**实验三 基于划分的聚类算法研究与实现**

**一、实验目的**

通过编程实现k均值和k中心点聚类算法。

**二、实验设备和环境**

win10,codeblocks,c++11

**三、实验内容**

编程实现k均值及k中心点聚类算法，并在数据集上完成测试，分别给出k=5、10、20时的聚类结果，在实验结果的基础上分析k均值及k中心点算法的优缺点。

**四、实验过程、结果及分析**

1.

实验过程

先观察数据格式，实现读入数据和预处理的部分；

然后实现k-means，寻找错误，调整参数，检验正确性；

最后复用k-means实现过程中的一些代码，在此基础上实现k-medios。

关键代码

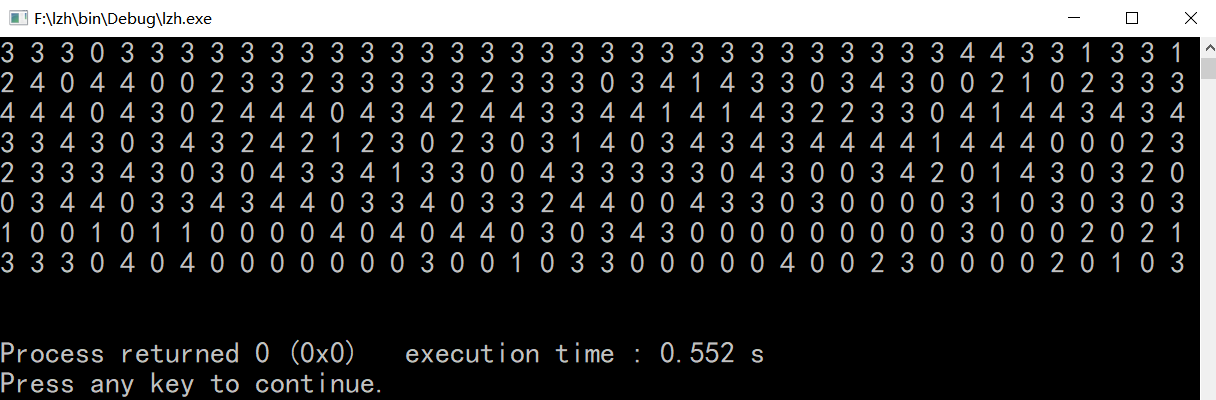
1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **void** readIn(**char**\***FILE**,vector<vector<**float**> >&dat){
4. freopen(**FILE**,"r",stdin);
5. **char** buf[1<<16];
6. **while**(gets(buf+1)){
7. **int** len=strlen(buf+1);
8. vector<**float**>tmp;
9. **for**(**int** i=1;i<=len;i++)
10. **if**(i==1||(buf[i-1]==' ')){
11. **float** x;
12. sscanf(buf+i,"%f",&x);
13. tmp.push\_back(x);
14. }
15. dat.push\_back(tmp);
16. }
17. }
19. **float** getDis(vector<**float**>&A,vector<**float**>&B){
20. **int** L=A.size();
21. **float** res=0;
22. **for**(**int** i=0;i<L;i++)
23. res+=(A[i]-B[i])\*(A[i]-B[i]);
24. **return** res;
25. }
26. vector<**int**> kmeans(vector<vector<**float**> >dat,**int** k){
27. **int** n=dat.size(),m=dat[0].size();
28. vector<**int**>res(n,-1);
29. vector<**int**>cnt(k,0);
30. vector<**int**>perm;
31. vector<vector<**float**> >center;
32. **for**(**int** i=0;i<n;i++)
33. perm.push\_back(i);
34. srand(time(NULL));
35. random\_shuffle(perm.begin(),perm.end());
36. **for**(**int** i=0;i<k;i++){
37. center.push\_back(dat[perm[i]]);
38. res[perm[i]]=i;
39. cnt[i]=1;
40. }
42. **for**(**int** i=k;i<n;i++){
43. **float** minDis=1e9;
44. **int** minId;
45. **for**(**int** j=0;j<k;j++){
46. **float** d=getDis(dat[perm[i]],center[j]);
47. **if**(d<minDis)
48. minDis=d,minId=j;
49. }
50. res[perm[i]]=minId;
51. **for**(**int** j=0;j<m;j++){
52. center[minId][j]=(center[minId][j]\*cnt[minId]+dat[perm[i]][j])/(cnt[minId]+1);
53. cnt[minId]++;
54. }
55. }
56. **return** res;
57. }
59. **float** classify(vector<vector<**float**> >&dat,vector<vector<**int**> >&cluster,vector<**int**>¢er,**int** k,**int** id){
60. **float** minDis=1e9;
61. **int** minId;
62. **for**(**int** j=0;j<k;j++){
63. **float** d=getDis(dat[id],dat[center[j]]);
64. **if**(d<minDis)
65. minDis=d,minId=j;
66. }
67. cluster[minId].push\_back(id);
68. **return** minDis;
69. }
71. vector<**int**> kmedios(vector<vector<**float**> >dat,**int** k,**int** iters){
72. **int** n=dat.size(),m=dat[0].size();
73. vector<**int**>res(n,-1);
74. vector<**int**>perm;
75. vector<vector<**int**> >cluster;
76. vector<**int**>center;
77. **for**(**int** i=0;i<n;i++)
78. perm.push\_back(i);
79. srand(time(NULL));
80. random\_shuffle(perm.begin(),perm.end());
81. **for**(**int** i=0;i<k;i++){
82. vector<**int**>tmp;tmp.push\_back(perm[i]);
83. cluster.push\_back(tmp);
84. center.push\_back(perm[i]);
85. }
86. **float** curLoss=0;
87. **for**(**int** i=k;i<n;i++)
88. curLoss+=classify(dat,cluster,center,k,perm[i]);
89. **float** minLoss=1e9;
90. **for**(**int** I=0;I<iters;I++){
91. **bool** flag=0;
92. **for**(**int** i=0;i<k;i++){
93. **for**(**int** j=k;j<n;j++){
94. swap(perm[i],perm[j]);
95. vector<vector<**int**> >newCluster;
96. vector<**int**>newCenter;
97. **for**(**int** p=0;p<k;p++){
98. vector<**int**>tmp;tmp.push\_back(perm[p]);
99. newCluster.push\_back(tmp);
100. newCenter.push\_back(perm[p]);
101. }
102. **float** newLoss=0;
103. **for**(**int** p=k;p<n;p++)
104. newLoss+=classify(dat,cluster,center,k,perm[p]);
105. **if**(newLoss<minLoss){
106. minLoss=newLoss;
107. cluster=newCluster;
108. center=newCenter;
109. flag=1;
110. }
111. }
112. }
113. **if**(!flag)
114. **break**;
115. printf("iter:%d, loss:%.2f\n",I,minLoss);
116. }
117. **for**(**int** i=0;i<k;i++)
118. **for**(**int** j=0;j<cluster[i].size();j++)
119. res[cluster[i][j]]=i;
120. **return** res;
121. }
123. **void** print(**char**\***FILE**,**int** k,vector<**int**>res){
124. freopen(**FILE**,"w",stdout);
125. vector<**int**>s[k];
126. **for**(**int** i=0;i<res.size();i++)
127. s[res[i]].push\_back(i);
128. **for**(**int** i=0;i<k;i++){
129. printf("class %d, total number:%d:",i,s[i].size());
130. **for**(**int** j=0;j<s[i].size();j++)
131. printf(" %d",s[i][j]);
132. printf("\n");
133. }
134. }
135. **int** main(){
136. vector<vector<**float**> >dat;
137. readIn("data.txt",dat);
138. vector<**int**>res=kmedios(dat,5,20);
139. **for**(**int** i=0;i<res.size();i++)
140. printf("%d ",res[i]);
141. printf("\n");
142. print("5.txt",5,res);
143. **return** 0;
144. }

实验结果

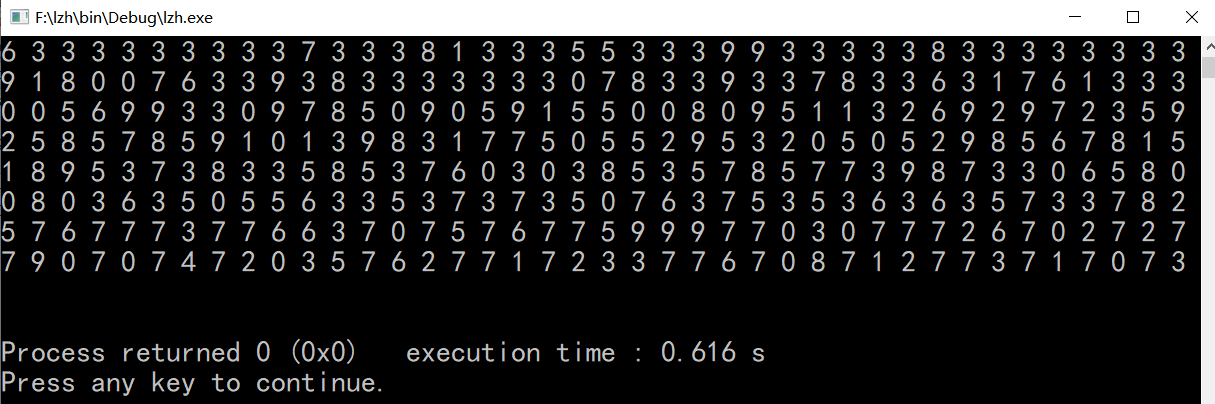
k-means:随机算法，随机数种子不同时产生的结果不同，此处为保证随机性，使用当前时间作为种子，产生结果如下。

第i个数字表示第i个元素属于哪个集合。

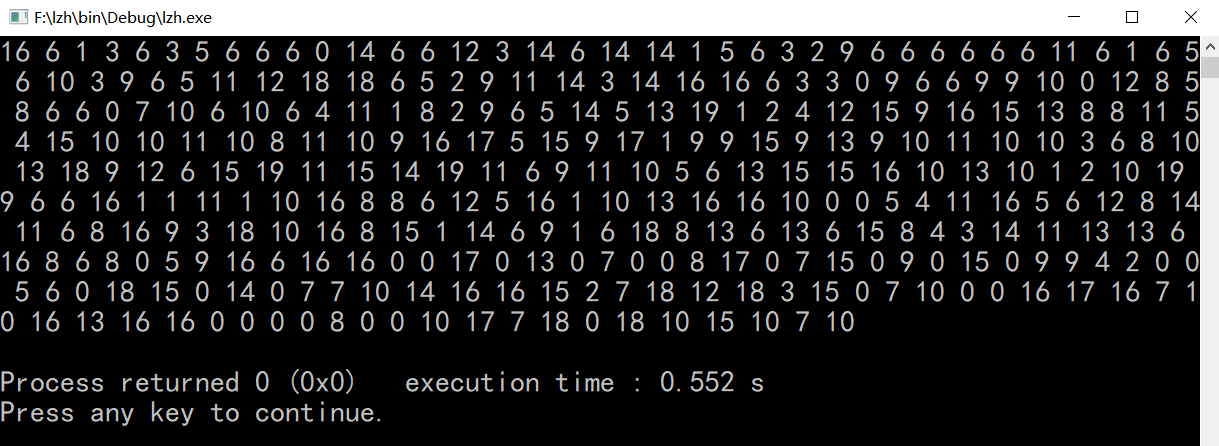
k=5



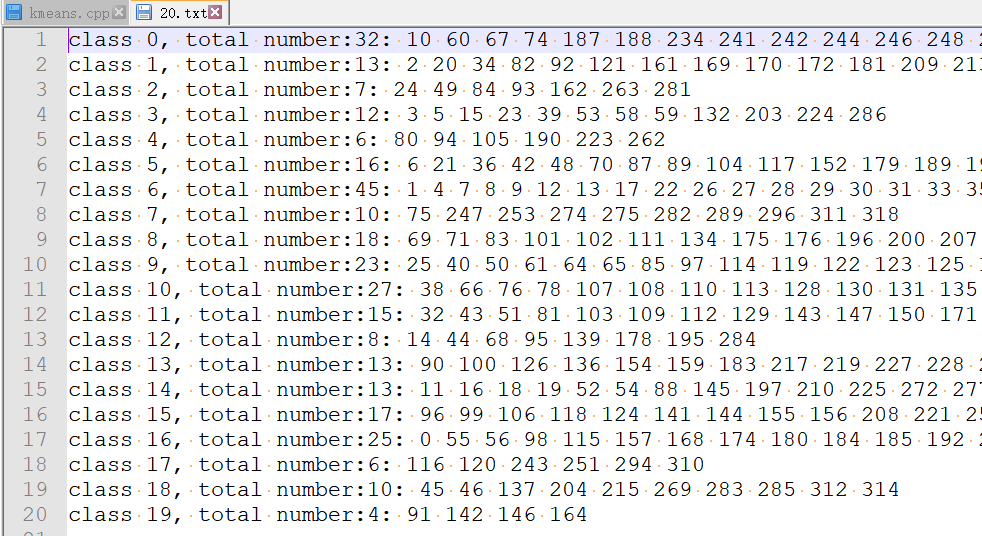
k=10



k=20

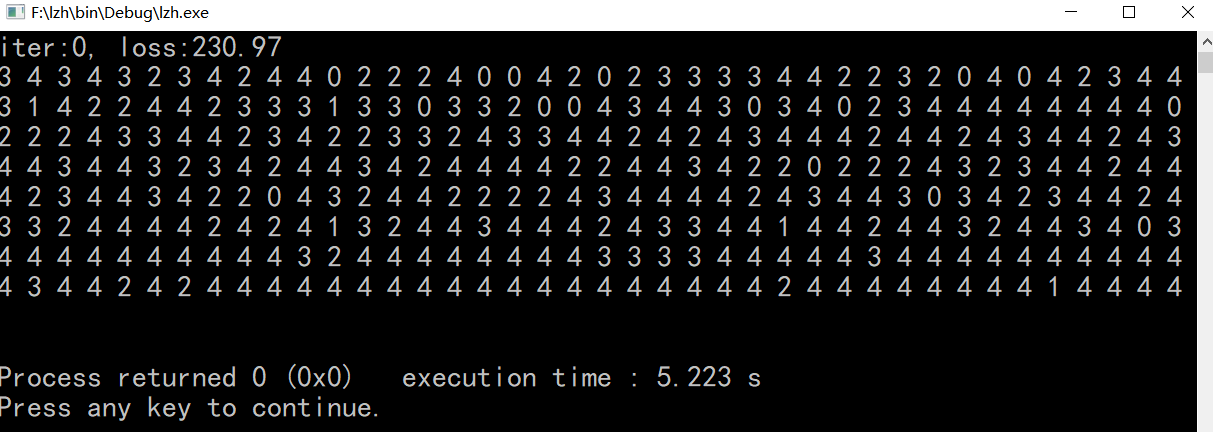


结果也以另一种格式存入了文件中。例如：

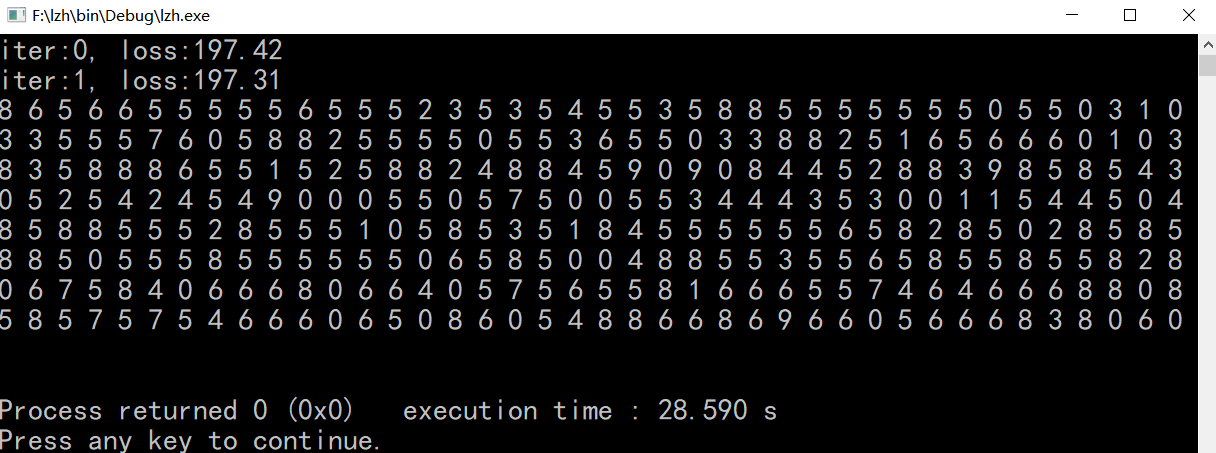


k-medios:设置最大迭代次数为20，k较小的时候并不需要这么多次，运行速度明显慢于k-means。

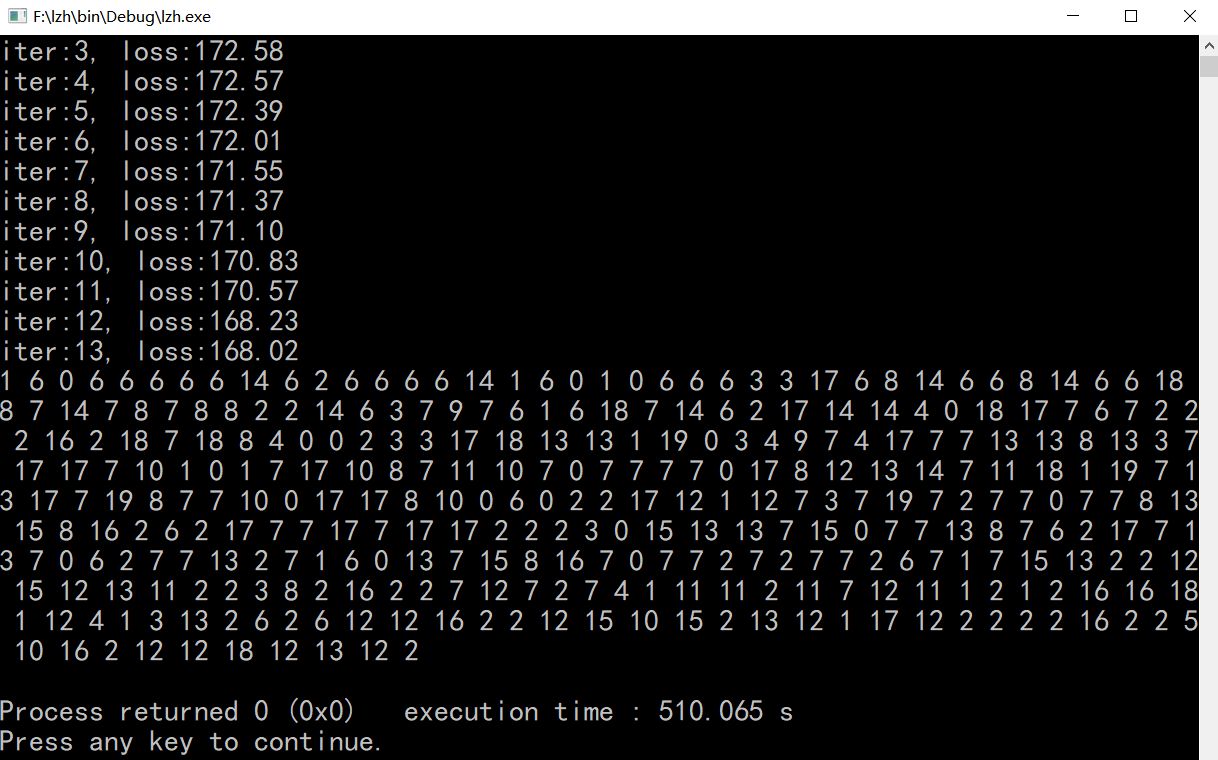
k=5



k=10

****

k=20

****

**2.遇到问题和解决方法**

问题：

实现k-means时，没有考虑后续实现k-medios时进行代码复用，导致k-means存储中心点的数据结构不能与k-medios相同，大部分代码无法复用。

解决方法：

最终为了避免之前的代码出错，没有统一写法，而是将之前的代码进行复制粘贴用于实现k-medios，导致代码相对冗长。

**3.存在问题**

(1)k-medios运行速度慢。

(2)很多模块化代码没有预先设计好，导致有些代码其实作用相同（例如将每个点加入最近的簇）但也不方便统一起来。

**4.心得体会**

k-means的结果受离群点影响很大，不同的随机数种子选定的初始点会导致结果差距非常大，有时也会算出很差的结果。

k-medios由于需要反复迭代，枚举每个点作为新的中心点，时间复杂度非常高。k=20，耗时已经非常久了。

**实验成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**