基于聚类优化的KNN分类器——细菌菌落分类

15352445 钟思根 15352441 钟丹彬 15352447 钟镇威

2018 年 1 月 13 日

摘要

单个细菌形状千姿百态，而细菌菌落也各有特点。微生物在自然界中起着非常重要的作用，近年研究、培养、提取微生物及其衍生物成为热点。细菌等微生物检验的结果为各种诊断提供了重要的参考依据，但是，临床微生物检验工作正面临着诸多问题，检验环境越来越复杂，检验结果对临床医学的作用越来越明显，患者对检验工作的要求越来越高。从这个角度讲，需要重视临床微生物检验的地位和作用，通过多种方式提高检验的精度和准确度，才能为临床工作提供更多、更有价值的指导信息。临床微生物检验是一项精度极高、操作极为复杂的工作，需要在实践和理论研究的过程中不断的加以完善和改进。本实验应用具有噪声的基于密度的空间聚类算法(Density-Based Spatial Clustering of Application with Noise，DBSCAN)和k近邻分类(k-nearest neighbor classification，KNN)，对微生物及其形成的菌落进行聚类和分类，得出菌落个数、菌落种类等信息，辅助研究人员完成繁复的微生物检验操作。

目录

1 需求建模

单个细菌形状千姿百态，而细菌菌落也各有特点。微生物在自然界中起着非常重要的作用，近年研究、培养、提取微生物及其衍生物成为热点。

细菌等微生物检验的结果为各种诊断提供了重要的参考依据，但是，临床微生物检验工作正面临着诸多问题，检验环境越来越复杂，检验结果对临床医学的作用越来越明显，患者对检验工作的要求越来越高。从这个角度讲，需要重视临床微生物检验的地位和作用，通过多种方式提高检验的精度和准确度，才能为临床工作提供更多、更有价值的指导信息。临床微生物检验是一项精度极高、操作极为复杂的工作，需要在实践和理论研究的过程中不断的加以完善和改进。

本实验应用具有噪声的基于密度的空间聚类算法(Density-Based Spatial Clustering of Application with Noise，DBSCAN)和k近邻分类(k-nearest neighbor classification，KNN)，对微生物及其形成的菌落进行聚类和分类，得出菌落个数、菌落种类等信息，辅助研究人员完成繁复的微生物检验操作。

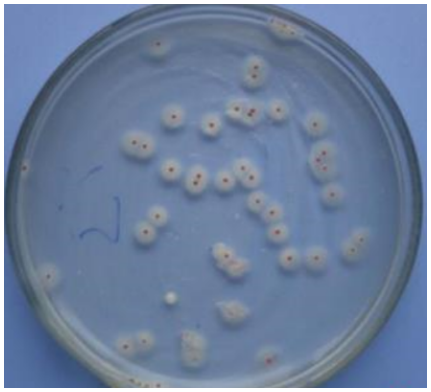


图 1: 放线菌聚落

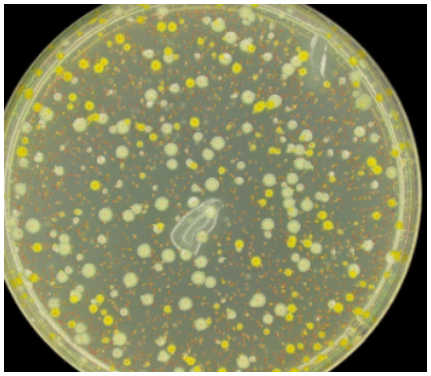


图 2: 黄单胞杆菌聚落

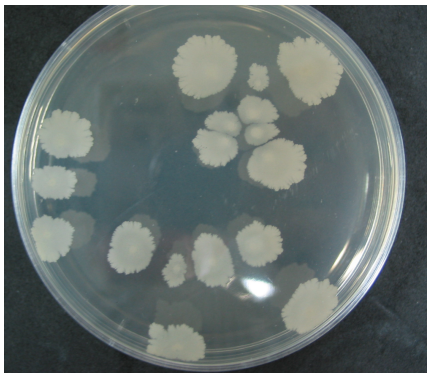


图 3: 枯草芽孢杆菌聚落

图 4: 欧文氏菌聚落

1.1 实验内容

当研究人员完成前期的提取、育菌种、培养提纯菌种后，得到培养基中，一个或多个细菌的菌落，将菌落进行特征提取后，本实验的算法作用是辅助研究人员，计算出该培养基内的菌种个数以及种类。使用

基于密度聚类算法DBSCAN算法对细菌菌落进行聚类并计算提取出每个菌落，再对菌落应用K近邻算法标记菌落，完成后研究人员可使用该算法的结论进行参考。

1.2 创新点

将基于密度聚类算法和K近邻算法应用于细菌研究的辅助算法，能够处理大型数据，算法具体实现中，创新性的设计了RDD之间进行数学运算的方法。噪声和非相关性特征的存在，或特征尺度与它们的重要性不一致会使K近邻算法的准确性严重降低。而DBSCAN算法拥有良好的去噪属性和选取特征的能力，利用聚类算法提高KNN的准确度。

2 基本配置

平台：spark+hadoop+scala
java-jdk 1.8.0
spark 1.6.3
scala 2.10.5
hadoop 2.6.0

2.1 环境搭建

在实验课lab1 6的基础之上，使用scala语言进行算法实现，scala、sbt等安装操作只需在master上进行

1. 安装scala
使用lrzszd的rz命令，上传scala-2.11.12.tgz至master
解压并改名至/usr/local/scala
2. 配置环境变量
将scala加入环境变量，完整环境变量配置如下

图 5: 环境变量

```
export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/jvm/jdk1.8.0_60/bin:/usr/lo
sr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/jvm/jdk1.8.0_60/bin:/usr/local/hive/bin

export HADOOP_HOME=/usr/local/hadoop
export HIVE_HOME=/usr/local/hive
export JAVA_HOME=/usr/local/jvm/jdk1.8.0_60
export JRE_HOME=${JAVA_HOME}/jre
export SPARK_HOME=/usr/local/spark
export SCALA_HOME=/usr/local/scala
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:${SPARK_HOME}/lib:${SCALA_HOME}/lib:$($HADOOP_HOME/bin/hadoop classpath):$CLASSPATH
export PATH=$PATH:${JAVA_HOME}/bin:$HADOOP_HOME/bin:$HADOOP/sbin:$PATH:${HIVE_HOME}/bin:$PATH:${SCALA_HOME}/bin
~
```

3. 安装sbt，用于生成jar包使用lrzszd的rz命令，上传sbt-1.0.3.tgz至master
解压并改名至/usr/local/sbt
vi 新建脚本文件如下

图 6: sbt配置文件

```
#!/bin/bash
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256M"
java $SBT_OPTS -jar `dirname $0`/sbt-launch.jar "$@"
~
```

保存脚本文件并给予权限为sbt脚本增加可执行权限 chmod u+x sbt

4. 配置sbt对于实验project的打包脚本 vi 新建脚本文件 simple.sbt

图 7: simple.sbt

```
name := "Simeple Project"

version :="1.0"

scalaVersion :="2.10.5"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.6.3"
```

2.2 具体运行流程

1.新建项目文件夹sparkapp，保证文件夹的内容和结构如下
simple.sbt要放在 /sparkapp/下，是打包的配置文件
其中/src/main/scala目录下存放 KNN.scala,DBSCAN.scala等scala编程文件

图 8: 项目文件夹目录结构

```
hadoop@master:~/sparkapp$ find .
.
./simple.sbt
./src
./src/main
./src/main/scala
./src/main/scala/DBSCAN.scala
```

2.把项目文件达成jar包，更方便直接提交到spark集群上运行
键入命令sudo /usr/local/sbt/sbt package
第一次打包需要一些时间，之后的打包过程会比较快
若代码没有错误，则显示如下截图

图 9: sbt打包成功

```
hadoop@master:~/sparkapp$ sudo /usr/local/sbt/sbt package
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=256M; support was removed in 8.0
[info] Set current project to Simeple Project (in build file:/home/hadoop/sparkapp/)
[info] Updating {file:/home/hadoop/sparkapp/}sparkapp...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/hadoop/sparkapp/target/scala-2.10/classes...
[info] Packaging /home/hadoop/sparkapp/target/scala-2.10/simeple-project_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 25 s, completed Jan 13, 2018 6:21:13 PM
```

打包后的jar包在这

图 10: jar包存放位置

```
hadoop@master:~/sparkapp/target/scala-2.10$ ls
classes  simeple-project_2.10-1.0.jar
hadoop@master:~/sparkapp/target/scala-2.10$ pwd
/home/hadoop/sparkapp/target/scala-2.10
```

3.开启hadoop，spark，historyserver
/usr/local/hadoop/sbin/start-all.sh

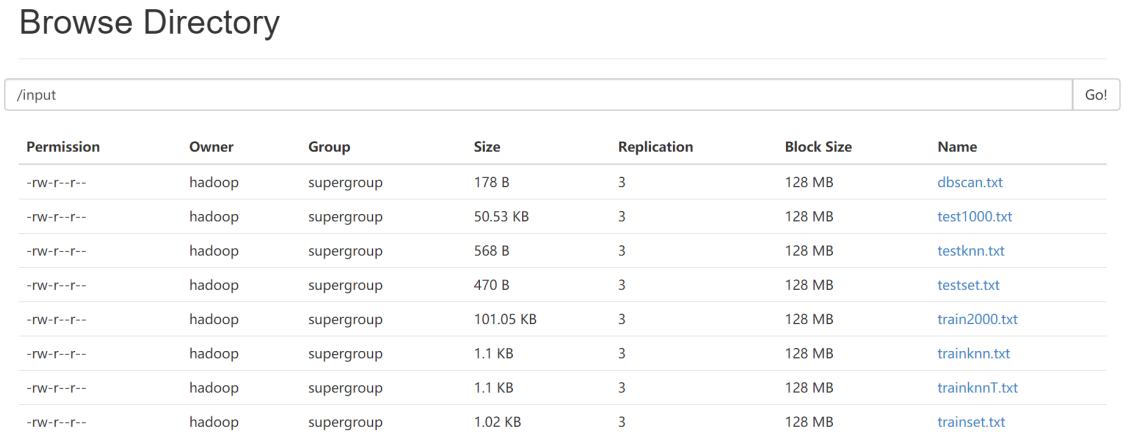
/usr/local/spark/sbin/start-all.sh

/usr/local/spark/sbin/start-history-server.sh

4.将输入文件上传至hdfs

/usr/local/hadoop/bin/hdfs dfs -put /usr/local/data/*.txt /input

图 11: 将输入文件上传至hdfs



5.提交jar包至spark集群，使用cluster模式

/usr/local/spark/bin/spark-submit

–deploy-mode cluster

–master yarn –driver-memory 1G

–class ”SimpleKnn”

–executor-memory 1G

–total-executor-cores 2

/sparkapp/target/scala-2.10/simeple-project-2.10-1.0.jar hdfs://master:9000/input/train2000.txt hdfs://master:9
5

6.运行中运行后需要等待片刻

图 12: 运行中截图

```
18/01/13 18:43:38 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:39 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:40 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:41 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:42 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:43 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:44 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:45 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:46 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:47 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 18:43:48 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
```

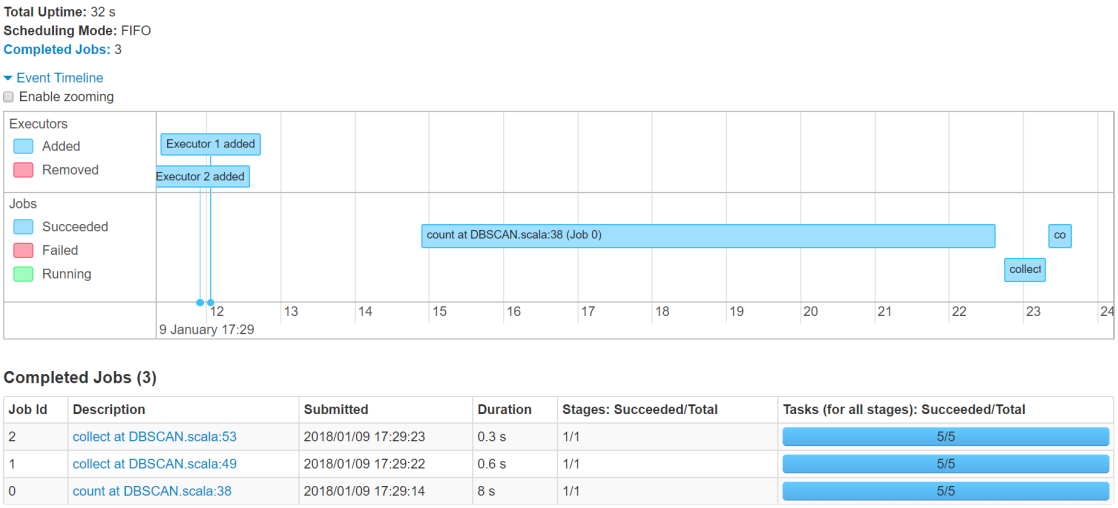
图 13: 运行成功截图

```
18/01/13 20:46:10 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 20:46:11 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 20:46:12 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 20:46:13 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 20:46:14 INFO yarn.Client: Application report for application_1515839833128_0001 (state: RUNNING)
18/01/13 20:46:15 INFO yarn.Client: Application report for application_1515839833128_0001 (state: FINISHED)
18/01/13 20:46:15 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 192.168.142.128
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1515840136918
         final status: SUCCEEDED
         tracking URL: http://master:8088/proxy/application_1515839833128_0001/history/application_1515839833128_0001/1
         user: hadoop
18/01/13 20:46:16 INFO util.ShutdownHookManager: Shutdown hook called
18/01/13 20:46:16 INFO util.ShutdownHookManager: Deleting directory /usr/local/spark/spark-8225e71d-d89f-4d10-8ca1-409cc3a2cf17
```

7. 查看结果

a.使用webIU查看历史运行记录192.168.142.128:18080

图 14: 使用webIU查看DBSCAN的历史运行记录



Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	collect at DBSCAN.scala:53	2018/01/09 17:29:23	0.3 s	1/1	5/5
1	collect at DBSCAN.scala:49	2018/01/09 17:29:22	0.6 s	1/1	5/5
0	count at DBSCAN.scala:38	2018/01/09 17:29:14	8 s	1/1	5/5

b.查看运行输出日志

/usr/local/hadoop/bin/yarn logs -applicationId XXXXXXXX

c.查看输出文件（即结果文件）

cat /usr/local/data/out/part* > out.txt

因为文件是分布式存储的，我在设计算法是设置分成五个part，所以查看输出结果需要使用cat命令将多个结果文件合并

图 15: 结果输出到文件(KNN分类器结果)形式为(index,label)

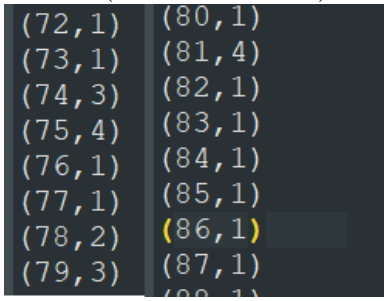


图 16: 查看输出文件

```
hadoop@master:/usr/local/data/out$ ls
out.txt  part-00000  part-00001  part-00002  part-00003  part-00004  part-00005  _SUCCESS
```

3 核心算法

3.1 DBSCAN原理

DBSCAN(Density-Based Spatial Clustering of Application with Noise) 具有噪声的基于密度的空间聚类算法该算法将具有足够密度的区域划分为簇，并在具有噪声的空间数据中发现任意形状的簇，将簇定义为密度相连的点的最大集合。该算法利用基于密度聚类的概念，要求聚类空间中的一定区域内(eps)所包含的对象不小于某一给定阈值(MinPts)，DBSCAN的显著优点是聚类速度快，且能够有效的处理噪声点和发现任意形状的空间聚类。DBSCAN直接对整个数据集进行操作，且进行聚类时，使用了全局性的表征密度的参数。

（1）Eps邻域：给定对象半径Eps内的邻域称为该对象的Eps邻域;

（2）核心点（core point）：如果对象的Eps邻域至少包含最小数目MinPts的对象，则称该对象为核心对象;

（3）边界点（edge point）：边界点不是核心点，但落在某个核心点的邻域内;

（4）噪音点（outlier point）：既不是核心点，也不是边界点的任何点;

（5）直接密度可达(directly density-reachable)：给定一个对象集合D，如果p在q的Eps邻域内，而q是一个核心对象，则称对象p从对象q出发时是直接密度可达的;

（6）密度可达(density-reachable)：如果存在一个对象链 p1, ⋯,pi,.., pn，满足p1 = p 和pn = q，pi是从pi+1关于Eps和MinPts直接密度可达的，则对象p是从对象q关于Eps和MinPts密度可达的;

（7）密度相连(density-connected)：如果存在对象O∈D，使对象p和q都是从O关于Eps和MinPts密度可达的，那么对象p到q是关于Eps和MinPts密度相连的。

（8）簇（cluster）:称非空集合$C \subset X$是X的一个簇（cluster）,如果它满足： (a)(Maximality)对于$x, y \in C$,且y是从x密度可达的，则$y \in C$ (b)(Connectivity)若$x \in C, y \in C$,则x, y是密度相连的

满足(a),(b)则称构成一个类簇

核心点、边界点、噪音点以及直接密度可达、密度可达和密度相连解释如图

图 17: 红色为核心点、黄色为边界点、蓝色为噪声点

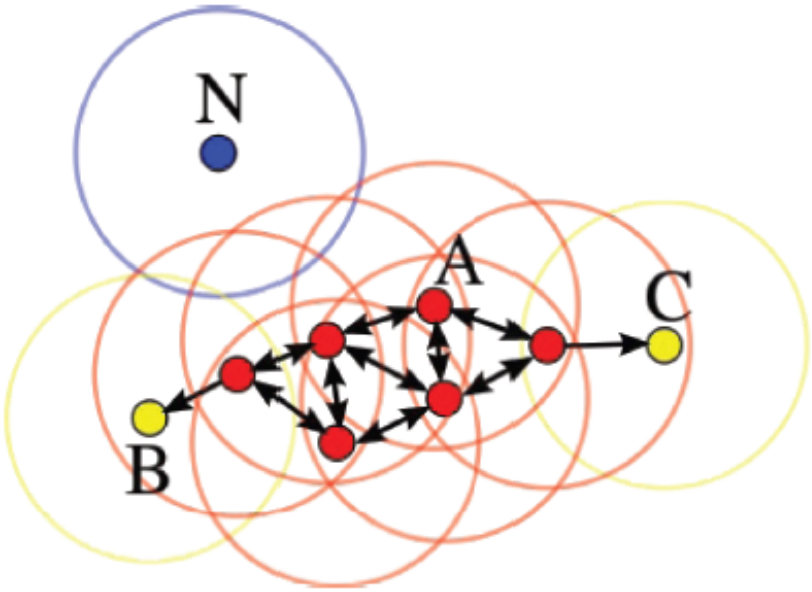
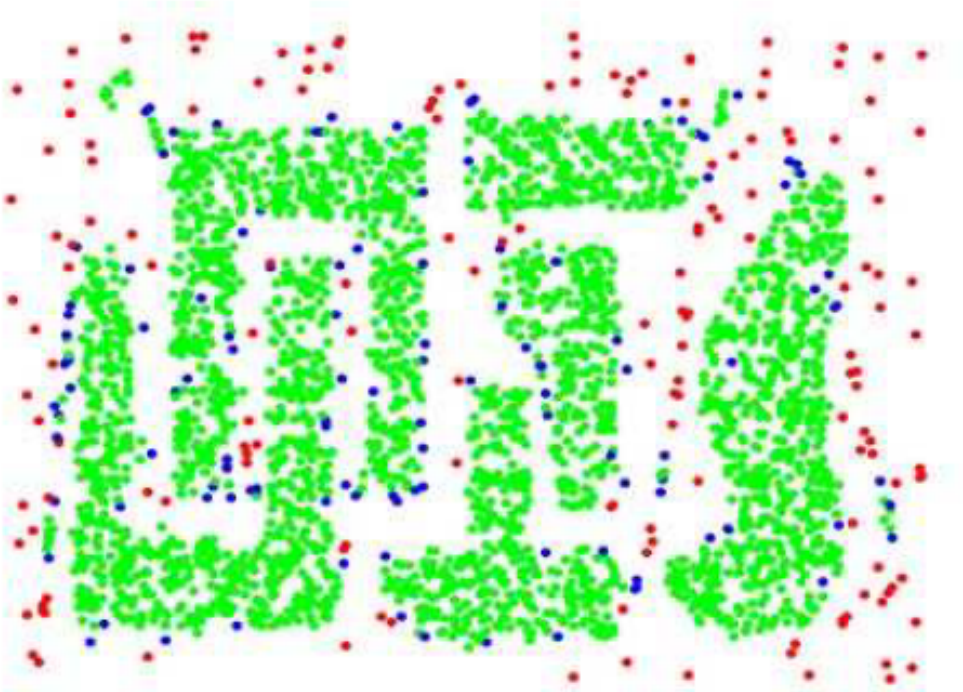


图 18: 基于密度聚类可形成任意形状的簇



DBSCAN的主要优点有： 1）可以对任意形状的稠密数据集进行聚类，相对的，K-Means之类的聚类算法一般只适用于凸数据集。

2）可以在聚类的同时发现异常点，对数据集中的噪声点不敏感。

3）聚类结果没有偏倚，相对的，K-Means之类的聚类算法随机初始值对聚类结果有很大影响。

3.2 DBSCAN伪代码

Algorithm 1 Density-Based Spatial Clustering of Application with Noise

Input: D: A dataset

Eps：Neighborhood radius

Minpts：The minimum number of points required for high-density areas

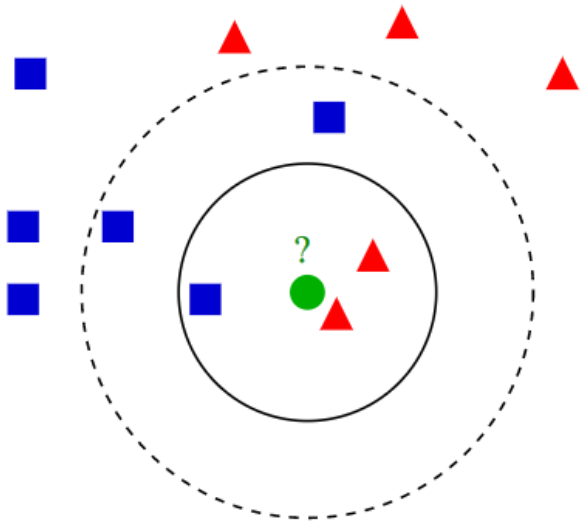
Output: A collection of density-based clusters

```
1: ----------
2: DBSCAN(D, eps, MinPts)
3: C = 0
4: For each point P in dataset D
5: if P is visited : continue next point,mark P as visited
6: NeighborPts = regionQuery(P, eps)
7: if sizeof(NeighborPts) < MinPts : mark P as NOISE
8: else : C = next cluster,expandCluster(P, NeighborPts, C, eps, MinPts)
9: ----------
10: expandCluster(P, NeighborPts, C, eps, MinPts)
11: add P to cluster C
12: For each point P' in NeighborPts
13: if P' is not visited : mark P' as visited,NeighborPts' = regionQuery(P', eps)
14: if sizeof(NeighborPts') >= MinPts : NeighborPts = NeighborPts joined with NeighborPts'
15: if P' is not yet member of any cluster : add P' to cluster C
16: ----------
17: regionQuery(P, eps)
18: return all points within P's eps-neighborhood (including P)
```

3.3 KNN原理

最近邻居法（KNN算法，称K-近邻算法）是一种用于分类的非参数统计方法。思想简述为包含特征空间中的k个最接近的训练样本为参考进行测试数据的预测。在KNN分类中，输出是一个分类族群。一个对象的分类是由其邻居的“多数表决”确定的，k个最近邻居（k为正整数，通常较小）中最常见的分类决定了赋予该对象的类别。若k = 1，则该对象的类别直接由最近的一个节点赋予。最近邻居法采用向量空间模型来分类，概念为相同类别的案例，彼此的相似度高，而可以借由计算与已知类别案例之相似度，来评估未知类别案例可能的分类。KNN是一种基于实例的学习，或者是局部近似和将所有计算推迟到分类之后的惰性学习。k-近邻算法是所有的机器学习算法中最简单的之一。无论是分类还是回归，衡量邻居的权重都非常有用，使较近邻居的权重比较远邻居的权重大。例如，一种常见的加权方案是给每个邻居权重赋值为1/ d，其中d是到邻居的距离。噪声和非相关性特征的存在，或特征尺度与它们的重要性不一致会使K近邻算法的准确性严重降低。所以本实验利用DBSCAN的对噪声不敏感的特性进行优化，旨在提高KNN的准确度。

图 19: K近邻算法图示



3.4 KNN伪代码

Algorithm 2 k-nearest neighbor classification

Input: *dataMat*: TrainSet with label,TestSet;
K: the K-value of KNN

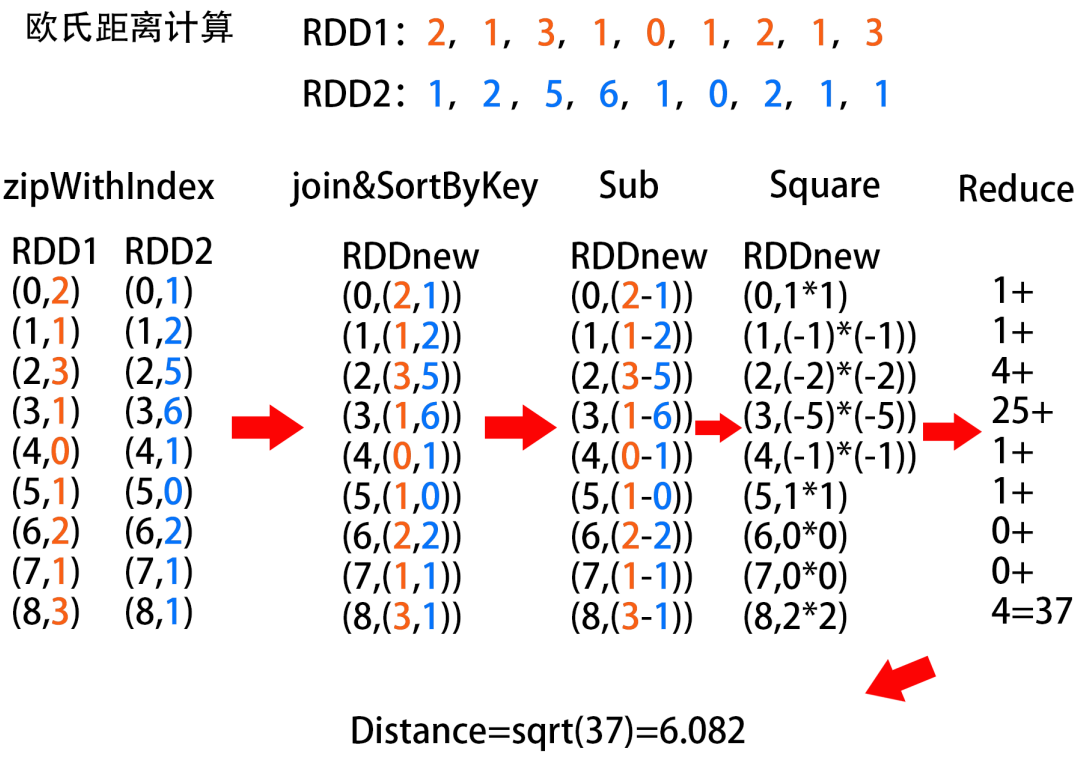
Output: A series label for TestSet

1: load TrainSet,TestSet
2: filter some useless character
3: compute each test-sample distances with all train-sample
4: Sort the distances from small to large
5: Select the first K distance samples for classification
6: Choose the most appear label for prediction

3.5 实现难点和关键技术

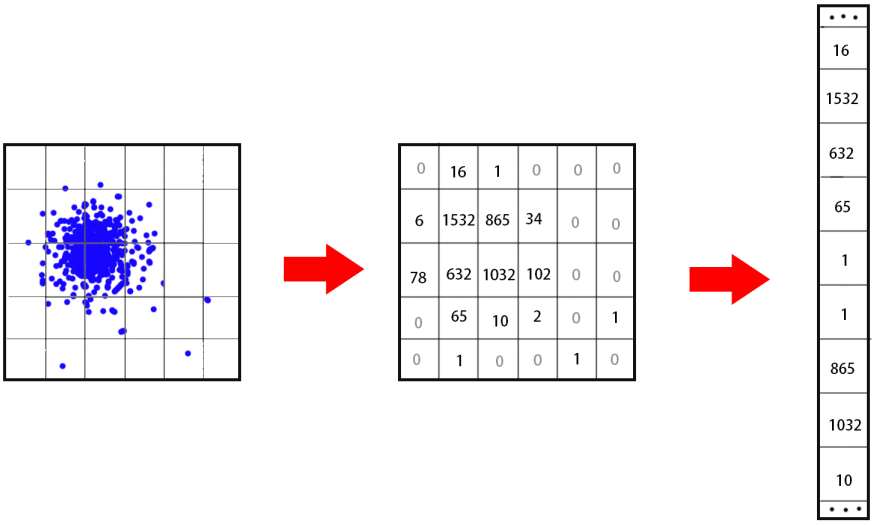
1.两个RDD之间实现数学运算，而不是先使用collect，toArray，toVector算子转出为数组或者vector之后，再进行运算符的重载

图 20: 两个不同的RDD之间计算欧式距离



2.KNN分类，将聚类结果归一化，二值化后，展成序列输入KNN分类器

图 21: 分类器工作预处理部分



4 实验结果

由于Ubuntu输出的结果只是数据，所以把展示数据使用matlab可视化的

图 22: 聚类结果

```
Vector(7.5, 9.0)
Vector(8.0, 8.5)
Vector(8.5, 8.0)
Vector(9.0, 7.5)
簇0包含的元素如下:
Vector(9.0, 7.5)
簇5包含的元素如下:
Vector(5.0, 8.0)
Vector(5.5, 8.5)
簇1包含的元素如下:
Vector(2.0, 5.0)
Vector(2.5, 4.5)
簇6包含的元素如下:
Vector(6.0, 9.0)
Vector(6.5, 9.0)
Vector(7.0, 9.0)
Vector(7.5, 9.0)
簇2包含的元素如下:
Vector(3.0, 4.0)
Vector(3.5, 3.5)
Vector(4.0, 3.5)
Vector(4.5, 3.5)
Vector(5.0, 3.5)
簇7包含的元素如下:
Vector(8.0, 8.5)
Vector(8.5, 8.0)
簇3包含的元素如下:
Vector(5.5, 4.0)
Vector(6.0, 4.5)
簇4包含的元素如下:
Vector(6.5, 5.0)
Vector(4.5, 7.5)
噪音点如下:
Vector(2.0, 5.0)
Vector(3.0, 4.0)
Vector(5.5, 4.0)
```

注：下图中黑色部分为聚类算法判定的噪声点，这些噪声点将不会参与分类过程。

数据集1聚类算法统计菌落个数为6个，分类结果为3种

Cluster1 种类1,分类正确

Cluster2 种类1,分类正确

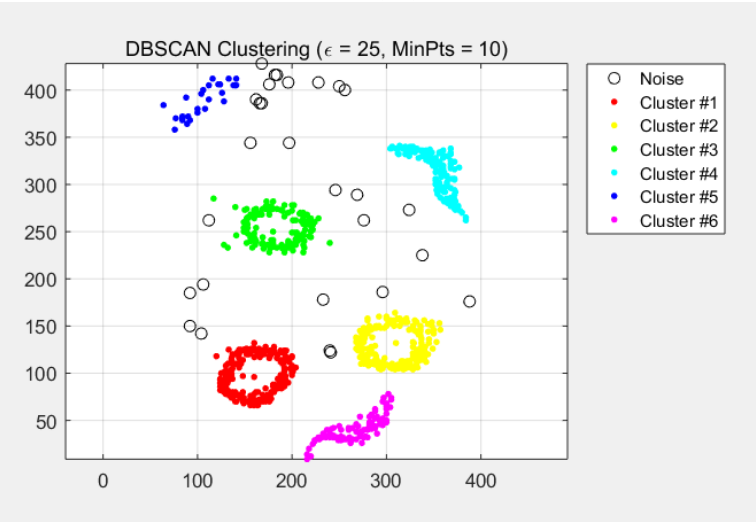
Cluster3 种类1,分类正确

Cluster4 种类2,分类正确

Cluster5 种类2,分类正确

Cluster6 种类3,分类正确

图 23: 数据集1

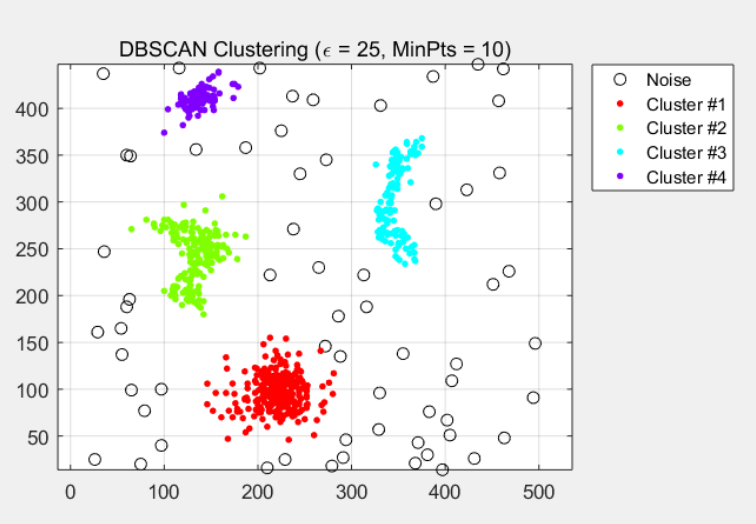


数据集2聚类算法统计菌落个数为4个，分类结果为2种

Cluster1 种类4,分类正确

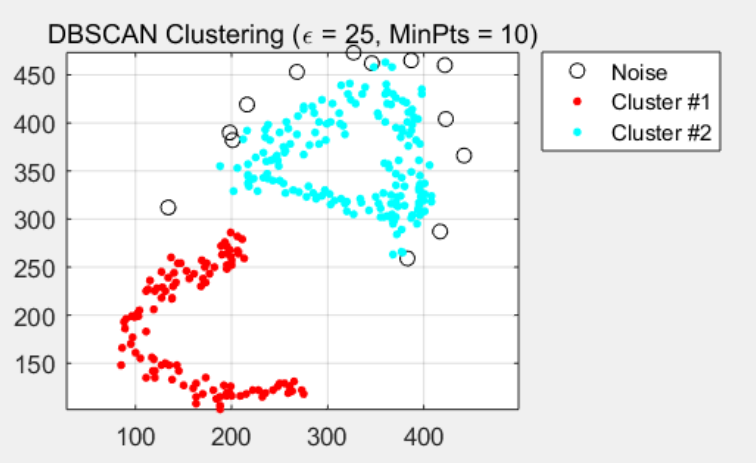
Cluster2 种类4,分类正确
Cluster3 种类2,分类正确
Cluster4 种类4,分类错误

图 24: 数据集2



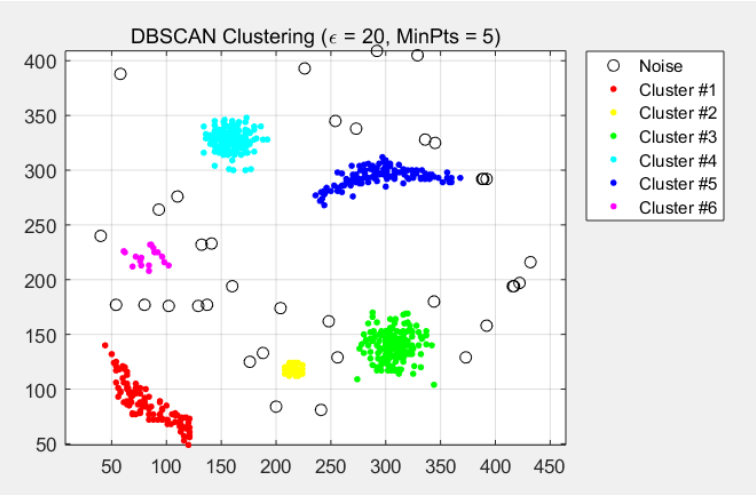
数据集3聚类算法统计菌落个数为2个，分类结果为2种
Cluster1 种类1,分类正确
Cluster2 种类5,分类正确

图 25: 数据集3



数据集4聚类算法统计菌落个数为6个，分类结果为2种
Cluster1 种类2,分类正确
Cluster2 种类4,分类正确
Cluster3 种类4,分类正确
Cluster4 种类4,分类正确
Cluster5 种类2,分类错误
Cluster6 种类4,分类错误

图 26: 数据集4



数据集5聚类算法统计菌落个数为8个，分类结果为2种

Cluster1 种类1,分类错误

Cluster2 种类2,分类错误

Cluster3 种类2,分类错误

Cluster4 种类2,分类错误

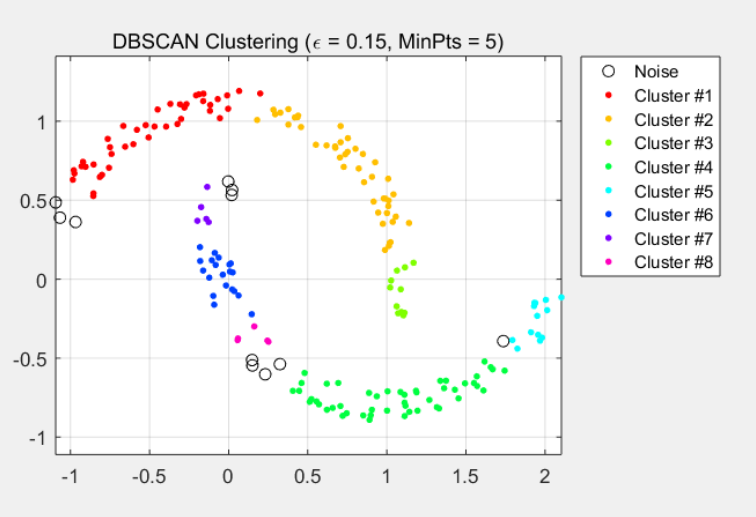
Cluster5 种类2,分类错误

Cluster6 种类2,分类错误

Cluster7 种类1,分类错误

Cluster8 种类1,分类错误

图 27: 数据集5：半径eps为0.15，最小邻域点阈值MinPts为5的聚类结果

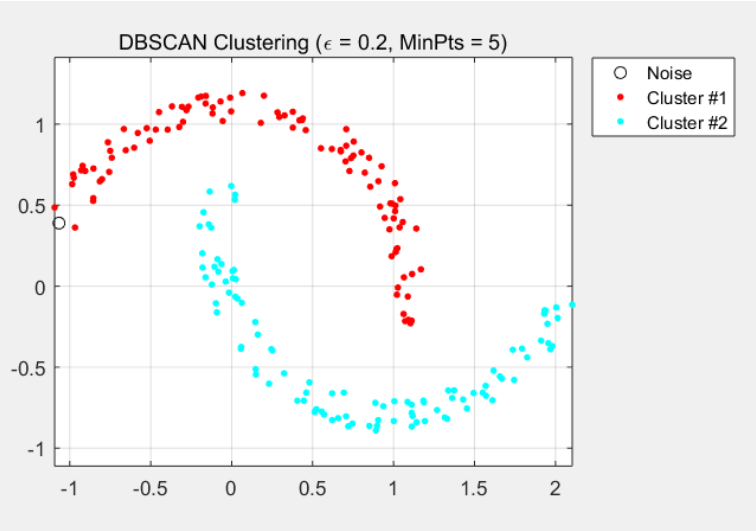


经过调整半径和阈值后，数据集5聚类算法统计菌落个数为2个，分类结果为1种

Cluster1 种类1,分类错误

Cluster2 种类2,分类正确

图 28: 数据集5：半径eps为0.2，最小邻域点阈值MinPts为5的聚类结果



但实际上他们是同一种菌落。

5 总结

检验的自动化,大大方便了微生物检验工作的开展,提高了检验速度和质量,在目前情况下，微生物检验的结果为临床诊断提供了重要的参考依据，但是，临床微生物检验工作正面临着诸多问题，检验环境越来越复杂，检验结果对临床医学的作用越来越明显，患者对检验工作的要求越来越高。从这个角度讲，医院或者医学实验室只有重视临床微生物检验的地位和作用，通过多种方式提高检验的精度和准确度，才能为临床工作提供更多、更有价值的指导信息.临床微生物检验是一项精度极高、操作极为复杂的工作，需要在实践和理论研究的过程中不断的加以完善和改进。

我们小组做本次实验的意义是希望在大数据平台非常火热时候，微生物的相关研究也能够在该平台得到应用，可以作为辅助手段帮助研究人员更好的进行实验。设计的通过聚类优化的KNN的分类器表现较好，但是还有很大的改进的空间。对于一些特殊的菌落鲁棒性不好，并且关于参数的调节，也不能适应所有的菌落。改进方向是设计算法自动调节参数，并且加入研究人员的手动标注，使得算法能自行学习被错误划分的样本。

6 小组分工

6.1 小组成员信息

上课时段：周四下午7-10节
组号：21
组长: 15352445 钟思根
组员：15352441 钟丹彬
组员：15352447 钟镇威

6.2 小组分工和贡献

组长: 15352445 钟思根 33.3%

组员：15352441 钟丹彬 33.3%

组员：15352447 钟镇威 33.3%

7 源代码

7.1 KNN.scala

```
//dbscan.scala
import java.util.Random
import scala.math.exp
import org.apache.hadoop.conf.Configuration
import org.apache.spark._
import org.apache.spark.scheduler.InputFormatInfo
import scala.math._
import scala.collection.mutable.ArrayBuffer

object SimpleKnn {
  def main(args: Array[String]) {
       //read traindata,testdata,and the K of K-Nearist Neighbor Algorithm
    if (args.length < 2) {
      System.err.println("Usage: SimpleApp <filetrain> <filetest> <kvalue>")
      System.exit(1)
    }

       val kvalue = args(2).toInt
       //output the K on console
    println("kvalue: " + kvalue)

    //configuration spark runable
       val conf = new SparkConf().setAppName("Simple")
       val sc = new SparkContext(conf)
       //Set input data to distributed parallelized storage
       val trainFile = sc.textFile(args(0), 5);
       val testFile = sc.textFile(args(1), 5);

       //Calculate the number of training sets
       val train_count = trainFile.count()
       println("train_count line : " + train_count)

       //Calculate the number of testing sets
       val test_count = testFile.count()
       println("test_count line : " + test_count)

       //Split the data and extract it
       var testdata=testFile.map(line1 => line1.split(" ")).map(word1 => word1.map(x1 =>
           x1.toDouble)).collect()

       val traindata=trainFile.map(line2 => line2.split(" ")).map(word2 => word2.map(x2 =>
           x2.toDouble)).collect()
```

```
    //Algorithm results stored
    val test_label=ArrayBuffer[Tuple2[Double, Int]]()

    //Cartesian product
for( a <- 0 to test_count-1)
    {
           //Subscript of the sample data by index(0 ~ n)
    val test=sc.parallelize(testdata(a).zipWithIndex.map(q=>(q._2,q._1)))

    //Filter some features of the sample to remove unwanted subscripts, such as sample numbers
        and label
    val datatestfilter=test.filterByRange(1,12)

           //distance results stored
           val distance = ArrayBuffer[Tuple4[Double, Double, Double, Int]]()

           for(c <- 0 to train_count-1)
           {
                  //Subscript of the sample data by index(0 ~ n)
                  val train=sc.parallelize(traindata(c).zipWithIndex.map(q=>(q._2,q._1)))

                  //Filter some features of the sample to remove unwanted subscripts, such as
                      sample numbers and label
                  val datatrainfilter=train.filterByRange(1,12)

                  //Filter some features of the sample and select the sample classification label
                  val datatrainlabel=train.filterByRange(13,13).map{case(i,label) =>
                      label.toInt}.collect()

                  //According to the key (labeled 0 ~ n subscript) to merge two RDD
                  val datasub=datatrainfilter.join(datatestfilter).sortByKey()

                  //Subtraction
                  val resultsub=datasub.map(c=>((c._2._1)-(c._2._2)))

                  //Square
                  val resultsquare=resultsub.map(x => x*x)

                  //Sum
                  var dis=resultsquare.reduce((x,y) => x + y)

                  //Stored the calculated Euclidean distance
                  distance += Tuple4(a,c,Math.sqrt(dis),datatrainlabel(0))
           }

           val rdd = sc.parallelize(distance)
           //Sort the distances from small to large
           val knnByDistance = rdd.map{ case(testid,trainid,dist,label) =>
               (dist,label)}.sortBy(-_._1,false)
```

```
            //Select the first K distance samples for classification
            val kdis = knnByDistance.take(kvalue);
            kdis.foreach(println(_))
            println("=================")

            //The number of sample labels that appear most in K distances is counted, and the
                label is used as the prediction label of the test sample
            val giveLabel = sc.parallelize(kdis,5).map{ case(dist,label) =>
                (label,1)}.reduceByKey(_+_).collect.sortBy(-_._2)
            giveLabel.foreach(println(_))

            test_label+=Tuple2(a,giveLabel(0)._1)
    }

    //Output the results to the file
        val label_out = sc.parallelize(test_label)
        label_out.saveAsTextFile("file:///usr/local/data/out")

    //end
    sc.stop()
  }
}
```

7.2 DBSCAN.scala

```
import scala.collection.mutable.ArrayBuffer
import scala.io.Source
import scala.util.control.Breaks._
import java.util.Random
import scala.math.exp
import org.apache.hadoop.conf.Configuration
import org.apache.spark._
import org.apache.spark.scheduler.InputFormatInfo
import scala.math._

object DBSCAN {

  def main(args: Array[String]): Unit = {

    if (args.length < 3) {
        System.err.println("Usage: SimpleApp <filetrain> <minPts> <ePs> <dim>")
        System.exit(1)
    }

    val minPts = args(1).toInt
    println("minPts: " + minPts)
```

```
  val ePs = args(2).toDouble
  println("ePs: " + ePs)

  val dim = args(3).toInt
  println("dim: " + dim)

  val conf = new SparkConf().setAppName("DBSCAN")
  val sc = new SparkContext(conf)
  val trainFile = sc.textFile(args(0), 5);

  val train_count = trainFile.count()
  println("train_count line : " + train_count)

  //Split, extract data
  val traindata=trainFile.map(line => {
      val datas = line.split(",").map(x => x.toDouble).toVector
      (datas)
  })


  traindata.collect.foreach(println(_))
  println("--------------------")
  //p.foreach(println(_))

  //Run dbscan clustering algorithm
  val (cluster,types) = runDBSCAN(points,ePs,minPts)
  ShowResult(points,cluster,types)

}

//compute distance
def vectorDis(v1: Vector[Double], v2: Vector[Double]):Double = {
  var distance = 0.0
  for(i <- 0 to v1.length - 1){
    distance += (v1(i) - v2(i)) * (v1(i) - v2(i))
  }
  distance = math.sqrt(distance)
  distance
}


def runDBSCAN(data:Array[Vector[Double]],ePs:Double,minPts:Int):(Array[Int],Array[Int]) ={
    //Used to distinguish core point 1, boundary point 0, and noise point -1 (that is, the point in
        cluster with a value of 0)
    val types = (for(i <- 0 to data.length - 1) yield -1).toArray

    //Used to determine whether the point has been dealt with or not
    val visited = (for(i <- 0 to data.length - 1) yield 0).toArray

    var xTempPoint = Vector(0.0,0.0)
```

```
var yTempPoint = Vector(0.0,0.0)

//Temporary storage distance
var distance = new Array[(Double,Int)](1)
var distanceTemp = new Array[(Double,Int)](1)

//Temporary storage neighbor
val neighPoints = new ArrayBuffer[Vector[Double]]()
var neighPointsTemp = new Array[Vector[Double]](1)

//Used to mark the category that each data point belongs to clustering results
val cluster = new Array[Int](data.length)

var index = 0

//Cluster mark
var number = 1
for(i <- 0 to data.length - 1)
{
   //If Not processed
   if(visited(i) == 0){
    //Marked as processed
     visited(i) == 1
     xTempPoint = data(i)

     //Get the distance from this point to all other points
     distance = data.map(x => (vectorDis(x,xTempPoint),data.indexOf(x)))

     //Find all points within a radius ePs, which is the set of density connected points
     neighPoints ++= distance.filter(x => x._1 <= ePs).map(v => data(v._2))

     //The number of neighborhood aggregates does not reach the threshold, but it is non-core
         points
     if(neighPoints.length > 1 && neighPoints.length < minPts){
       breakable{
        //If there is a core point in the field, the point is the boundary point
         for(i <- 0 to neighPoints.length -1 ){
           var index = data.indexOf(neighPoints(i))
           if(types(index) == 1){
             types(i) = 0//boundary point
             break
           }
         }
       }
     }

    //If the neighborhood set exceeds the threshold, it is marked as the core point
     if(neighPoints.length >= minPts){
       types(i) = 1
       cluster(i) = number
```

```
            //The core points are searched for the point iterations in the area of the core point
                until the set of points in the radius of all the core point fields no longer expand
            while(neighPoints.isEmpty == false){
              //Take the first point in the neighborhood set
              yTempPoint =neighPoints.head
              index = data.indexOf(yTempPoint)
              if(visited(index) == 0)
              {//If the point is not processed then the marker has been processed
                visited(index) = 1
                if(cluster(index)==0) cluster(index) = number
                distanceTemp = data.map(x => (vectorDis(x,yTempPoint),data.indexOf(x)))
                neighPointsTemp = distanceTemp.filter(x => x._1 <= ePs).map(v => data(v._2))

                if(neighPointsTemp.length >= minPts) {
                 //Put into the same core with the cluster
                  types(index) = 1
                  for (i <- 0 to neighPointsTemp.length - 1) {
                    //Put the unclassified points in the field into clusters
                    if (cluster(data.indexOf(neighPointsTemp(i))) == 0) {
                      cluster(data.indexOf(neighPointsTemp(i))) = number
                      neighPoints += neighPointsTemp(i)
                    }
                  }
                }
                if(neighPointsTemp.length > 1 && neighPointsTemp.length < minPts){
                  breakable{
                   //This is a non-core point, if there is a core point in the field, the point is
                       the boundary point
                    for(i <- 0 to neighPointsTemp.length -1 ){
                      var index1 = data.indexOf(neighPointsTemp(i))
                      if(types(index1) == 1){
                        types(index) = 0//boundary point
                        break
                      }
                    }
                  }
                }
              }
              //Delete the first point in the neighborhood set
              neighPoints-=yTempPoint
            }//end-while
            number += 1 //New cluster mark
          }
        }
      }
    (cluster,types)
  }
}
```