心和旧质心之间的距离大于设定的阈值，那么迭代3~5步骤。
注意：新的质心选举并不是从数据集中直接选择一个质心，而是聚类中数据集合的均值。如果数据是二维的，那么新的质心应该是x坐标的均值和y坐标的均值，以此类推。

我们先利用代码生成几组随机数据并且生成可视化散点图

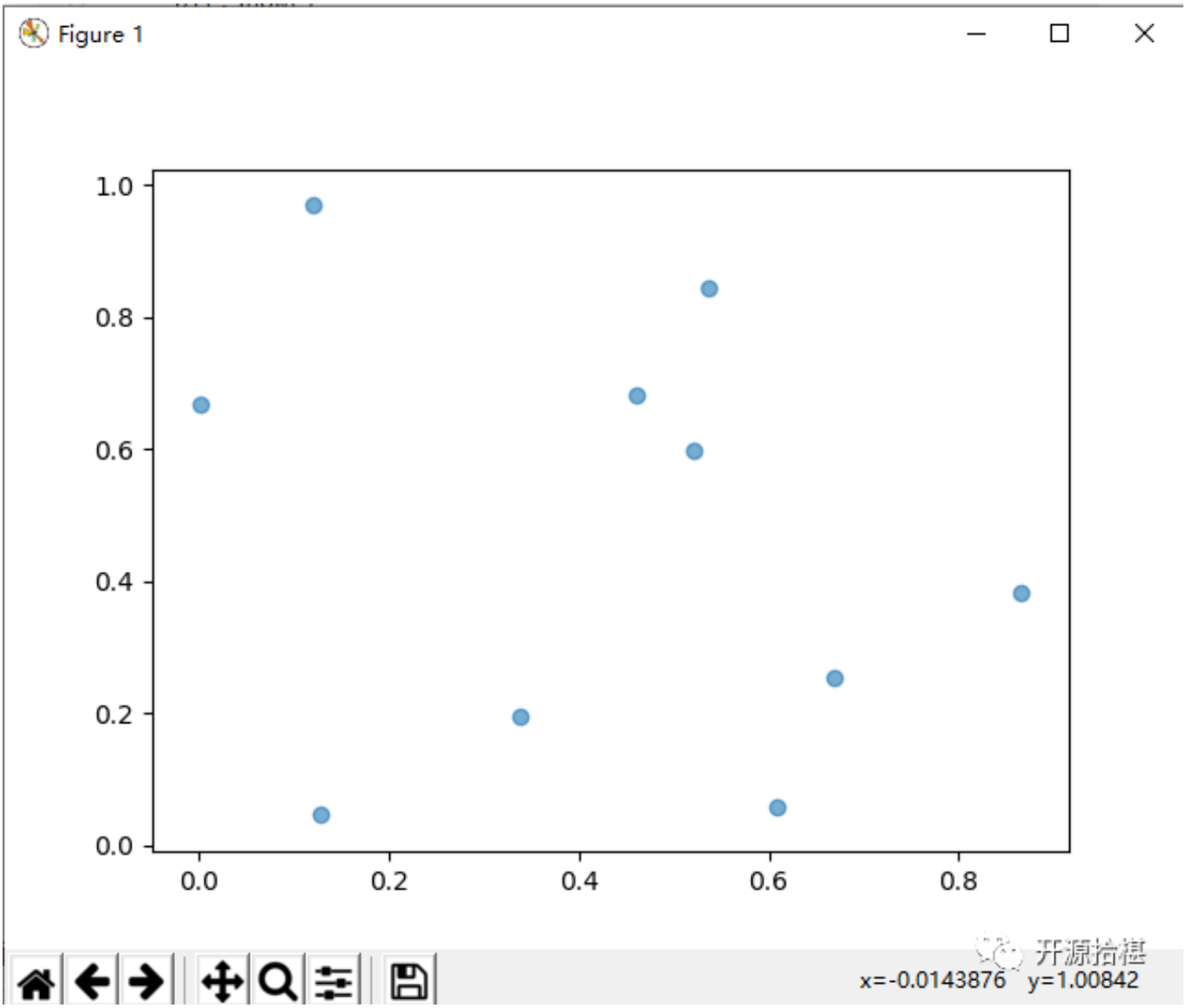
```
# -*- coding: utf-8 -*-
# @author:maxuefeng
import numpy as np
import matplotlib.pyplot as plt
def x_y():
    N = 10
    x = np.random.rand(N)
    y = np.random.rand(N)
```

```
    print(x)
    print(y)
    plt.scatter(x, y, alpha=0.6)
    plt.show()
if __name__ == '__main__':
    x_y()
```

得出随机测试数据如下

```
[1.27247485e-01 5.21309118e-01 5.08923602e-04 8.66144258e-01
 4.60536729e-01 1.20380895e-01 6.08852765e-01 6.69778492e-01
 3.37491693e-01 5.36524838e-01]
[0.04482642 0.5960499  0.66588187 0.38219811 0.68073081 0.96841576
 0.05869255 0.25329887 0.19501486 0.84297165]
```

散点图



1 选举K个质心

```
# 从数据集中先选出K个数据返回 也就是K个质心
def init(data, k):
    data = list(data)
    return random.sample(data, k)
```

2 计算距离并且分类

```
# 计算距离并且进行分类，根据不同质心的最短距离分类，使用字典保存
def min_dis(data, centroid_list):
    cluster_dict = dict()
    # 对属于每个data集合的item 计算item与centroid_list中k个质心的距离
    # 找出距离最小的，并将item加入响应的族类中
    k = len(centroid_list)
    for item in data:
        p1 = item
        flag = -1
        # 初始化为最大值
        min_dis = float("inf")

        for i in range(k):
            p2 = centroid_list[i]
            # 距离计算
            dis = calc_dis(p1, p2)
            if dis < min_dis:
                min_dis = dis
                # 循环结束  flag保存当前item最近的族标记
                flag = i

        if flag not in cluster_dict.keys():
            cluster_dict.setdefault(flag, [])
        cluster_dict[flag].append(item)  # append到对应的类别中

    return cluster_dict
```

2.1 距离计算

```
# 计算p1于p2之间的距离
```

```
def calc_dis(data, centroid_list):
    # 对每个属于dataSet的item， 计算item与centroidList中k个质心的距离，找出距离最小
    clusterDict = dict()  # dict保存簇类结果
    k = len(centroid_list)
    for item in data:
        p1 = item
        flag = -1
        minDis = float("inf")  # 初始化为最大值
        for i in range(k):
            p2 = centroid_list[i]
            dis = calc_dis(p1, p2)  # error
            if dis < minDis:
                minDis = dis
                flag = i  # 循环结束时， flag保存与当前item最近的蔟标记
        if flag not in clusterDict.keys():
            clusterDict.setdefault(flag, [])
        clusterDict[flag].append(item)  # 加入相应的类别中
    return clusterDict  # 不同的类别
```

3 重新计算质心

```
# 重新计算K个质心
def get_centroids(cluster_dict):
    import numpy as np
    centroid_list = []
    for key in cluster_dict.keys():
        centroid = np.mean(cluster_dict[key])
        centroid_list.append(centroid)
    # 得到新的质心的集合
    return centroid_list
```

4 计算各个族集合之间的均方误差

```
def get_var(centroid_list, cluster_dict):
    # 将族类中各个向量与质心的距离累加求和
    sum = 0
    for k in centroid_list.keys():
```

```
        p1 = centroid_list[k]
        dis = 0
        for item in cluster_dict[k]:
            p2 = item
            dis += calc_dis(p1, p2)
        sum += dis
    return sum
```

5 进行迭代

```
def load_data():
    import numpy as np
    dataSet = np.loadtxt("data.csv")
    return dataSet

def test():
    data = []

    centroid_list = init(data, 100)
    cluster_dict = min_dis(data, centroid_list)

    new_var = get_var(centroid_list, cluster_dict)
    old_var = 1  # 当两次聚类的误差小于某个值 说明质心基本确定

    time = 2

    while abs(new_var - old_var) >= 0.00001:
        centroid_list = get_centroids(cluster_dict)
        cluster_dict = min(data, centroid_list)
        old_var = new_var
        new_var = get_var(centroid_list, cluster_dict)
        time += 1

        show_cluster(centroid_list, cluster_dict)


def show_cluster(centroid_list, cluster_dict):
    import matplotlib.pyplot as plt
    # 展示聚类结果
    # 不同簇类标记，o表示圆形，另一个表示颜色
```

```
    colorMark = ['or', 'ob', 'og', 'ok', 'oy', 'ow']
    centroidMark = ['dr', 'db', 'dg', 'dk', 'dy', 'dw']

    for key in cluster_dict.keys():
        plt.plot(centroid_list[key][0], centroid_list[key][1], centroidMark[k
        for item in cluster_dict[key]:
            plt.plot(item[0], item[1], colorMark[key])
    plt.show()


if __name__ == '__main__':
    test()
```

Kmeans 算法Java实现。

```
package com.planet.data.basic.algorithm;

import org.apache.http.util.Asserts;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public abstract class KmeansTemplate<T> {

    // 待分类的原始值
    private List<T> originData;

    // 将要分成的类别个数
    private int K;

    // 最大迭代次数
    private int maxIterationTimes;

    // 聚类的结果
    private List<List<T>> clusterList;

    // 质心
```

```
24      private List<T> centroid;
25
26      public KmeansTemplate(List<T> originData, int maxIterationTimes, List<T>
27
28          // 检查为空
29          Asserts.check(originData == null || originData.size() == 0, "origin
30
31          //
32          Asserts.check(K >= maxIterationTimes || K < 2, "Please check you par
33
34          this.originData = originData;
35          this.maxIterationTimes = maxIterationTimes;
36          this.centroid = centroid;
37      }
38
39      /**
40       * @return
41       * @Author:lulei
42       * @Description: 对数据进行聚类
43       */
44      public List<List<T>> clustering() {
45          if (originData == null) {
46              return null;
47          }
48          //初始K个点为数组中的前K个点
49          int size = K > originData.size() ? originData.size() : K;
50          List<T> centerT = new ArrayList<T>(size);
51          //对数据进行打乱
52          Collections.shuffle(originData);
53          for (int i = 0; i < size; i++) {
54              centerT.add(originData.get(i));
55          }
56          clustering(centerT, 0);
57          return clusterList;
58      }
59
60      /**
61       * @param preCenter
62       * @param times
63       * @Author:lulei
```

```
     * @Description: 一轮聚类
     */
    private void clustering(List<T> preCenter, int times) {
        if (preCenter == null || preCenter.size() < 2) {
            return;
        }
        //打乱质心的顺序
        Collections.shuffle(preCenter);
        List<List<T>> clusterList = getListT(preCenter.size());
        for (T o1 : this.originData) {
            //寻找最相似的质心
            int max = 0;
            double maxScore = similarScore(o1, preCenter.get(0));
            for (int i = 1; i < preCenter.size(); i++) {
                if (maxScore < similarScore(o1, preCenter.get(i))) {
                    maxScore = similarScore(o1, preCenter.get(i));
                    max = i;
                }
            }
            clusterList.get(max).add(o1);
        }
        //计算本次聚类结果每个类别的质心
        List<T> nowCenter = new ArrayList<T>();
        for (List<T> list : clusterList) {
            nowCenter.add(getCenterT(list));
        }
        //是否达到最大迭代次数
        if (times >= this.maxIterationTimes || preCenter.size() < this.K) {
            this.clusterList = clusterList;
            return;
        }
        this.centroid = nowCenter;
        //判断质心是否发生移动，如果没有移动，结束本次聚类，否则进行下一轮
        if (isCenterChange(preCenter, nowCenter)) {
            clear(clusterList);
            clustering(nowCenter, times + 1);
        } else {
            this.clusterList = clusterList;
        }
    }
```

```
104
105     /**
106      * @param size
107      * @return
108      * @Author:lulei
109      * @Description: 初始化一个聚类结果
110      */
111     private List<List<T>> getListT(int size) {
112         List<List<T>> list = new ArrayList<List<T>>(size);
113         for (int i = 0; i < size; i++) {
114             list.add(new ArrayList<T>());
115         }
116         return list;
117     }
118
119     /**
120      * @param lists
121      * @Author:lulei
122      * @Description: 清空无用数组
123      */
124     private void clear(List<List<T>> lists) {
125         for (List<T> list : lists) {
126             list.clear();
127         }
128         lists.clear();
129     }
130
131     /**
132      * @param value
133      * @Author:lulei
134      * @Description: 向模型中添加记录
135      */
136     public void addRecord(T value) {
137         if (originData == null) {
138             originData = new ArrayList<T>();
139         }
140         originData.add(value);
141     }
142
143     /**
```

```
144         * @param preT
145         * @param nowT
146         * @return
147         * @Author:lulei
148         * @Description: 判断质心是否发生移动
149         */
150        private boolean isCenterChange(List<T> preT, List<T> nowT) {
151            if (preT == null || nowT == null) {
152                return false;
153            }
154            for (T t1 : preT) {
155                boolean bol = true;
156                for (T t2 : nowT) {
157                    if (equals(t1, t2)) {//t1在t2中有相等的，认为该质心未移动
158                        bol = false;
159                        break;
160                    }
161                }
162                //有一个质心发生移动，认为需要进行下一次计算
163                if (bol) {
164                    return bol;
165                }
166            }
167            return false;
168        }
169
170        /**
171         * @param o1
172         * @param o2
173         * @return
174         * @Author:lulei
175         * @Description: o1 o2之间的相似度
176         */
177        public abstract double similarScore(T o1, T o2);
178
179        /**
180         * @param o1
181         * @param o2
182         * @return
183         * @Author:lulei
```

```
184         * @Description: 判断o1 o2是否相等
185         */
186        public abstract boolean equals(T o1, T o2);
187
188        /**
189         * @param list
190         * @return
191         * @Author:lulei
192         * @Description: 求一组数据的质心
193         */
194        public abstract T getCenterT(List<T> list);
195    }
```

结语：机器学习算法我们平时尽量不要引用第三方包，当然第三方包给我们的工作带来了很多便利，但是使用numpy，panda等数学库推导公式可以增强我们对机器学习算法的理解。感谢各位读者用心阅读。