# 聚类算法开发指南

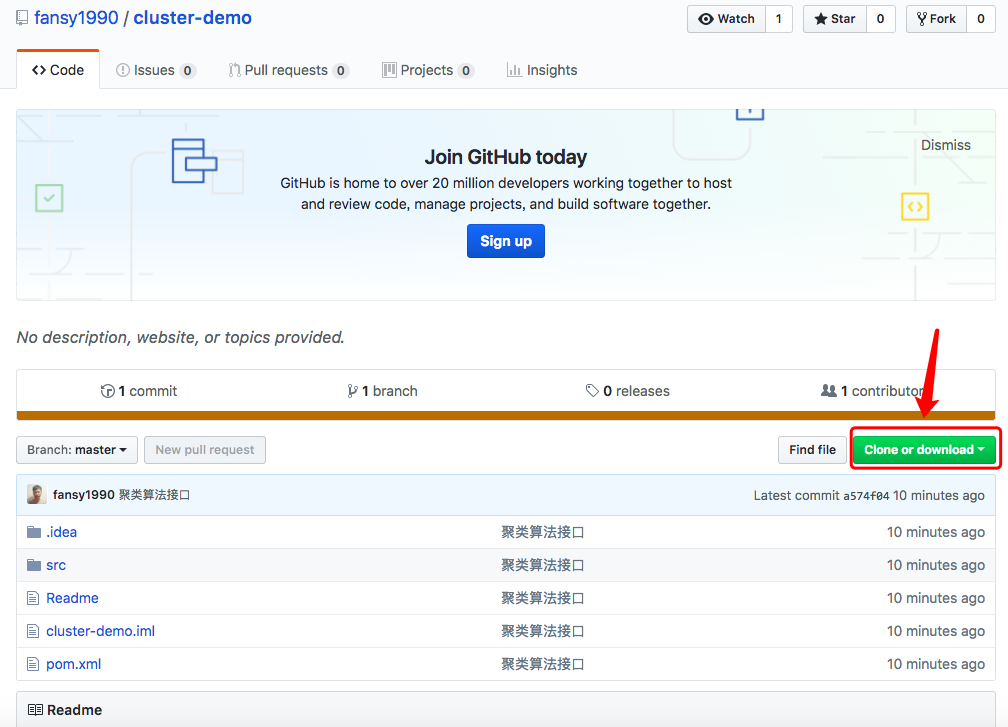
## 聚类算法开发基础知识

1. Java开发
2. Maven开发
3. Github使用
4. 不要改动非自己包下面的文件及代码（pom.xml文件除外）

## 具体开发

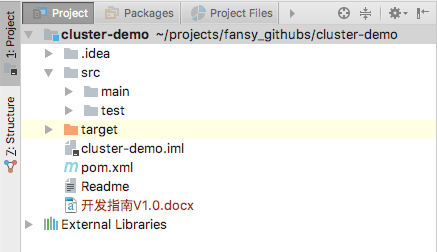
### 下载工程

从路径：<https://github.com/fansy1990/cluster-demo> 下载工程：



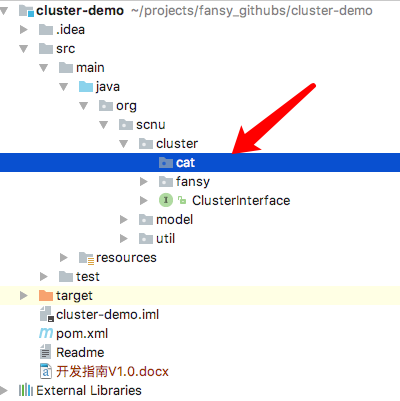
并导入到自己到IDE环境（如Eclipse或IDEA）。

导入后如下：



### 开发算法描述

现在比如我要开发一个KMeans算法，我的名字是cat，那么我需要先在org.scnu.cluster下面新建一个包cat，如下：



接着，在该包下面新建一个类CatCluster，如下所示（需要实现ClusterInterface接口）

|  |
| --- |
| **public class CatCluster implements** ClusterInterface {  *@Override* **public** List<**Instance**> runCluster(List<**Instance**> inputs, Map params) **throws Exception** {  **return null**;  } } |



### 编写算法

设置算法参数为三个：

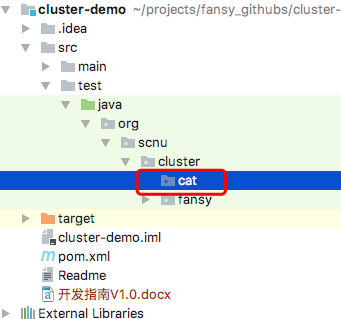
|  |
| --- |
| *k: k个聚类中心；  delta ： 前后两次聚类中心误差阈值；  iteration： 最大循环次数；* |

代码如下：

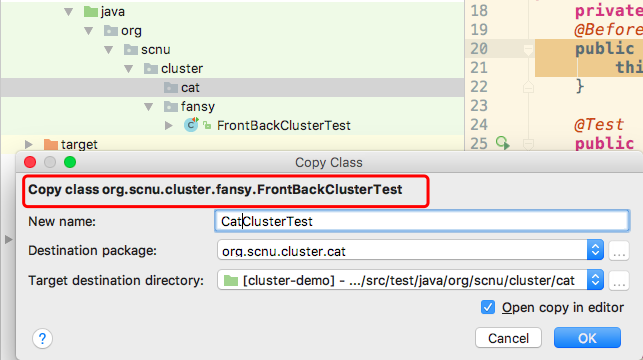
|  |
| --- |
| **package org.scnu.cluster.cat**;  **import org.scnu.cluster.**ClusterInterface; **import org.scnu.model.Instance**;  **import java.util.**\*;  */\*\*  \* KMeans算法Java实现  \* 算法参数：  \* k: k个聚类中心；  \* delta ： 前后两次聚类中心误差阈值；  \* iteration： 最大循环次数；  \*  \* @author fanzhe  \* @email fansy1990@foxmail.com  \* @date 2018/3/8 下午3:21.  \*/* **public class CatCluster implements** ClusterInterface {    **int** k ;  **double** delta;  **int** iteration;  **static int** *seed* =11;    *@Override* **public** List<**Instance**> runCluster(List<**Instance**> inputs, Map params) **throws Exception** {  *// 1. 初始化参数* init(params);    *// 2. 从所有数据中获取初次聚类中心* List<**Instance**> preCenter = *getRandomCenter*(inputs,k);   *// 3. 初始化下次聚类中心* List<**Instance**> nextCenter = **new** ArrayList<>();   *// 4. 更新循环聚类中心，并验证是否满足阈值* **for**(**int** i=0;i<iteration;i++){  nextCenter = *getCenter*(inputs,preCenter);  **if**(!*shouldRunNext*(preCenter,nextCenter,delta)){  *// 循环满足阈值，退出* **System**.***out***.println("iteration:"+i);  *// 5. 返回分类后的结果* **return** classify(nextCenter,inputs);  }  preCenter= nextCenter;  }   **return null**;  }   */\*\*  \* 执行分类  \* @param nextCenter  \* @param inputs  \* @return  \*/* **private** List<**Instance**> classify(List<**Instance**> nextCenter, List<**Instance**> inputs) **throws CloneNotSupportedException** {  List<**Instance**> result = **new** ArrayList<>();  **for**(**Instance** instance : inputs){  result.add(instance.addLabel(*getLineIndex*(instance,nextCenter)));  }  **return** result;  }   */\*\*  \* 初始化数据  \* @param params  \*/* **private void** init(Map params) {  **try**{  k = (**Integer**) params.get("k");  delta = (**Double** ) params.get("delta");  iteration =(**Integer**) params.get("iteration");  }**catch** (**Exception** e){    }    }    */\*\*  \* 是否应该继续下次循环  \* @param preCenter  \* @param nextCenter  \* @param delta  \* @return  \*/* **private static boolean** shouldRunNext(List<**Instance**> preCenter,  List<**Instance**> nextCenter,**double** delta) {  *printCenter*("preCenter:",preCenter);  *printCenter*("nextCenter:",nextCenter);  **double** distance = *centerDistance*(preCenter,nextCenter);  **if**(distance<delta){  **System**.***out***.println("delta:"+distance);  **return false**;  }   **return true**;  }   */\*\*  \* 两个聚类中心的距离d  \* Pij 代表前次聚类中心的点；  \* Nij 代表前次聚类中心的点；  \* 则 d = sqrt(sum((Pij-Nij)^2))  \* @param preCenter  \* @param nextCenter  \* @return  \*/* **private static double** centerDistance(List<**Instance**> preCenter,  List<**Instance**> nextCenter) {  **double** distance =0.0;  *// 参考公式完善功能* **for**(**int** i =0 ;i< preCenter.size();i++){  distance += *calDistance*(preCenter.get(i),nextCenter.get(i));  }  **return Math**.*sqrt*(distance);  }   */\*\*  \* 打印聚类中心  \* @param info  \* @param preCenter  \*/* **private static void** printCenter(**String** info, List<**Instance**> preCenter) {  **System**.***out***.println(info);  **for**(**Instance** centerI:preCenter){  **for**(**double** centerIJ:centerI.getData()){  **System**.***out***.print(centerIJ);  **System**.***out***.print(",");  }  **System**.***out***.println();  }  }  */\*\*  \* 根据前一次聚类中心，计算下一次聚类中心  \* @param data  \* @param preCenter  \* @return  \*/* **private static** List<**Instance**> getCenter(List<**Instance**> data, List<**Instance**> preCenter) {  **double**[][] nextCenter = **new double**[preCenter.size()][];  **int**[] count = **new int**[preCenter.size()];  **int** lineIndex =-1;  **for**(**Instance** line :data){  lineIndex = *getLineIndex*(line,preCenter);  *addToNextCenter*(nextCenter,line,lineIndex);  count[lineIndex]++;  }   List<**Instance**> nextCenter\_ = **new** ArrayList<>();  **for**(**int** i=0;i<nextCenter.length;i++){  *average*(nextCenter[i],count[i]);  nextCenter\_.add(**new** Instance(nextCenter[i]));  }   **return** nextCenter\_;  }  */\*\*  \* 添加到对应的值上  \* @param preCenter  \* @param line  \* @param lineIndex  \*/* **private static void** addToNextCenter(**double**[][] preCenter, **Instance** line, **int** lineIndex) {  **if**(preCenter[lineIndex] == **null**){  preCenter[lineIndex] = **new double**[line.getData().length];  }  **for**(**int** j=0;j<preCenter[lineIndex].length;j++){  preCenter[lineIndex][j]+=line.getData()[j];  }  }   */\*\*  \* 求平均值  \* @param centerI  \* @param count  \* @return  \*/* **private static void** average(**double**[] centerI, **long** count) {  **for**(**int** i=0;i<centerI.length;i++){  centerI[i] = centerI[i]/count;  }  **return** ;  }   */\*\*  \* 获取当前line距离最近的聚类中心向量下标  \* @param line  \* @param preCenter  \* @return  \*/* **private static int** getLineIndex(**Instance** line, List<**Instance**> preCenter) {  **int** index =-1;  **double** distance = **Double**.***MAX\_VALUE***;  **double** currDistance=0.0f;  **for**(**int** i=0;i<preCenter.size();i++){  currDistance = *calDistance*(line,preCenter.get(i));  **if**(currDistance<distance){  distance=currDistance;  index= i;  }  }  **return** index;  }   */\*\*  \* 求欧式距离  \* distance= sqrt((line(1)-centerI(1))^2+(line(2)-centerI(2))^2+...)  \* @param line  \* @param centerI  \* @return  \*/* **private static double** calDistance(**Instance** line, **Instance** centerI) {  **double** sum =0.0;  *// 参考公式完成代码* **double**[] data1 = line.getData();  **double**[] data2 = centerI.getData();  **for**(**int** i = 0 ;i< data1.length;i++){  sum += (data1[i] - data2[i]) \* (data1[i] - data2[i]);  }  **return Math**.*sqrt*(sum);  }  */\*\*  \* 获取随机聚类中心  \* @param data  \* @param k  \* @return  \*/* **private static** List<**Instance**> getRandomCenter(List<**Instance**> data, **int** k) {  Set<**Integer**> centerIndex = **new** HashSet<>();  **Random** random = **new** Random(*seed*);  **while**(centerIndex.size()<k){  centerIndex.add(random.nextInt(data.size()));  }  List<**Instance**> center = **new** ArrayList<>();  **int** i=0;  **for**(**Integer** index:centerIndex){  center.add( data.get(index));  }  **return** center;  } } |

### 编写测试

在test下新建包：



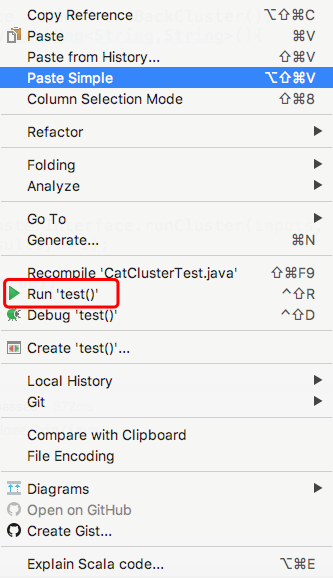
复制fansy下面的类到cat目录，并改名字为CatClusterTest



修改其中的test函数：



右键运行：



测试通过即可：

