

作业7——KMeans

代码仓库<https://github.com/Bruce-Boss/hw7>

✓ 运行环境

- win10+IDEA+hadoop单机完成编码调试
- bdkit完成集群中任务提交和运行。

✓ 源码解读与修改

开始时，因为有了先前两次的mapreduce编码经验，也完全掌握了K-Means算法，我考虑从零完成这一任务。后来经过测试，我发现本次作业可以直接使用黄宜华老师教材提供的源码。因此，我决定将本次编码任务转变为代码解读任务，尝试分析该项目源码，并进行细微修改。

我刚打开源码时有些意外，因为不是原来那种一个.java文件就能完成作业的情况。该项目有11个.java文件，还有名为king的包，让我吃了一惊。本以为可能只有名为KMeans的文件是真正有用的（其他都是扩充功能用到的类），只用五分钟就能看个大概。没想到这个类里面甚至连main函数都没有，还导入了奇怪的名为King的包。（后来发现，King其实是项目的作者...）

在放弃了短时间内读懂一个源文件就能解决任务的幻想后，我开始仔细分析这个项目所用到的各个类以及他们之间的关系和在任务中所起的作用。

首先，我在KMeansDriver中找到了main函数。找到main函数，就找到了程序运行的入口。之后我抽丝剥茧，逐渐理清了项目的逻辑（具体说明见源码中的注释）：

类（.java文件）名	描述
Instance	代表坐标的数据类型
Cluster	代表簇的数据类型（核心数据成员是簇id，中心坐标，包含点数）
RandomClusterGenerator	用于生成初始聚类中心
Kmeans	用于完成算法迭代过程中的簇更新（不包括初始化）
King.Utills.*	定义了距离接口，完成了几个实现，用于在算法中计算各种自定义的距离。（本例中使用到的文件只有接口Distance和欧氏距离类EuclideanDistance。

类 (.java文件) 名	描述
KMeansCluster	完成最后一次迭代的结果输出
KMeansDriver	整个流程的控制单元，负责读入原始数据。之后调用不同的类，完成初始聚类中心的产生，簇的迭代和最终结果的生成。

之后我开始尝试对代码进行一些有价值的修改。我感觉类定义的有些多了，封装过度了，但是在尝试修改后发现当前的封装模式确实很清晰，也便于维护。后来我又尝试将KMeans类和KMeansCluster合并，因为二者都是迭代的步骤，只是输出的结果不同。但是后来发现，二者在功能上和逻辑上确实有不小区别，理应作为两个类。

最后我只对源码进行了小幅修改，删除了King.Utils中的一些无用文件，修改了最终结果的输出格式（便于可视化过程中的解析）。

✓ 运行说明

输入文件

格式如下:

1,2,3,4,5

3,4,6,5,1

每行一个实例。

运行

输入参数:

k: 簇中心数

iteration num: 迭代数

input path: 输入路径

output path: 输出路径

打包成jar后，运行：

```
hadoop jar target/K-Means-1.0.jar <k> <iteration num> <input path>
<output path>
```

✓ 运行截图

源码打包

```
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ K-Means ---
[INFO] Building jar: /workspace/hw7/target/K-Means-1.0.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.630 s
[INFO] Finished at: 2020-11-15T13:49:19Z
[INFO] -----
```

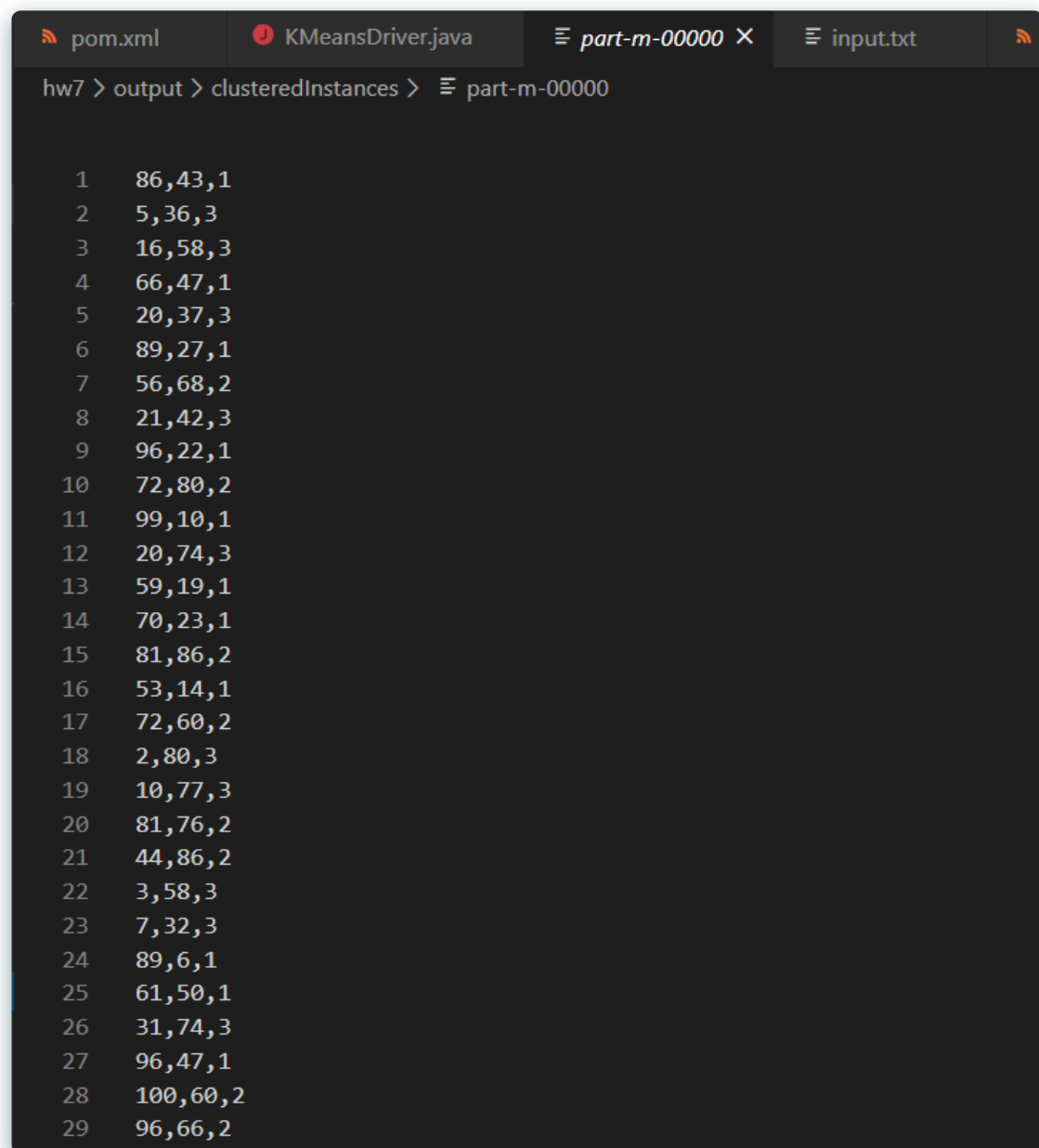
Resource-Manager中显示先完成了5次迭代，最后一次完成最终结果

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1605446380684_0006	root	KMeansClusterJob	MAPREDUCE	default	Sun Nov 15 21:52:35 +0800 2020	Sun Nov 15 21:52:48 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1605446380684_0005	root	clusterCenterJob4	MAPREDUCE	default	Sun Nov 15 21:52:18 +0800 2020	Sun Nov 15 21:52:34 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1605446380684_0004	root	clusterCenterJob3	MAPREDUCE	default	Sun Nov 15 21:52:00 +0800 2020	Sun Nov 15 21:52:16 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1605446380684_0003	root	clusterCenterJob2	MAPREDUCE	default	Sun Nov 15 21:51:43 +0800 2020	Sun Nov 15 21:51:59 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1605446380684_0002	root	clusterCenterJob1	MAPREDUCE	default	Sun Nov 15 21:51:24 +0800 2020	Sun Nov 15 21:51:41 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1605446380684_0001	root	clusterCenterJob0	MAPREDUCE	default	Sun Nov 15 21:51:11 +0800 2020	Sun Nov 15 21:51:23 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A

输出结果（我修改了输出格式，使得key和value之间以逗号分隔，便于之后可视化时读入后的字符串解析）

```
root@cyj181870013-master:/workspace# hadoop fs -cat /output/clusteredInstances
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so which might have disabled stack guard. The VM
try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
20/11/15 14:43:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
cat: '/output/clusteredInstances': Is a directory
root@cyj181870013-master:/workspace# hadoop fs -cat /output/clusteredInstances/part-m-00000
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so which might have disabled stack guard. The VM
try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
20/11/15 14:43:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
86,43,1
5,36,3
16,58,3
66,47,1
20,37,3
89,27,1
56,68,2
21,42,3
96,22,1
72,80,2
99,10,1
20,74,3
59,19,1
70,23,1
81,86,2
53,14,1
72,60,2
2,80,3
10,77,3
81,76,2
44,86,2
3,58,3
7,32,3
89,6,1
61,50,1
31,74,3
96,47,1
100,60,2
96,66,2
0,82,3
```

下载到本地的输出文件



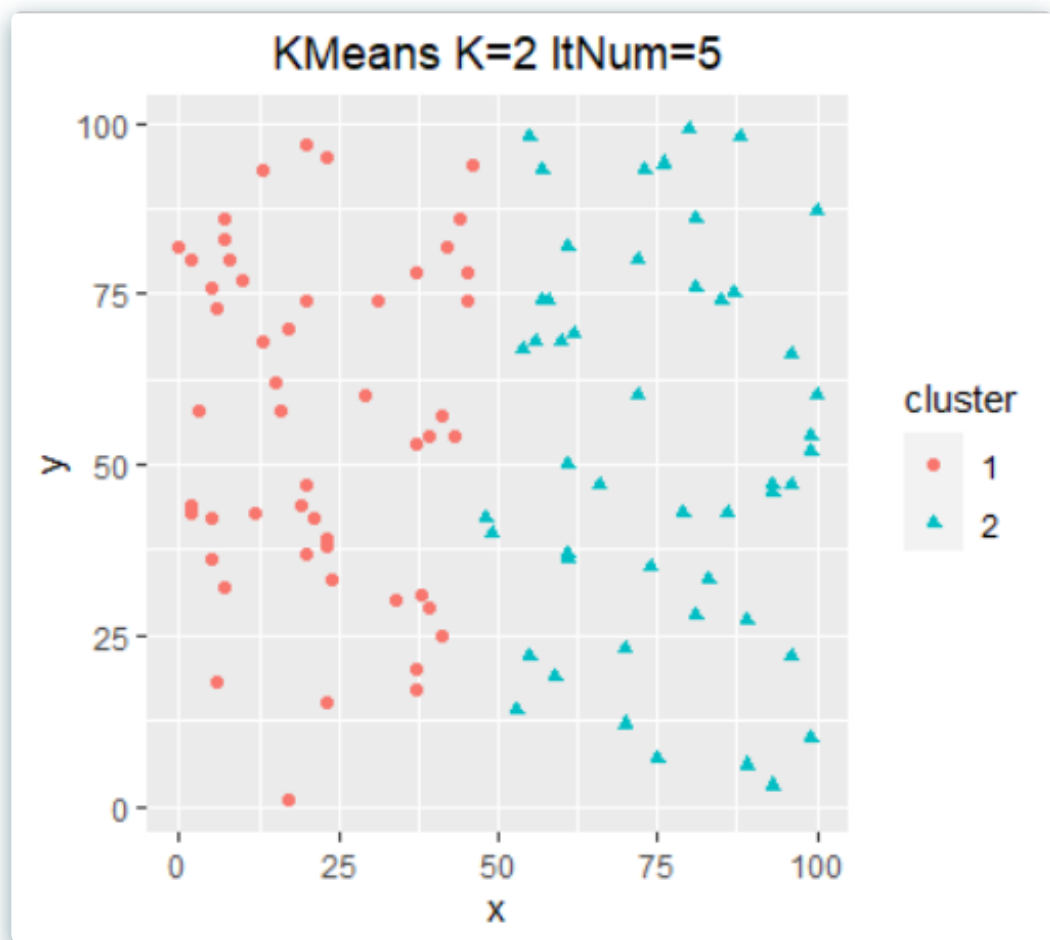
```
hw7 > output > clusteredInstances > part-m-00000

1 86,43,1
2 5,36,3
3 16,58,3
4 66,47,1
5 20,37,3
6 89,27,1
7 56,68,2
8 21,42,3
9 96,22,1
10 72,80,2
11 99,10,1
12 20,74,3
13 59,19,1
14 70,23,1
15 81,86,2
16 53,14,1
17 72,60,2
18 2,80,3
19 10,77,3
20 81,76,2
21 44,86,2
22 3,58,3
23 7,32,3
24 89,6,1
25 61,50,1
26 31,74,3
27 96,47,1
28 100,60,2
29 96,66,2
```

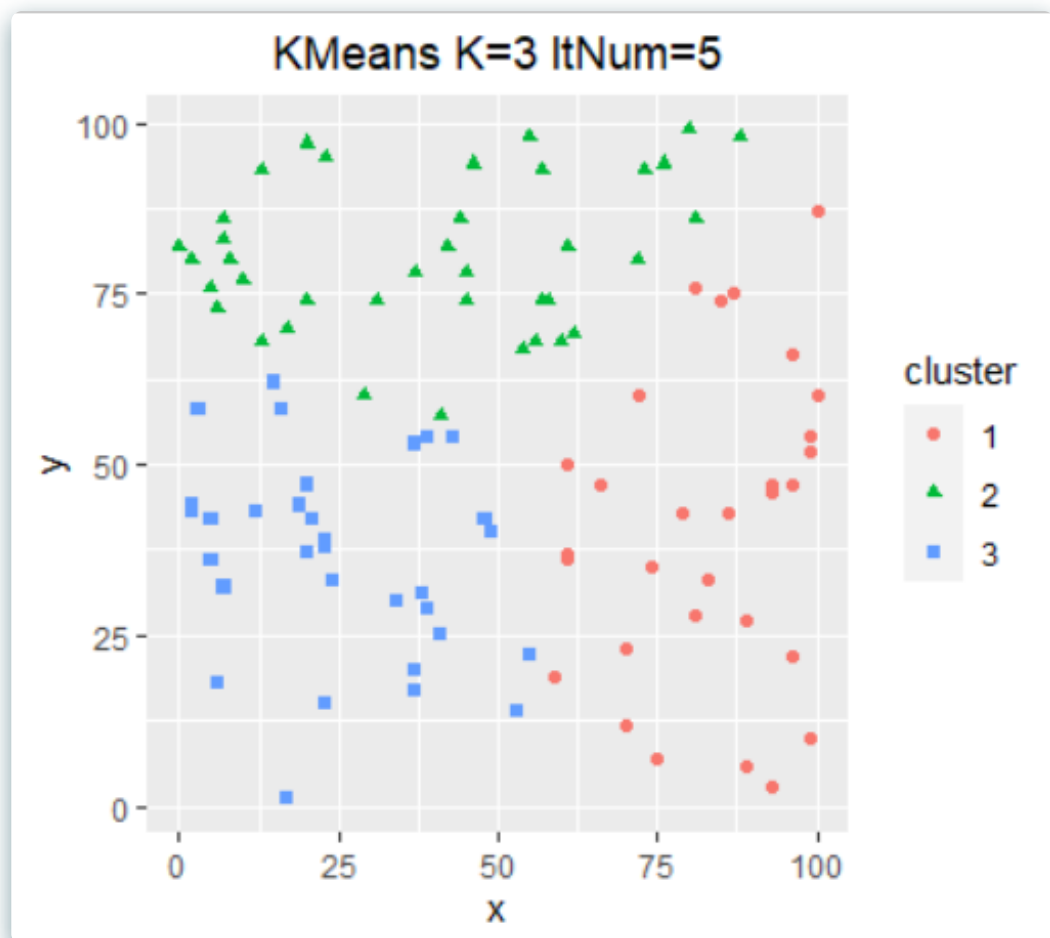
✓ 结果可视化

使用R语言的ggplot2包进行绘图。设置相同迭代次数和不同聚类数，观察聚类结果

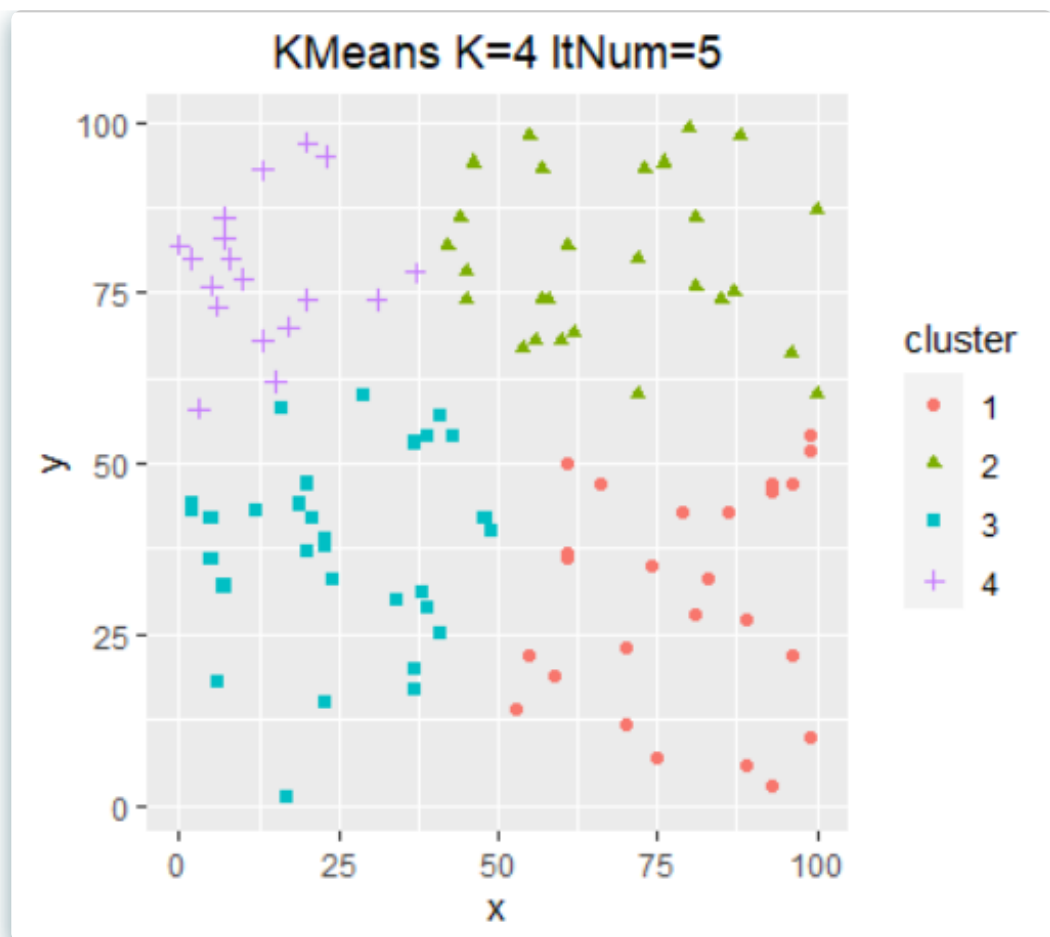
- K=2



- k=3



- K=4



✓ 使用手肘法找出最优的K:

手肘法

- 核心指标: SSE(sum of the squared errors, 误差平方和)

$$SSE = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2$$

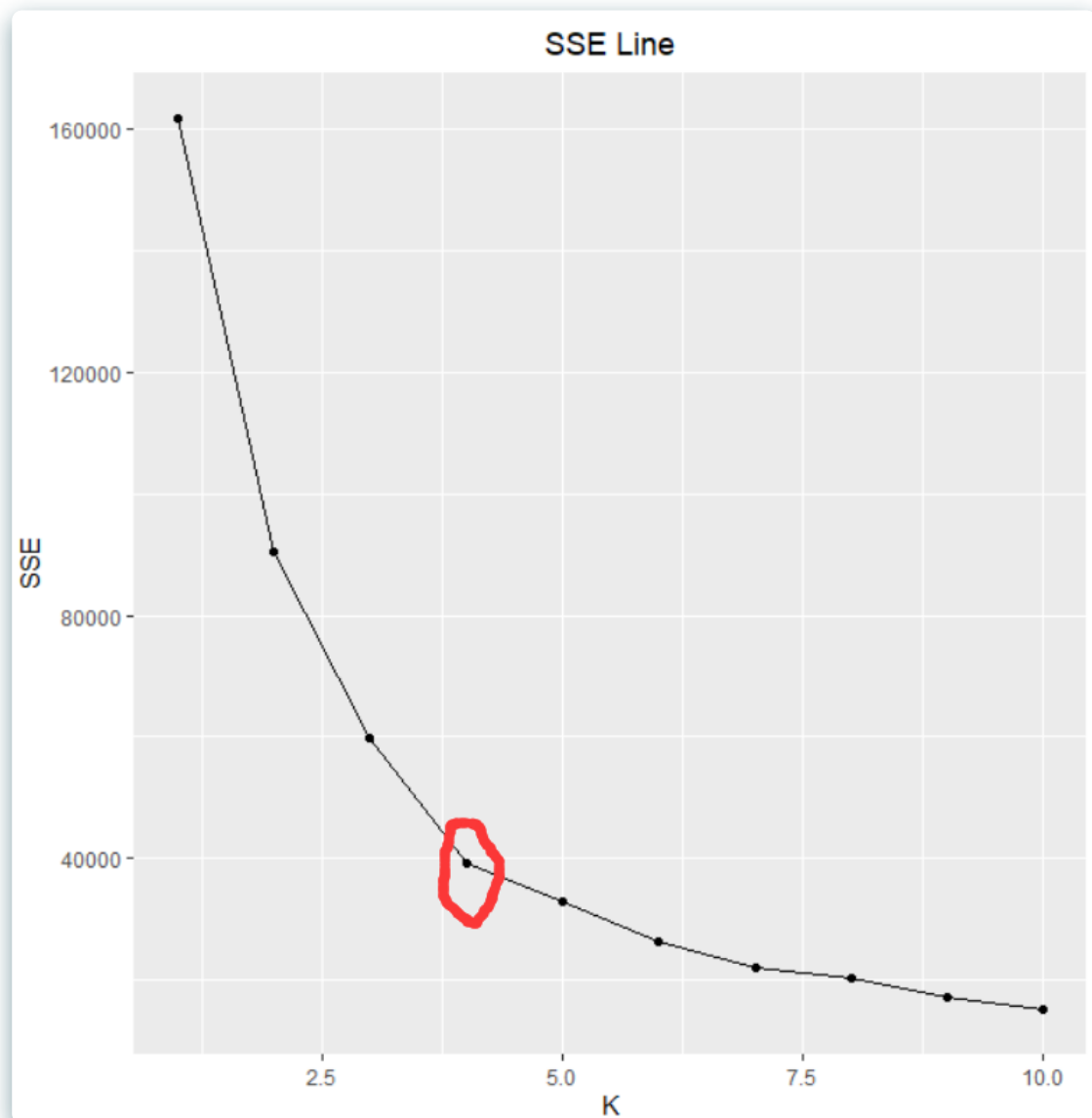
http://blog.csdn.net/qq_15738501

- C_i 是第*i*个簇
- p 是 C_i 中的样本点
- m_i 是 C_i 的质心 (C_i 中所有样本的均值)
- SSE是所有样本的聚类误差, 代表了聚类效果的好坏。

-手肘法核心思想

- 随着聚类数 k 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和SSE自然会逐渐变小。
- 当 k 小于真实聚类数时，由于 k 的增大会大幅增加每个簇的聚合程度，故SSE的下降幅度会很大，而当 k 到达真实聚类数时，再增加 k 所得到的聚合程度回报会迅速变小，所以SSE的下降幅度会骤减，然后随着 k 值的继续增大而趋于平缓，也就是说SSE和 k 的关系图是一个手肘的形状，而这个肘部对应的 k 值就是数据的真实聚类数

-手肘法可视化



可以认为手肘的拐点在4，因此最佳的K为4。（其实也看不出来是4，因为数据是随机生成的，本身没有很强的聚类性）

✓ 有趣的发现

之前mapreduce任务输出的文件名为part-r-00000，而本次作业中，中间迭代任务的输出文件为part-r-00000，最终结果为part-m-00000。

原因：当Reduce函数中有落盘操作，且指定CombinerClass为Reduce函数，则输出结果文件为多个包含“-m-”的文件，如果不指定CombinerClass，则生成文件为包含“-r-”的单个文件。当然，输出的文件名也是可以自定义的。