1. 输入 DataSet<Tuple3<String, String, Integer>> triplets

//人，歌，听多少次

new Tuple3<String, String, Integer>("user\_1", "song\_1", 100);

1. 获得bad记录(歌曲id不能被信任)的id集合mismatches

DataSet<Tuple1<String>> mismatches = getMismatchesData(env).map(new ExtractMismatchSongIds());

其中bad记录的形式是：“ERROR: <song\_8 track\_8> Sever”，提取出song\_8

1. 从triplets中过滤song\_id为mismatches的记录，得到有效集合validTriplets

DataSet<Tuple3<String, String, Integer>> validTriplets = triplets

.coGroup(mismatches).where(1).equalTo(0)

.with(new FilterOutMismatches());

1. 生成用户->歌曲二分图，图中边的属性为用户播放歌曲的次数

Graph<String, NullValue, Integer> userSongGraph = Graph.*fromTupleDataSet*(validTriplets, env);

1. 保留用户听歌曲次数最多的边

DataSet<Tuple2<String, String>> usersWithTopTrack = userSongGraph

.groupReduceOnEdges(new GetTopSongPerUser(), EdgeDirection.OUT)

.filter(new FilterSongNodes());

|  |
| --- |
| public static final class GetTopSongPerUser implements EdgesFunctionWithVertexValue<String, NullValue, Integer, Tuple2<String, String>> {   public void iterateEdges(Vertex<String, NullValue> vertex, Iterable<Edge<String, Integer>> edges, Collector<Tuple2<String, String>> out) throws Exception {   int maxPlaycount = 0;  String topSong = "";  for (Edge<String, Integer> edge : edges) {  if (edge.getValue() > maxPlaycount) {  maxPlaycount = edge.getValue();  topSong = edge.getTarget();  }  }  out.collect(new Tuple2<String, String>(vertex.getId(), topSong));  } } |

1. 生成一个user -> user的Edge集合，其中每条边连接的两个用户喜欢同一个歌曲。

|  |
| --- |
| DataSet<Edge<String, NullValue>> similarUsers = userSongGraph  .getEdges()  // filter out user-song edges that are below the playcount threshold  .filter(new FilterFunction<Edge<String, Integer>>() {  public boolean filter(Edge<String, Integer> edge) {  return (edge.getValue() > *playcountThreshold*);  }  }).groupBy(1) //根据歌曲id将user->song的Edge分组  .reduceGroup(new CreateSimilarUserEdges()).distinct(); //对同一组内的user进行互连 |

|  |
| --- |
| public static final class CreateSimilarUserEdges implements GroupReduceFunction<Edge<String, Integer>, Edge<String, NullValue>> {   public void reduce(Iterable<Edge<String, Integer>> edges, Collector<Edge<String, NullValue>> out) {  List<String> listeners = new ArrayList<String>();  for (Edge<String, Integer> edge : edges) {  listeners.add(edge.getSource());  }  for (int i = 0; i < listeners.size() - 1; i++) {  for (int j = i + 1; j < listeners.size(); j++) {  out.collect(new Edge<String, NullValue>(listeners.get(i),  listeners.get(j), NullValue.*getInstance*()));  }  }  } } |

1. 生成user-user的用户相似图

Graph<String, Long, NullValue> similarUsersGraph = Graph.fromDataSet(similarUsers,

new MapFunction<String, Long>() {

public Long map(String value) {

return 1l;

}

}, env).getUndirected();

1. 生成一个user-communityid的集合

|  |
| --- |
| DataSet<Tuple2<String, Long>> idsWithInitialLabels = DataSetUtils  .*zipWithUniqueId*(similarUsersGraph.getVertexIds())  .map(new MapFunction<Tuple2<Long, String>, Tuple2<String, Long>>() {  @Override  public Tuple2<String, Long> map(Tuple2<Long, String> tuple2) throws Exception {  return new Tuple2<String, Long>(tuple2.f1, tuple2.f0);  }  }); |

1. 将每个user的communityid赋值到similarUsersGraph的顶点属性中，并在similarUsersGraph上运行LPA算法。

|  |
| --- |
| DataSet<Vertex<String, Long>> verticesWithCommunity = similarUsersGraph  .joinWithVertices(idsWithInitialLabels,  new VertexJoinFunction<Long, Long>() {  public Long vertexJoin(Long vertexValue, Long inputValue) {  return inputValue;  }  }).run(new LabelPropagation<String, Long, NullValue>(*maxIterations*)); |

1. 打印verticesWithCommunity，(user, user所属的社区)如下：

(user\_2,10)

(user\_1,6)

(user\_3,6)

(user\_5,6)

(user\_6,10)

(user\_4,6)