C++实现K-means，聚类原理解析（并用在图片像素点聚类）

标签： K-means聚类 C++程序实现 图片像素点聚类

2016-05-21 09:14 2478人阅读 评论(0) 收藏 举报

分类： C/C++学习（15） 机器学习（9） opencv（8）

最近用到图像中的点的聚类，于是就写了一个k-means的类。

验证的过程是将一幅图的所有点的(B, G, R)作为数据点，进行聚类。

算出K个中心类后，对图像中的每个点进行重新上色。按照类别给给每类生成一种随机色彩。

使用该类，可以自定义聚类中心K的个数、数据维度N的大小。

数据类型可以是float、int。

同时在迭代过程中，可以选择输出每次迭代的中心点信息等。

完整的工程文件在：https://github.com/SunnyCat2013/toy-k-means.git

下面分以下几个部分：

一，K-means的思路

二，基本公式与程序实现细节

三，参考

四，图像处理结果

K-means 算法是一种简单有效的无监督学习方法，它可以有效地将多维空间（用N表示）中的点聚成一个个紧密的簇。

K-means算法的优化目标是使求出K个中心点，使每一个点到该点的欧氏距离平方之和尽量小（不知道现在有没有什么算法能保证一定得到全局最优解）。

简单来说就是把一个分到一个类中的所有数据点的每一维相加，得一个向量。然后，该向量的每一维除以该类的点的个数。这样得的向量就是该类的中心(centroid).

算法的思路如下：

1. 初始化K个中心点。

这K个点可以是在所有输入数据点中随机抽取的，也可以是取前K个点，也可以是从N维空间中任意一个点。这些点和数据点之间的距离只要不是相差的太离谱都没有关系。

2. 对任意一个数据点，求与它最近的中心点，并认为该数据点属于该中心点所代表的类。对于M(假设共有M个数据点）个数据点，分别计算每个点与K个当前的中心点的欧氏距离平方值，点x_i与哪个中心点（如c_j）的欧氏距离平方最小那么它就分成该类。（该过程可以求出一些指标，用于终止程序。如，求出整体欧氏距离之和）

一般暴力的方法是要计算M * K次欧氏距离。《An Efficient k-Means Clustering Algorithm:Analysis and Implementation》提供了一种利用KD树的超子空间到K个点心点的距离对中心点进行减枝的方法。它的主要思想是，将一个KD树中的子空间作为一个超球面的中心点，以K个中心点中到子空间中心最近的距离为半径形成了一个新的超球面，用这个超球面将当前的K个中心点分成两部分：与该子空间的最近中心点的待选集合与不可能是该子空间中任意一点的最近中心点的集合。

3. 更新每个类的中心点。

4. 由 2 得出的指标判断是否可以终止：否，进行 2 ；是，终止，并给出中心点信息。

程序实现细节：

0. K-means类定义。这个程序的一部分目的也是为了测试类模版，原本想兼容多种数据类型（float、int），到最后才发现主要还是对float类型做处理。(所以使用类模版在这个程序中有点鸡肋。完全可以用float代替。维度N也是同样的，维度N和待求中心点的个数K都可以在类声明时候定义）

```cpp
template <typename T, int N> class KMeansTest
{
private:
    vector<vector<T>> centers;// 保存中心点
    bool data_is_ok(vector<vector<T>> points, int k);
    T calculate_center_once
(vector<vector<T>> points);// calculate centers for once, and return the total sum
    int k_centers = 0;
    T Total_Sum = 0; // sum of square, or square root of sum
    T Total_Sum_Sqrt = 0; // sum of square, or square root of sum
public:
    int end_modal = 0;
    bool random_initial_centers = false; // trye: get random points as centers; else, get top k po
    bool show_info_flag = false; // true: show kmeans object centers' information; false: show out
    KMeansTest();
    KMeansTest(vector<vector<T>> points, int k);
    void init_k(vector<vector<T>> points, int k);
    int get_centers(vector<T> point); // return the center index and center point
    void show_info();
    void sort_ascend(); // sort average of each point in ascend order.
};
```

1. 初始化。我设置了两种可以初始化的方式：取前K个点；随机取K个点。代码如下：

```cpp
if (random_initial_centers) {

    for (int i = 0; i < k; i++) {
        int iSecret;

        srand (time(0)+i);// use current time as seed for random generator

        iSecret = rand() % points.size();
        centers.push_back(points[iSecret]);
    }
}
else
{
    for (int i = 0; i < k; i++) {
```

```
        centers.push_back(points[i]);
    }
}
```

2. 寻找最近中心点。我用的暴力的枚举法，计算每一个数据点与当前的所有的点心点的距离，找最小的那个。

```cpp
T total_sum = 0; //

// initialize clusters storage
//    vector<vector<vector<T>>> clusters(N, vector<vector<T>>);

map<int, vector<vector<T>>> clusters;
// calculate square sum
for (typename vector<vector<T>>::iterator it = points.begin(); it!=points.end(); it++) {

    ////////////// BEGIN: find the nearest center, and the minimum squared distance//////////////
    T min = ((*it)[0]-centers[0][0]) * ((*it)[0]-centers[0][0]); // how to find the maximum of Type T, and initialize 0 to T?
    //
    for (int i=1; i<N; i++) {
        min += ((*it)[i]-centers[0][i]) * ((*it)[i]-centers[0][i]);
    }

    //        cout<<"Begin min:"<<min<<endl;
    int min_i = 0;// the nearest center's index

    for (int i = 1; i<k_centers; i++) { // k, centers
        // test

        T sum = ((*it)[0]-centers[i][0]) * ((*it)[0]-centers[i][0]);
        //            cout<<"sum: "<<sum<<"\t";
        for (int j = 1; j<N; j++) { // N, dimension
            // WRONG FORMAT:                sum += (*it[j]-centers[i][j]) * (*it[j]-centers[i][j]);
            sum += ((*it)[j]-centers[i][j]) * ((*it)[j]-centers[i][j]);
            //                cout<<sum<<"\t";
        }
        //            cout<<"compare sum and min||sum:"<<sum<<"||min:"<<min<<"||sum<min:"<<(sum<min)<<endl;
        if (sum < min) {
            //                waitKey(0);
            min = sum;
            min_i = i;
            //                cout<<"EXCANG: min:"<<min<<"\t min_i:"<<min_i<<"\t";
        }
        //            cout<<endl;
    }
    /////////////// END: find the nearest center, and the minimum squared distance///////////////

    /////////////// store points for cluster min_i ////////////////////////
    clusters[min_i].push_back(*it); // save point to cluster min_i
    total_sum += min;
    //        cout<<"min: "<<min<<"\ttotal_sum:"<<total_sum<<endl;
}
```

3. 更新中心点。

```cpp
////////////////////// update centers //////////////////////

for (int i = 0; i < k_centers; i++) {

    if (clusters.find(i)==clusters.end()) // if there is no point assigned to this center, do nothing about it. You can also change this
    {
        continue;
    }
    vector<T> center = clusters[i][0];


```

```
    int c_i_size = (int)clusters[i].size(); // might be wrong , when initial centers are same

    // if there is no point in cluster[i], continue
    //        if (c_i_size==0) {
    //            continue;
    //        }

    for (int j=1; j<c_i_size; j++) {
        for (int d = 0; d < N; d++) {
            center[d] += clusters[i][j][d]; // might be wrong , when initial centers are same.
        }
    }

    // update centers
    centers[i].clear();
    for (int d = 0; d < N; d++) {
        centers[i].push_back(center[d]/c_i_size);
    }
}

Total_Sum_Sqrt = sqrt(total_sum);
//    Total_Sum = total_sum;
```

4. 终止步骤 2 和 3 的迭代过程的条件可以自己选择，比如：

a. 设定迭代次数

b. 分配给每个类的点数不再变动

c. 中心点不再变动

d. 整体欧氏距离平方和不再变动

e. 整体欧氏距离平方和小于某个阈值

在这里我用了条件 d，同时我也在寻找过程中记录了整体欧氏距离平方和最小情况下的中心点信息。

实践经验是：K-means收敛挺快，还没遇到过无限迭代的情况；整体欧氏距离平方和有时候会先下降到一定程度，然后再稍稍上升一点，目前还不知道为什么；下面这两种方式先出来的中心点一般差别不大，在实验的图片上的效果看不出差别。

代码如下：

```cpp
switch (end_modal) {
     case 0:
     {
         do{
             pre_sum = total_sum;
             //        show_info();
             total_sum = calculate_center_once(points);
         }while (pre_sum-total_sum);

     }
         break;
     case 1:
     {
         T min_sum = total_sum;
         tem_centers = centers;
         do{
             pre_sum = total_sum;
             show_info();
             total_sum = calculate_center_once(points);

             if (min_sum > total_sum) {
                 min_sum = total_sum;
                 //                    tem_centers.clear();
                 tem_centers = centers;
             }
         }while (pre_sum-total_sum);

```

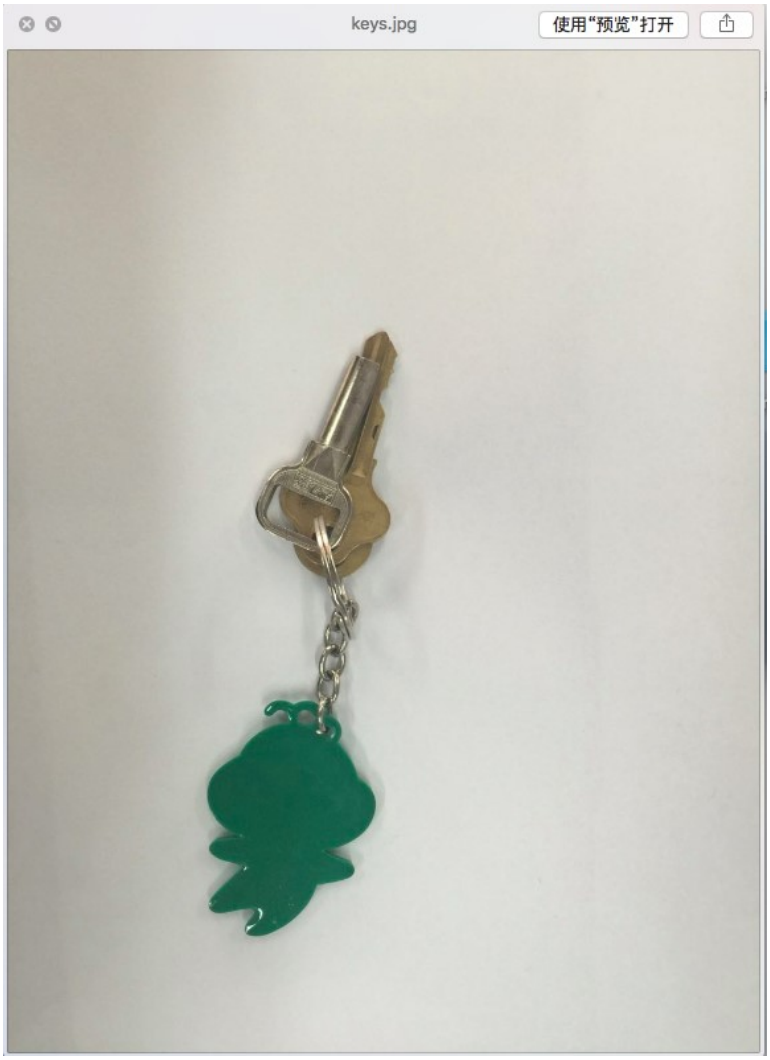
```
            //            centers.clear();
            centers = tem_centers;
            Total_Sum_Sqrt = min_sum;

        }
            break;
        default:
            break;
    }
```

参考：http://nlp.stanford.edu/IR-book/html/htmledition/k-means-1.html

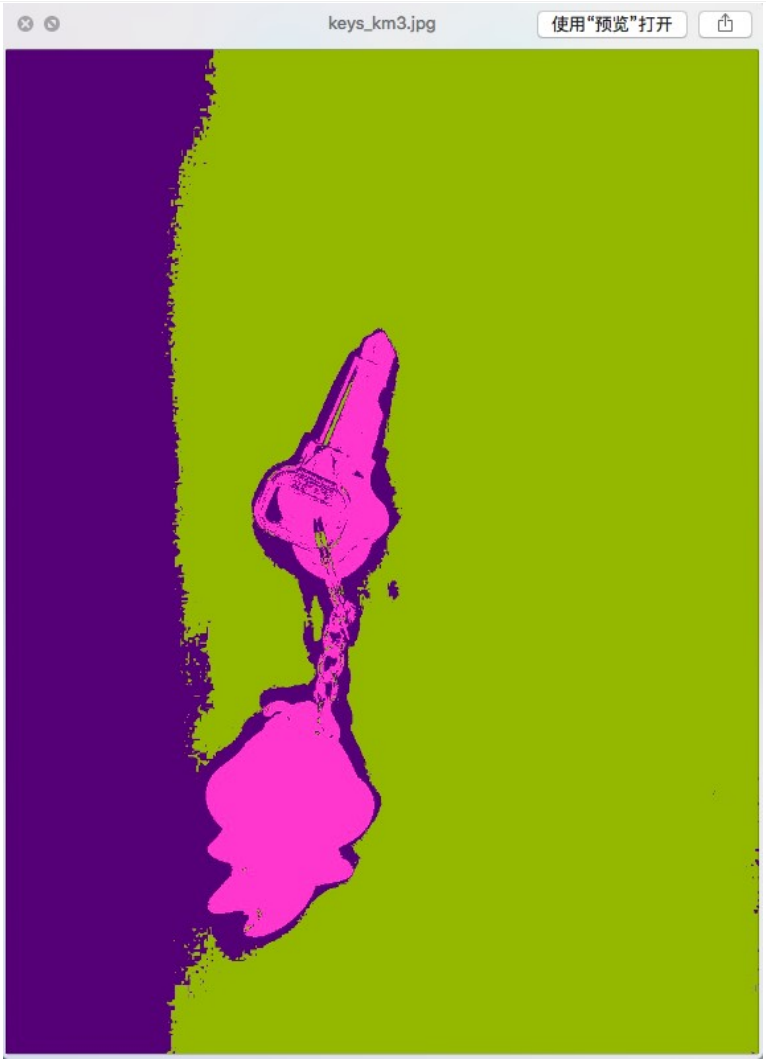
https://www.cs.umd.edu/~mount/Projects/KMeans/pami02.pdf

同时，我对一张图片进行了一些测试。

原图：



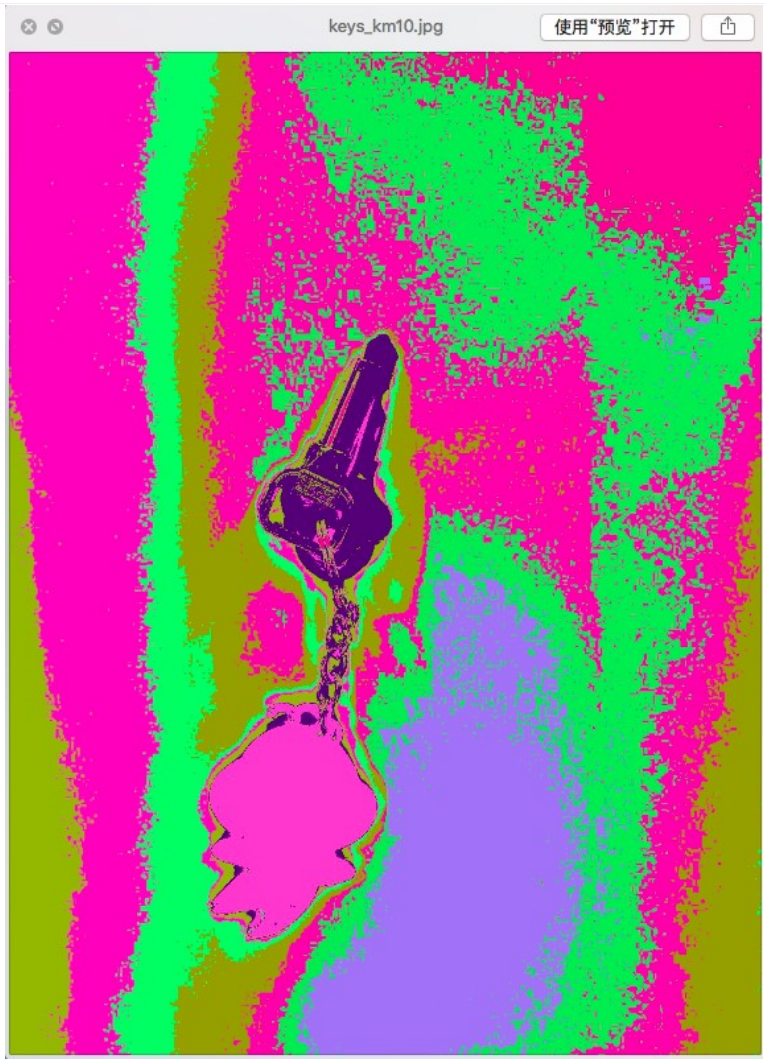
聚3类：



聚4类：

聚7类：

聚10类：



聚20类：

顶 1　　踩 0

上一篇　RGB、HSB\HSV、HSL三种颜色空间的原理理解与转换
下一篇　leetcode----Combinations

相关文章推荐

- 模式识别经典算法——Kmeans图像聚类分割（以…
- 携程机票大数据基础平台架构演进-- 许鹏
- 【机器学习实战之三】：C++实现K-均值（K-Mea…
- Python可以这样学--董付国
- 图像聚类-层次聚类
- 一步一步学Spring Boot
- 图像聚类-K均值聚类
- 深入浅出C++程序设计
- Shiny应用基础（1）：导言
- Android Material Design 新控件
- 图像聚类-谱聚类
- 机器学习需要用到的数学知识
- Python计算机视觉：第六章 图像聚类
- C++实现K-means，聚类原理解析（并用在图片像…
- k-means(k均值聚类)算法介绍及实现(c++)
- k-means(k均值聚类)算法介绍及实现(c++)

查看评论

暂无评论

发表评论

用 户 名： matrix6666

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场