在我的上一篇里我写的那个只是个人对KMeans聚类在这个项目中的一部分，今天花了很长时间写完和完整的运行测试完这个代码，篇幅很长，都是结合写的加上自己完善的异常检测部分，废话不多说，直接代码实战：

```
package internet

import org.apache.spark.mllib.clustering.{KMeansModel, KMeans}
import org.apache.spark.mllib.linalg.{Vectors,Vector}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkContext, SparkConf}

/**
  * Created by 汪本成 on 2016/7/24.
  */
object CheckAll {

  def main(args: Array[String]) {
    //创建入口对象
   val conf = new SparkConf().setAppName("CheckAll").setMaster("local")
    val sc= new SparkContext(conf)
    val HDFS_DATA_PATH = "hdfs://node1:9000/user/spark/sparkLearning/cluster/kddcup.data"
    val rawData = sc.textFile(HDFS_DATA_PATH)

    /** 分类统计样本，降序排序 **/
//    clusteringTake1(rawData)
    /** 评价k值 **/
//    clusteringTake2(rawData)
//    clusteringTake3(rawData)
//    clusteringTake4(rawData)
//    clusteringTake5(rawData)
    /** R数据可视化 **/
    /** 异常检测 **/
    var beg = System.currentTimeMillis()
    anomalies(rawData)
    var end = System.currentTimeMillis()
    println("用时：" + (end - beg) / 1000 + "s")
  }



  //Clustering，Task1
  def clusteringTake1(rawData: RDD[String]) = {
    //分类统计样本个数，降序排序
   rawData.map(_.split(",").last).countByValue().toSeq.sortBy(_._2).reverse.foreach(println)


    val labelsAndData = rawData.map {
      line =>
        //将csv格式的行拆分成列，创建一个buffer，是一个可变列表
       val buffer = line.split(",").toBuffer
        //删除下标从1开始的三个类别型列
       buffer.remove(1, 3)
        //删除下标最后的标号列
       val label = buffer.remove(buffer.length - 1)
        //保留其他值并将其转换成一个数值型(Double型对象)数组
       val vector = Vectors.dense(buffer.map(_.toDouble).toArray)
        //将数组和标号组成一个元祖
       (label, vector)
    }
```

```scala
    /**
      * 为啥要进行labelsAndData => data转化?
      * 1、k均值在运行过程中只用到特征向量(即没有用到数据集的目标标号列)
      * 2、使data这个RDD只包含元祖的只包含元组的第二个元素
     * 3、实现2可以通过元组类型RDD的values属性得到，在放入缓存中，减少落地
     */
    //提取出元组的特征向量
   val data = labelsAndData.values.cache()

    //实例化Kmeans类对象
   val kmeans = new KMeans()
    //建立KMeansModel
    val model = kmeans.run(data)
    //输出每个簇的质心
   model.clusterCenters.foreach(println)

    val clusterLabelCount = labelsAndData.map {
      case (label, datum) =>
        //预测样本datum的分类cluster
        val cluster = model.predict(datum)
        //返回类别-簇的元组
       (cluster, label)
    }.countByValue()

    //对簇-类别对分别进行计数，并以可读方式输出
   clusterLabelCount.toSeq.sorted.foreach {
      case ((cluster, label), count) =>
        println(f"$cluster%1s$label%18s$count%8s")
    }
    data.unpersist()
  }

  /**
    * 欧氏距离公式
   * a.toArray.zip(b.toArray)对应 "两个向量相应元素"
    * map(p => p._1 - p._2)对应 "差"
    * map(d => d*d).sum对应 "平方和"
    * math.sqrt()对应 "平方根"
    * @param a
    * @param b
    * @return
    */
  def distance(a: Vector, b: Vector) =
    math.sqrt(a.toArray.zip(b.toArray).map(p => p._1 - p._2).map(d => d * d).sum)

  /**
    * 欧氏距离公式应用到model中
   * KMeansModel.predict方法中调用了KMeans对象的findCloest方法
   * @param datum
    * @param model
    * @return
    */
  def distToCenter(datum: Vector, model: KMeansModel) = {
    //预测样本datum的分类cluster
    val cluster = model.predict(datum)
    //计算质心
   val center = model.clusterCenters(cluster)
    //应用距离公式
```

```scala
  distance(center, datum)
}

/**
  * 平均质心距离
 * @param data
  * @param k
  * @return
  */
def clusteringScore(data: RDD[Vector], k: Int): Double = {
  val kmeans = new KMeans()
  //设置k值
 kmeans.setK(k)
  //建立KMeansModel
  val model = kmeans.run(data)
  //计算k值model平均质心距离，mean()是平均函数
 data.map(datum => distToCenter(datum, model)).mean()
}

/**
  * 平均质心距离优化
 * @param data
  * @param k
  * @param run   运行次数
 * @param epsilon  阈值
 * @return
  */
def clusteringScore2(data: RDD[Vector], k: Int, run: Int, epsilon: Double): Double = {
  val kmeans = new KMeans()
  kmeans.setK(k)
  //设置k的运行次数
 kmeans.setRuns(run)
  //设置阈值
 kmeans.setEpsilon(epsilon)
  val model = kmeans.run(data)
  data.map(datum => distToCenter(datum, model)).mean()
}

//Clustering，Take2
def clusteringTake2(rawData: RDD[String]): Unit ={
  val data = rawData.map {
    line =>
      val buffer = line.split(",").toBuffer
      buffer.remove(1, 3)
      buffer.remove(buffer.length - 1)
      Vectors.dense(buffer.map(_.toDouble).toArray)
  }.cache()

  val run = 10
  val epsilon = 1.0e-4
  //在(5,30)区间内以5为等差数列数值不同k值对其评分
 (5 to 30 by 5).map(k => (k, clusteringScore(data, k))).foreach(println)
  //在(20,120)区间内以10为等差数列数值不同k值对其评分
 (30 to 100 by 10).par.map(k => (k, clusteringScore2(data, k, run, epsilon))).foreach(println)

  data.unpersist()
}
```

```scala
/**
  * 加工出R可视化数据存入HDFS中
 * @param rawData
  * @param k
  * @param run
  * @param epsilon
  */
def visualizationInR(rawData: RDD[String], k: Int, run: Int, epsilon: Double): Unit ={
  val data = rawData.map {
    line =>
      val buffer = line.split(",").toBuffer
      buffer.remove(1, 3)
      buffer.remove(buffer.length - 1)
      Vectors.dense(buffer.map(_.toDouble).toArray)
  }.cache()

  val kmeans = new KMeans()
  kmeans.setK(k)
  kmeans.setRuns(run)
  kmeans.setEpsilon(epsilon)
  val model = kmeans.run(data)

  val sample = data.map(
    datum =>
      model.predict(datum) + "," + datum.toArray.mkString(",")
  ).sample(false, 0.05)   //选择了5%行

  sample.saveAsTextFile("hdfs://nodel:9000/user/spark/R/sample")
  data.unpersist()
}

/**
  *
  * @param data
  * @return
  */
def buildNormalizationFunction(data: RDD[Vector]): (Vector => Vector) = {
  //将数组缓冲为Array
  val dataAsArray = data.map(_.toArray)
  //数据集第一个元素的长度
  val numCols = dataAsArray.first().length
  //返回数据集的元素个数
  val n = dataAsArray.count()
  //两个数组对应元素相加求和
  val sums = dataAsArray.reduce((a, b) => a.zip(b).map(t => t._1 + t._2))
  //将RDD聚合后进行求平方和操作
  val sumSquares = dataAsArray.aggregate(new Array[Double](numCols))(
    (a, b) => a.zip(b).map(t => t._1 + t._2 * t._2),
    (a, b) => a.zip(b).map(t => t._1 + t._2)
  )

  /** zip函数将传进来的两个参数中相应位置上的元素组成一个pair数组。
    * 如果其中一个参数元素比较长，那么多余的参数会被删掉。
    * 个人理解就是让两个数组里面的元素一一对应进行某些操作
    */
  val stdevs = sumSquares.zip(sums).map {
    case (sumSq, sum) => math.sqrt(n * sumSq - sum * sum) / n
```

```scala
  }
  val means = sums.map(_ / n)

  (datum : Vector) => {
    val normalizedArray = (datum.toArray, means, stdevs).zipped.map(
      (value, mean, stdev) =>
        if(stdev <= 0) (value- mean) else (value - mean) /stdev
    )
    Vectors.dense(normalizedArray)
  }
}


//clustering，Task3
def clusteringTake3(rawData: RDD[String]): Unit ={
  val data = rawData.map { line =>
    val buffer = line.split(',').toBuffer
    buffer.remove(1, 3)
    buffer.remove(buffer.length - 1)
    Vectors.dense(buffer.map(_.toDouble).toArray)
  }

  val run = 10
  val epsilon = 1.0e-4

  val normalizedData = data.map(buildNormalizationFunction(data)).cache()

  (60 to 120 by 10).par.map(
    k => (k, clusteringScore2(normalizedData, k, run, epsilon))
  ).toList.foreach(println)

  normalizedData.unpersist()
}

/**
  * 基于one-hot编码实现类别型变量替换逻辑
  * @param rawData
  * @return
  */
def buildCategoricalAndLabelFunction(rawData: RDD[String]): (String => (String, Vector))  = {
  val splitData = rawData.map(_.split(","))
  //建立三个特征
  val protocols = splitData.map(_(1)).distinct().collect().zipWithIndex.toMap   //特征值是1，0，0
  val services = splitData.map(_(2)).distinct().collect().zipWithIndex.toMap    //特征值是0，1，0
  val tcpStates = splitData.map(_(3)).distinct().collect().zipWithIndex.toMap   //特征值是0，0，1
  //
  (line: String) => {
    val buffer = line.split(",").toBuffer
    val protocol = buffer.remove(1)
    val service = buffer.remove(1)
    val tcpState = buffer.remove(1)
    val label = buffer.remove(buffer.length - 1)
    val vector = buffer.map(_.toDouble)

    val newProtocolFeatures = new Array[Double](protocols.size)
    newProtocolFeatures(protocols(protocol)) = 1.0
    val newServiceFeatures = new Array[Double](services.size)
    newServiceFeatures(services(service)) = 1.0
```

```scala
    val newTcpStateFeatures = new Array[Double](tcpStates.size)
    newTcpStateFeatures(tcpStates(tcpState)) = 1.0

    vector.insertAll(1, newTcpStateFeatures)
    vector.insertAll(1, newServiceFeatures)
    vector.insertAll(1, newProtocolFeatures)

    (label, Vectors.dense(vector.toArray))
  }
}


//Clustering，Task4
def clusteringTake4(rawData: RDD[String]): Unit ={
  val paraseFunction = buildCategoricalAndLabelFunction(rawData)
  val data = rawData.map(paraseFunction).values
  val normalizedData = data.map(buildNormalizationFunction(data)).cache()

  val run = 10
  val epsilon = 1.0e-4

  (80 to 160 by 10).map(
    k=> (k, clusteringScore2(normalizedData, k, run, epsilon))
  ).toList.foreach(println)

  normalizedData.unpersist()
}


//Clustering, Task5
/**
  * 对各个簇的熵加权平均，将结果作为聚类得分
 * @param counts
  * @return
  */
def entropy(counts: Iterable[Int]) = {
  val values = counts.filter(_ > 0)
  val n: Double = values.sum
  values.map {
    v =>
      val p = v / n
      -p * math.log(p)
  }.sum
}

/**
  * 计算熵的加权平均
 * @param normalizedLabelsAndData
  * @param k
  * @param run
  * @param epsilon
  * @return
  */
def clusteringScore3(normalizedLabelsAndData: RDD[(String, Vector)], k: Int, run: Int, epsilon: Double) =
  val kmeans = new KMeans()
  kmeans.setK(k)
  kmeans.setRuns(run)
  kmeans.setEpsilon(epsilon)
```

```
    //建立KMeansModel
    val model = kmeans.run(normalizedLabelsAndData.values)
    //对每个数据集预测簇类别
   val labelAndClusters = normalizedLabelsAndData.mapValues(model.predict)
    //将RDD[(String, Vector)]  => RDD[(String, Vector)],即swap Keys / Values，对换键和值
   val clustersAndLabels = labelAndClusters.map(_.swap)
    //按簇提取标号集合
   val labelsInCluster = clustersAndLabels.groupByKey().values
    //计算所有集合中有多少标签(label)，即标号的出现次数
   val labelCounts = labelsInCluster.map(_.groupBy(l => l).map(_._2.size))
    //通过类别大小来反映平均信息量，即熵
   val n = normalizedLabelsAndData.count()
    //根据簇大小计算熵的加权平均
   labelCounts.map(m => m.sum * entropy(m)).sum() / n
  }


  def clusteringTake5(rawData: RDD[String]): Unit ={
    val parseFunction = buildCategoricalAndLabelFunction(rawData)
    val labelAndData = rawData.map(parseFunction)
    val normalizedLabelsAndData = labelAndData.mapValues(buildNormalizationFunction(labelAndData.values)).ca
()

    val run = 10
    val epsilon = 1.0e-4

    (80 to 160 by 10).map(
      k => (k, clusteringScore3(normalizedLabelsAndData, k, run, epsilon))
    ).toList.foreach(println)

    normalizedLabelsAndData.unpersist()
  }


  //Detect anomalies(发现异常)
  def bulidAnomalyDetector(data: RDD[Vector], normalizeFunction: (Vector => Vector)): (Vector => Boolean) =
    val normalizedData = data.map(normalizeFunction)
    normalizedData.cache()

    val kmeans = new KMeans()
    kmeans.setK(150)
    kmeans.setRuns(10)
    kmeans.setEpsilon(1.0e-6)
    val model = kmeans.run(normalizedData)

    normalizedData.unpersist()

    //度量新数据点到最近簇质心的距离
   val distances = normalizedData.map(datum => distToCenter(datum, model))
    //设置阀值为已知数据中离中心点最远的第100个点到中心的距离
   val threshold = distances.top(100).last

    //检测，若超过该阀值就为异常点
   (datum: Vector) => distToCenter(normalizeFunction(datum), model) > threshold
  }

  /**
```

```
    * 异常检测
    * @param rawData
    */
  def anomalies(rawData: RDD[String]) = {
    val parseFunction = buildCategoricalAndLabelFunction(rawData)
    val originalAndData = rawData.map(line => (line, parseFunction(line)._2))
    val data = originalAndData.values
    val normalizeFunction = buildNormalizationFunction(data)
    val anomalyDetector = bulidAnomalyDetector(data, normalizeFunction)
    val anomalies = originalAndData.filter {
      case (original, datum) => anomalyDetector(datum)
    }.keys
    //取10个异常点打印出来
    anomalies.take(10).foreach(println)
  }
}
```

写的有点杂，但是全部自己封装好了。运行起来也没问题，仅供大家参考学习，多多关注下我写的注释就好。
累死我了，或许是我电脑不行缘故，计算这1G数据花了这么长时间，现在我把异常检测部分运行结果给大家看看好了

```
16/07/24 22:48:18 INFO Executor: Running task 0.0 in stage 65.0 (TID 385)
16/07/24 22:48:18 INFO HadoopRDD: Input split: hdfs://node1:9000/user/spark/sparkLearning/cluster/kddcup.data:0+134217728
16/07/24 22:48:30 INFO Executor: Finished task 0.0 in stage 65.0 (TID 385). 3611 bytes result sent to driver
16/07/24 22:48:30 INFO TaskSetManager: Finished task 0.0 in stage 65.0 (TID 385) in 11049 ms on localhost (1/1)
9,tcp,telnet,SF,307,2374,0,0,1,0,0,1,0,1,0,1,3,1,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,69,4,0.03,0.04,0.01,0.75,0.00,0.00,0.00,0.00,normal.
16/07/24 22:48:30 INFO TaskSchedulerImpl: Removed TaskSet 65.0, whose tasks have all completed, from pool
16/07/24 22:48:30 INFO DAGScheduler: ResultStage 65 (take at CheckAll.scala:413) finished in 11.049 s
16/07/24 22:48:30 INFO DAGScheduler: Job 41 finished: take at CheckAll.scala:413, took 11.052917 s
0,tcp,http,S1,299,26280,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,15,16,0.07,0.06,0.00,0.00,1.00,0.00,0.12,231,255,1.00,0.00,0.00,0.01,0.01,0.01,0.00,0.00,nor
0,tcp,telnet,S1,2895,14208,0,0,0,0,0,1,0,0,0,0,13,0,0,0,0,0,1,1,1.00,1.00,0.00,0.00,1.00,0.00,0.00,21,2,0.10,0.10,0.05,0.00,0.05,0.50,0.00,0.00,norm
23,tcp,telnet,SF,104,276,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,2,1.00,0.00,1.00,1.00,0.00,0.00,0.00,0.00,guess_pass
13,tcp,telnet,SF,246,11938,0,0,0,0,4,1,0,0,0,0,2,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,89,2,0.02,0.04,0.01,0.00,0.00,0.00,0.00,0.00,normal
12249,tcp,telnet,SF,3043,44466,0,0,0,1,0,1,13,1,0,0,12,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,61,8,0.13,0.05,0.02,0.00,0.00,0.00,0.00,0.00
60,tcp,telnet,S3,125,179,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1.00,1.00,0.00,0.00,1.00,0.00,0.00,1,1,1.00,0.00,1.00,0.00,1.00,1.00,0.00,0.00,guess_pass
60,tcp,telnet,S3,126,179,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,2,2,0.50,0.50,0.50,0.50,1.00,0.00,0.00,23,23,1.00,0.00,0.04,0.00,0.09,0.09,0.91,0.91,guess_p
583,tcp,telnet,SF,848,25323,0,0,0,1,0,1,107,1,1,100,1,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,1,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,nc
11447,tcp,telnet,SF,3131,45415,0,0,0,1,0,1,0,1,0,0,15,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,100,10,0.09,0.72,0.01,0.20,0.01,0.10,0.69,0.2
用时：4602s
16/07/24 22:48:30 INFO SparkContext: Invoking stop() from shutdown hook
16/07/24 22:48:30 INFO SparkUI: Stopped Spark web UI at http://192.168.1.102:4040
16/07/24 22:48:30 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
16/07/24 22:48:30 INFO MemoryStore: MemoryStore cleared
16/07/24 22:48:30 INFO BlockManager: BlockManager stopped
16/07/24 22:48:30 INFO BlockManagerMaster: BlockManagerMaster stopped
16/07/24 22:48:30 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/07/24 22:48:30 INFO SparkContext: Successfully stopped SparkContext
16/07/24 22:48:30 INFO ShutdownHookManager: Shutdown hook called
16/07/24 22:48:30 INFO ShutdownHookManager: Deleting directory C:\Users\Administrator\AppData\Local\Temp\spark-1ab0ec11-672d-47
9ae8-2050f44a5f91
16/07/24 22:48:30 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
16/07/24 22:48:30 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote trans

Process finished with exit code 0
```