实验5 KMeans的mapreduce

> owen

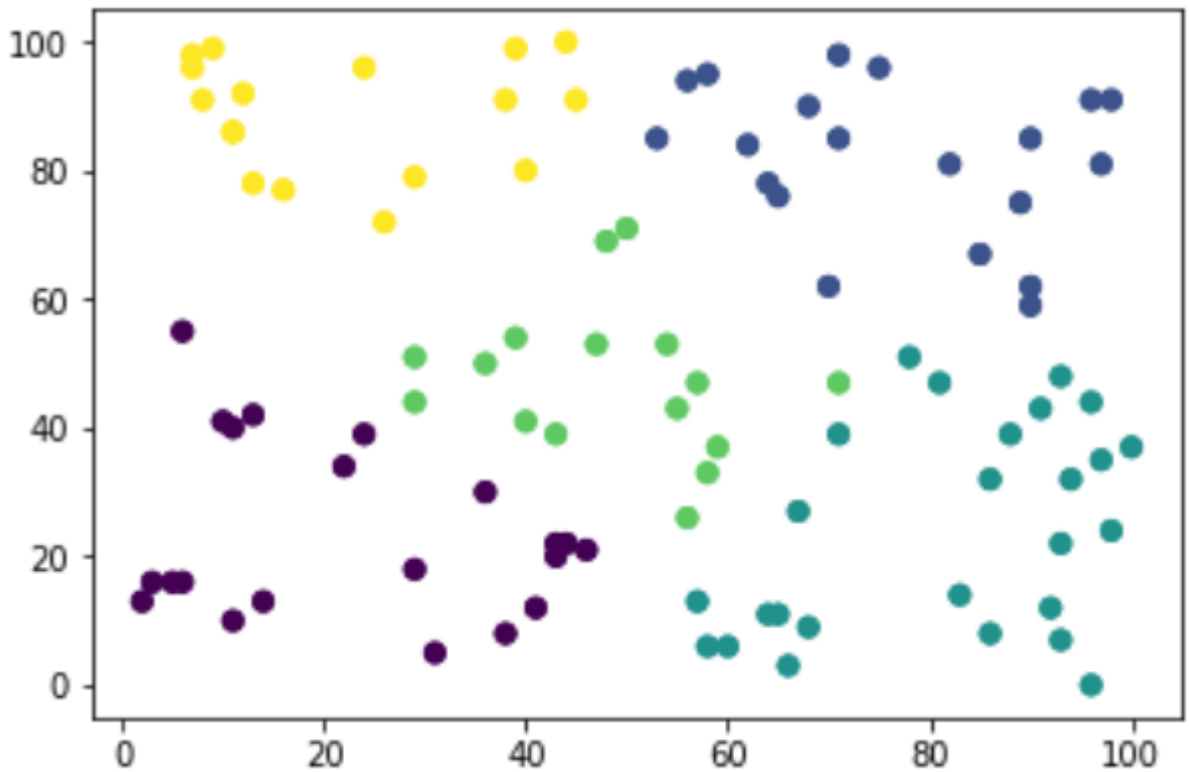
代码使用书上示例代码，全部代码见github仓库

> https://github.com/VoidFly/hadoop_practice

实验结果

```
root@f6815da6c635:/home/Hadoop/hadoop-3.2.1# bin/hadoop jar ../share-files/matrixMultiply.ja
r KMeansDriver 5 10 kInput/Instance2 kOutput
```

```
                Total megabyte-milliseconds taken by all map tasks=4045824
        Map-Reduce Framework
                Map input records=100
                Map output records=100
                Input split bytes=106
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=102
                CPU time spent (ms)=400
                Physical memory (bytes) snapshot=106463232
                Virtual memory (bytes) snapshot=2587877376
                Total committed heap usage (bytes)=32571392
                Peak Map Physical memory (bytes)=106463232
                Peak Map Virtual memory (bytes)=2587877376
        File Input Format Counters
                Bytes Read=584
        File Output Format Counters
                Bytes Written=784
finished!
```

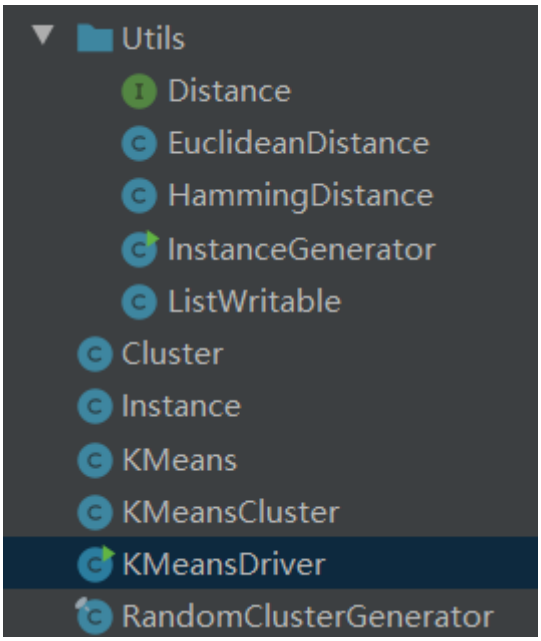


可视化代码

```
import matplotlib.pyplot as plt
import numpy as np
with open('dataBag/part-m-00000.txt','r') as f:
    data=f.readlines()
for i in range(len(data)):
    data[i]=data[i].strip('\n')
for i in range(len(data)):
    a=data[i].split('\t')[0]
    a1=float(a.split(',')[0])
    a2=float(a.split(',')[1])
    points_x.append(a1)
    points_y.append(a2)
    b=int(data[i].split('\t')[1])
    label.append(b)
plt.scatter(points_x,points_y,c=label)
```

示例代码理解

项目文件如下



main函数中，调用randomClusterGenerator读取instance信息，按照一定概率从instances点中选取centers，结束后将产生的centers信息保存在cluster-0文件下，并且在下一步迭代计算时装入mapreduce的distributed cache中，作为全局共享数据。

```
public static void main(String[] args) throws IOException, InterruptedException, Class
    System.out.println("start");
    Configuration conf = new Configuration();
    int k = Integer.parseInt(args[0]);
    int iterationNum = Integer.parseInt(args[1]);
    String sourcePath = args[2];
    String outputPath = args[3];
    KMeansDriver driver = new KMeansDriver(k, iterationNum, sourcePath, outputPath, co
    driver.generateInitialCluster();
    System.out.println("initial cluster finished");
    driver.clusterCenterJob();

    driver.KMeansClusterJod();
```

```
super.setup(context);
FileSystem fs = FileSystem.get(context.getConfiguration());
FileStatus[] fileList = fs.listStatus(new Path(context.getConfiguration().get("clusterPath")));
BufferedReader in = null;
```

Mapreduce过程为，在每个map节点，读取全局聚类中心信息，计算节点所分配的节点到这些中心的距离，并将节点归到最近距离的中心中（发射key，val对），

```
public void map(LongWritable key, Text value, Context context)throws
IOException, InterruptedException{
    Instance instance = new Instance(value.toString());
    int id;
    try {
        id = getNearest(instance);
        if(id == -1)
            throw new InterruptedException("id == -1");
        else{
            Cluster cluster = new Cluster(id, instance);
            cluster.setNumOfPoints(1);
            System.out.println("cluster that i emit is:" + cluster.toString());
            context.write(new IntWritable(id), cluster);
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
```

reduce节点接到这些键值对以后，按照相同key（中心id），根据对应的val（点坐标信息）重新计算中心坐标（注意，在reduce之前， 为了减轻传输压力，使用了combiner，先根据一个map节点发射的坐标计算一次中心（用均值计算），再发送给reducer，其实本质上combiner做的和reducer做的是同一件事。

```
public static class KMeansCombiner extends Reducer<IntWritable,Cluster,IntWritable,Cluster
    public void reduce(IntWritable key, Iterable<Cluster> value, Context context)throws
    IOException, InterruptedException{
        Instance instance = new Instance();
        int numOfPoints = 0;
        for(Cluster cluster : value){
            numOfPoints += cluster.getNumOfPoints();
            System.out.println("cluster is:" + cluster.toString());
            instance = instance.add(cluster.getCenter().multiply(cluster.getNumOfPoints())
        }
        Cluster cluster = new Cluster(key.get(),instance.divide(numOfPoints));
        cluster.setNumOfPoints(numOfPoints);
        System.out.println("combiner emit cluster:" + cluster.toString());
        context.write(key, cluster);
    }
```

```
public static class KMeansReducer extends Reducer<IntWritable,Cluster,NullWritable,Cluster
    public void reduce(IntWritable key, Iterable<Cluster> value, Context context)throws
    IOException, InterruptedException{
        Instance instance = new Instance();
        int numOfPoints = 0;
        for(Cluster cluster : value){
            numOfPoints += cluster.getNumOfPoints();
            instance = instance.add(cluster.getCenter().multiply(cluster.getNumOfPoints()
        }
        Cluster cluster = new Cluster(key.get(),instance.divide(numOfPoints));
        cluster.setNumOfPoints(numOfPoints);
        context.write(NullWritable.get(), cluster);
    }
}
```

在main中，clusterCenterJob用于迭代划分聚类，在迭代收敛以后，KMeansClusterjod用于读取instance，划分聚类，并进行最后的输出。

```
driver.clusterCenterJob();

driver.KMeansClusterJod();
```