

# 大数据实验处理手册

---

## 概述

---

本次实验内容为：

本手册在ubuntu18平台上验证，相关软件环境为

- Java8
- spark3.0.2
- Scala2.10

## 实验环境搭建

---

## 系统安装

---

下载 ubuntu18.04 镜像（你当然也可以使用别的 linux 版本）

<https://mirrors.aliyun.com/ubuntu-releases/18.04/>

选择

[ubuntu-18.04.6-desktop-amd64.iso](#)

2.3 GB

2021-09-17 05:46

## Java环境配置

---

```
sudo apt-get install openjdk-8-jdk
java -version
```

```
cc@ubuntu:~$ java -version
openjdk version "1.8.0_312"
OpenJDK Runtime Environment (build 1.8.0_312-8u312-b07-0ubuntu1~18.04-b07)
OpenJDK 64-Bit Server VM (build 25.312-b07, mixed mode)
```

出现此提示表示安装完成

## Scala配置

---

本环节使用了第三方工具 SDKMAN 来帮助配置 scala，避免繁琐的环境变量配置。如果你不能顺利完成这一步的操作，可以自行搜索一些 scala 安装教程（如自行下载Scala上传到ubuntu）

首先安装 SDKMAN

```
sudo apt install curl
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk version
```

```
==== BROADCAST =====
* 2021-04-24: jreleaser 0.2.0 available on SDKMAN! https://github.com/jreleaser/jreleaser/releases/tag/v0.2.0
* 2021-04-23: leiningen 2.9.6-1 available on SDKMAN!
* 2021-04-19: groovy 3.0.8 available on SDKMAN!
=====
SDKMAN 5.11.0+644
```

出现此提示表示安装成功

然后用SDKMAN安装Scala

```
sdk install scala
```

最后输入 scala -version



```
Scala code runner version 2.13.5 -- Copyright 2002-2020, LAMP/EPFL and Lightbend, Inc.
```

出现此提示表示安装成功

## Spark&Hadoop下载

下载 spark hadoop 合体安装包、解压：

<https://mirrors.tuna.tsinghua.edu.cn/apache/spark/spark-3.0.3/>

	<a href="#">spark-3.0.3-bin-hadoop2.7.tgz</a>	2021-06-17 13:28	210M
	<a href="#">spark-3.0.3-bin-hadoop3.2.tgz</a>	2021-06-17 13:28	214M

选择spark-3.0.3-bin-hadoop3.2-tgz下载

```
tar -zxvf spark-3.0.3-bin-hadoop3.2.tgz
```

## Python环境

```
sudo apt install python python-pip
```

## 实验部分

---

使用spark来进行高斯混合模型(GMM)聚类算法实验

## 数据准备

---

将数据文件iris.txt存放在/home/cc/文件夹下

(温馨提示：这个cc是我注册的ubuntu用户，如果注册的是别的用户名，就会是别的文件名)

## 模型的训练与分析

---

在spark-shell下运行，在spark对应目录下运行如下命令

```
./bin/spark-shell --master local[4]
```

创建SparkSession对象

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder().
    master("local").
    appName("my App Name").
    getOrCreate()
```

Spark的ML库提供的高斯混合模型都在org.apache.spark.ml.clustering包下，和其他的聚类方法类似，其具体实现分为两个类：用于抽象GMM的超参数并进行训练的GaussianMixture类（Estimator）和训练后的模型GaussianMixtureModel类（Transformer），在使用前，引入需要的包：

```
import org.apache.spark.ml.clustering.
{GaussianMixture,GaussianMixtureModel}
import org.apache.spark.ml.linalg.{Vector,Vectors}
```

```
}
```

开启RDD的隐式转换

```
import spark.implicits._
```

为了便于生成相应的DataFrame，这里定义一个名为model\_instance的case class作为DataFrame每一行（一个数据样本）的数据类型。

```
case class model_instance (features: Vector)
```

在定义数据类型完成后，即可将数据读入RDD[model\_instance]的结构中，并通过RDD的隐式转换.toDF()方法完成RDD到DataFrame的转换：

```
scala>val rawData = sc.textFile("file:///home/cc/iris.txt")

scala>val df = rawData.map(line => { model_instance(
  Vectors.dense(line.split(",").filter(p => p.matches("\\d*
  (\\.?)\\d*")).map(_.toDouble)) )}).toDF()
```

可以通过创建一个GaussianMixture类，设置相应的超参数，并调用fit(..)方法来训练一个GMM模型GaussianMixtureModel，在该方法调用前需要设置一系列超参数，如下表所示：

| 参数 | 含义 |

| ---- | :-----: |

| K | 聚类数目，默认为2 |

| maxIter | 最大迭代次数，默认为100 |

| seed | 随机数种子，默认为随机Long值 |

| Tol | 对数似然函数收敛阈值，默认为0.01 |

其中，每一个超参数均可通过名为setXXX(...)（如maxIterations即为

setMaxIterations()）的方法进行设置。这里，我们建立一个简单的GaussianMixture对象，设定其聚类数目为3，其他参数取默认值。

```
val gm = new GaussianMixture().setK(3).setPredictionCol("Prediction").
  setProbabilityCol("Probability")

val gmm = gm.fit(df)
```

调用transform()方法处理数据集之后，打印数据集，可以看到每一个样本的预测簇以及其概率分布向量

```
scala> val result = gmm.transform(df)
scala> result.show(150, false)
```

```
scala> result.show(150, false)
+-----+-----+-----+
|features|Probability|Prediction|
+-----+-----+-----+
|[5.1,3.5,1.4,0.2]| [0.9999999997843543,4.736667844118139E-17,2.1564571182403457E-10]| 0|
|[4.9,3.0,1.4,0.2]| [0.9999997483193522,8.45154006491501E-15,2.516806392174855E-7]| 0|
|[4.7,3.2,1.3,0.2]| [0.99999933748383,3.7770969537091725E-16,6.625161650737201E-8]| 0|
|[4.6,3.1,1.5,0.2]| [0.9999998801371643,1.7612202657909197E-13,1.1986265955662463E-7]| 0|
|[5.0,3.6,1.4,0.2]| [0.999999999086433,5.785655634622879E-17,9.135659394590451E-11]| 0|
|[5.4,3.9,1.7,0.4]| [0.999999997213767,2.4378414988252134E-16,2.786230699523192E-10]| 0|
|[4.6,3.4,1.4,0.3]| [0.999999818996692,5.14058007781449E-16,1.810033075138817E-7]| 0|
|[5.0,3.4,1.5,0.2]| [0.999999991338874,1.236850823880744E-16,8.661125223536269E-10]| 0|
```

得到模型后，即可查看模型的相关参数，与KMeans方法不同，GMM不直接给出聚类中心，而是给出各个混合成分（多元高斯分布）的参数。在ML的实现中，GMM的每一个混合成分都使用一个MultivariateGaussian类（位于org.apache.spark.ml.stat.distribution包）来存储，我们可以使用GaussianMixtureModel类的weights成员获取到各个混合成分的权重，使用gaussians成员来获取到各个混合成分的参数（均值向量和协方差矩阵）：

```
for (i <- 0 until gmm.getK) {
  | println("Component %d : weight is %f \n mu vector is %s \n sigma
matrix is %s" format
  | (i, gmm.weights(i), gmm.gaussians(i).mean, gmm.gaussians(i).cov))
  | }
}
```

```
scala> for (i <- 0 until gmm.getK) {
  | | println("Component %d : weight is %f \n mu vector is %s \n sigma matrix is %s" format
  | | | (i, gmm.weights(i), gmm.gaussians(i).mean, gmm.gaussians(i).cov))
  | | }
Component 0 : weight is 0.333125
mu vector is [5.006314246719396,3.418694395497936,1.4641012970703788,0.24396417257031788]
sigma matrix is 0.12168087738638096 0.09800182703307668 0.015774267712751044 0.01036003211519594
0.09800182703307668 0.14158808334314135 0.011341215948415424 0.011253882850321861
0.015774267712751044 0.011341215948415424 0.02950559914870197 0.005593016378577845
0.01036003211519594 0.011253882850321861 0.005593016378577845 0.011268742950506895
Component 1 : weight is 0.368098
mu vector is [6.222699937565372,2.954392138152098,5.078065462412692,1.8586222343664884]
sigma matrix is 0.2735199911826566 0.06838687336993228 0.2234592892400264 0.12721084910913963
0.06838687336993228 0.06450577050372237 0.06277964699515355 0.05153126202301464
0.2234592892400264 0.06277964699515355 0.3196993555552372 0.18498314349134487
0.12721084910913963 0.05153126202301464 0.18498314349134487 0.15842761384636647
Component 2 : weight is 0.298776
mu vector is [6.309193326774386,2.7700973458877263,4.691502001149995,1.4500485828397127]
sigma matrix is 0.6315714199651397 0.1950869909423567 0.7490441238499553 0.2337211896310255
0.1950869909423567 0.14659636689304717 0.20018917993217067 0.07227559438773999
0.7490441238499553 0.20018917993217067 1.0383024578677087 0.3261783601627534
0.2337211896310255 0.07227559438773999 0.3261783601627534 0.11255524692994398
```

参考网址<http://dblab.xmu.edu.cn/blog/1456/>