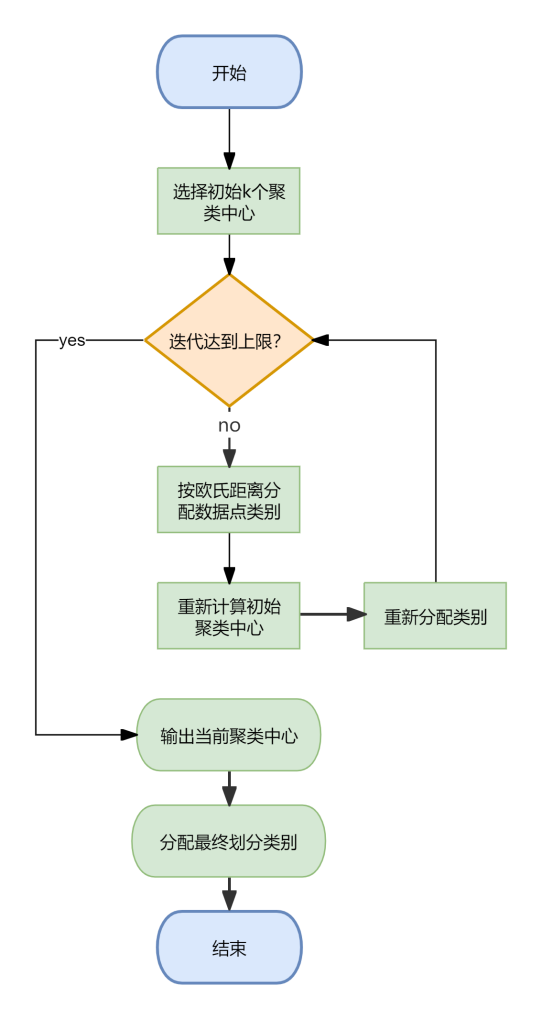
1. 项目详细说明
2. kmeans算法
3. 算法模型
4. means选择K个点作为聚类中心，根据与聚类中心距离最近原则将其他点划分到最邻近类簇，迭代更新各聚类中心，直至达到迭代终止条件：
5. 需要随机选择K个数据点作为初始的聚类中心。
6. 将每个数据点按欧氏距离分配给最近的聚类中心。
7. 对得到的聚类重新计算每个聚类的中心，新的聚类中心是该聚类内所有数据点的均值。
8. 迭代重复执行分配和更新步骤，直到迭代次数达到上限
9. 输出聚类中心，按最近邻原则分配划分类别。



1. 集成流程

为了适配oge平台所用的spark开发框架，聚类算子接收弹性分布式数据集RDD结构的n维特征点作为数据输入，算子集成后处理流程为：

1. 前端python界面调用Cluster.kmeans函数，经解析传入参数：sc(SparkContext):用于算法运行的Spark任务实例；feature:聚类点特征集JSON字符串；center\_num聚类中心数量（即k），max\_itereation最大迭代次数。
2. Scala算子解析JSON字符串，提取Feature数据并构建数据的RDD表示。

val X = jsonTo2DArray(feature)

val X\_RDD = sc.parallelize(X, 4)

val TrainingData = X\_RDD.map(line => Vectors.dense(line))

1. 将Feature部分和其他参数送入kmeans算法执行函数中，调用Apache官方库 org.apache.spark.mllib.clustering.KMeans执行聚类，输出特征点的聚类类别向量。

val model: KMeansModel = KMeans.train(TrainingData, center\_num, max\_iteration)

   model.clusterCenters.foreach(

     center => {

       println("Clustering Center:" + center)

     }

   )

   val labelPred = model.predict(TrainingData).map(\_.toDouble)

1. 重新结合特征向量与预测标签向量，通过转换函数转换成(Feature；Label)型的JSON字符串返回。

// 将原始特征与预测标签结合起来, 并将结果转换为Data

   val resultRDD = X\_RDD.zip(labelPred).map{ **case** (features, label) =>

      (features, label)

    }

    val resultArray = resultRDD.collect()

    val featuresWithLabel = resultArray.map { **case** (features, label) =>

      features :+ label

    }

    val labels = featuresWithLabel.map(\_.last)

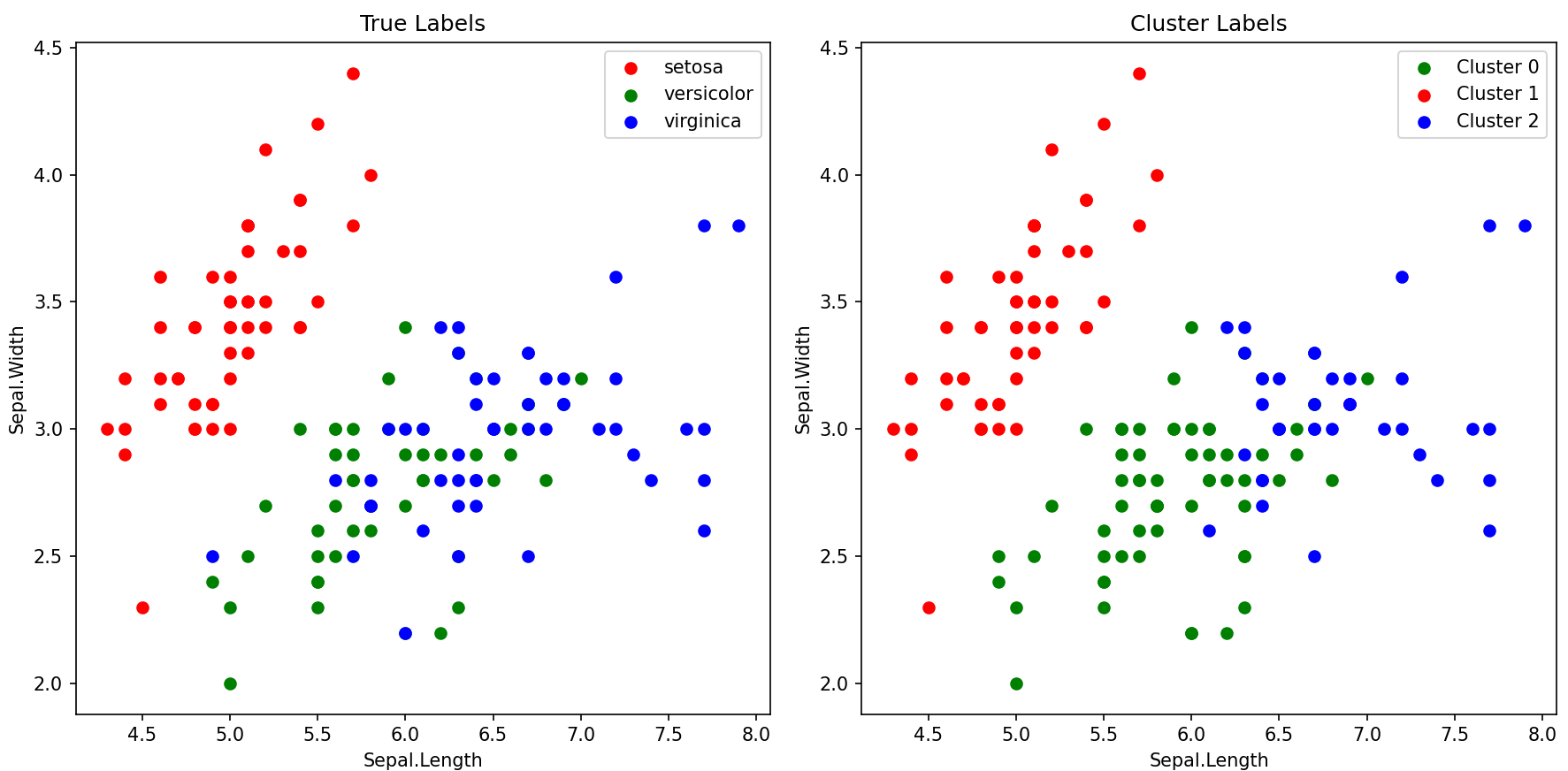
    val featuresFinal = featuresWithLabel.map(\_.dropRight(1))

    val resultData = Data(featuresFinal, labels)

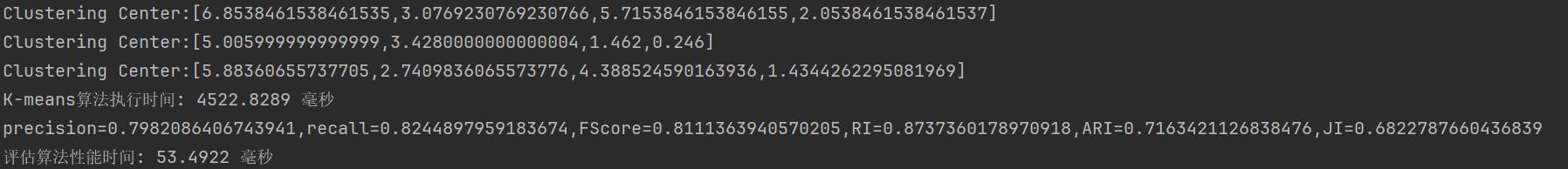
    return DataTojson(resultData)

1. 示例成果

用iris数据集在kmeans上进行聚类结果可视化如下：



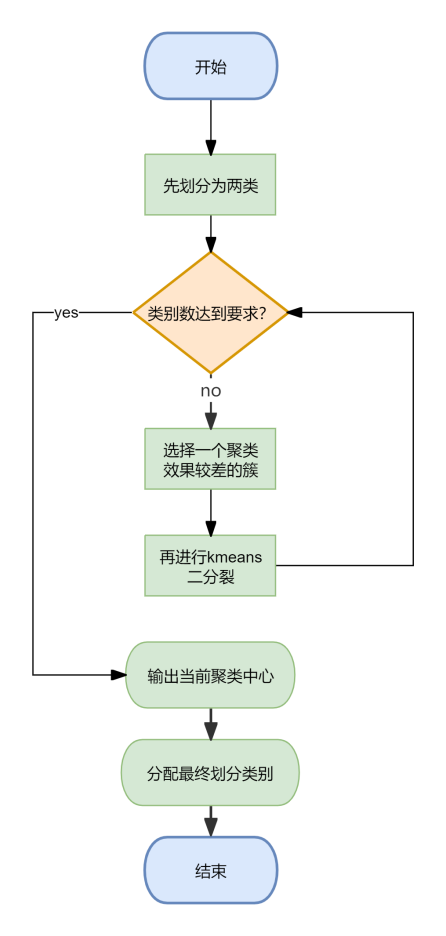
聚类中心与常见聚类指标评估结果如下：



1. Bisecting-kmeans算法
2. 算法模型

Bisecting-kmeans算法属于层次聚类算法之一，层次聚类根据数据之间的距离关系构建层次聚类树，二分kmeans通过自上而下的分裂方式进行层次聚类操作。

1. 先设定k=2，用kmeans算法进行二分类。
2. 综合考虑簇的元素个数以及聚类代价（误差平方和）选择一个聚类效果较差的簇。
3. 对较差的簇再进行kmeans二分裂。
4. 迭代进行选择和分裂过程，直到聚类中心个数满足要求。



1. 集成流程

为了适配oge平台所用的spark开发框架，聚类算子接收弹性分布式数据集RDD结构的n维特征点作为数据输入，算子集成后处理流程为：

1. 前端python界面调用Cluster.hierarchical函数，经解析传入参数：sc(SparkContext):用于算法运行的Spark任务实例；feature:聚类点特征集JSON字符串；center\_num聚类中心数量（即k），max\_itereation最大迭代次数。
2. Scala算子解析JSON字符串，提取Feature数据并构建数据的RDD表示。

val X = jsonTo2DArray(feature)

val X\_RDD = sc.parallelize(X, 4)

val TrainingData = X\_RDD.map(line => Vectors.dense(line))

1. 将Feature部分和其他参数送入Bisecting-kmeans算法执行函数中，调用Apache官方库 org.apache.spark.mllib.clustering.BisectingKMeans执行聚类，输出特征点的聚类类别向量。

val bkm = **new** BisectingKMeans()

     .setK(center\_num) // 期望的簇数目

      .setMaxIterations(max\_iteration) // 最大迭代次数

      .setSeed(12345L)

    // 实例化层次聚类器

    val model = bkm.run(TrainingData) // 使用训练集训练聚类模型

    model.clusterCenters.foreach( // 输出聚类中心

      center => {

        println("Clustering Center:" + center)

      })

    val labelPred = model.predict(TrainingData).map(\_.toDouble)

1. 重新结合特征向量与预测标签向量，通过转换函数转换成(Feature；Label)型的JSON字符串返回。

// 将原始特征和预测标签结合起来

    val resultRDD = X\_RDD.zip(labelPred).map { **case** (features, label) =>

      (features, label)

   }

    // 将结果转换为Data

    val resultArray = resultRDD.collect()

    val featuresWithLabel = resultArray.map { **case** (features, label) =>

      features :+ label // 将预测标签添加到特征中

    }

    val labels = featuresWithLabel.map(\_.last) // 提取预测标签

    val featuresFinal = featuresWithLabel.map(\_.dropRight(1)) // 提取特征

    // 创建Data对象

    val resultData = Data(featuresFinal, labels)

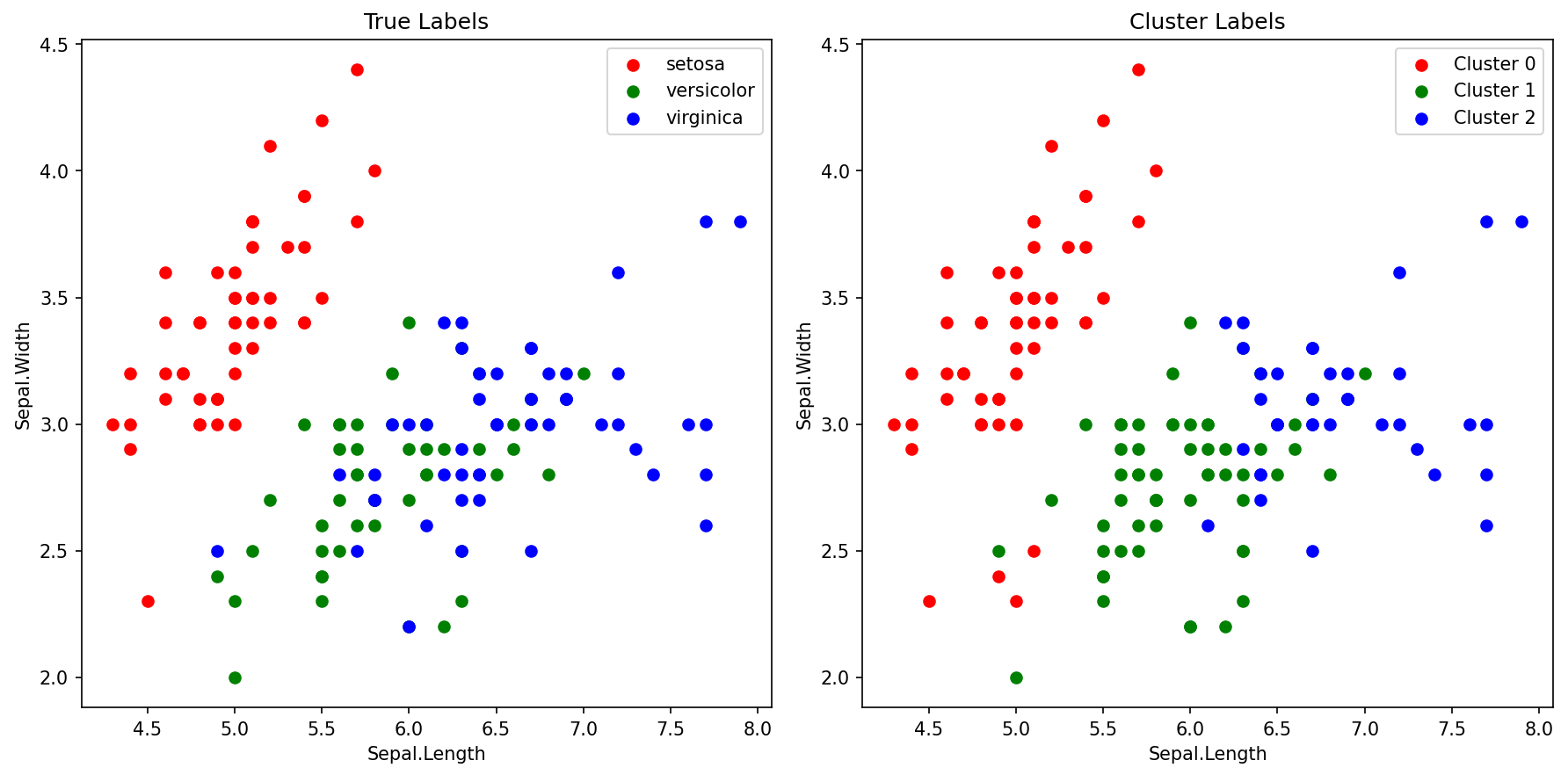
    //evaluation(Y\_RDD.map(\_.toInt).collect(), labelPred.map(\_.toInt).collect()) // 评估算法性能

    // 将标签数组转换为JSON格式

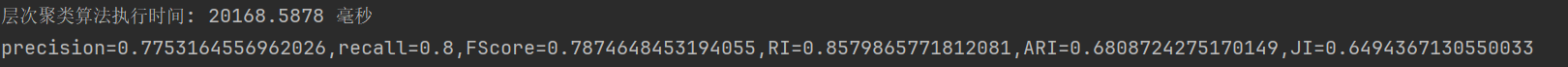
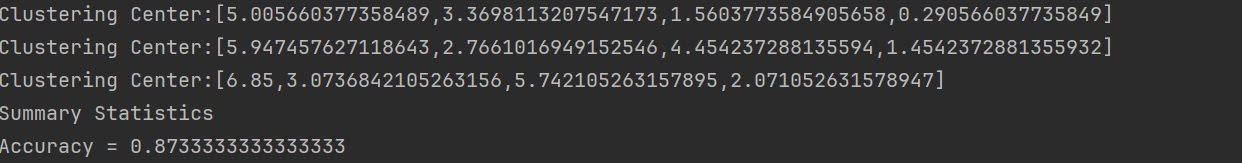
**return** DataTojson(resultData)

1. 示例成果

用iris数据集在Bisecting-kmeans上进行聚类结果可视化如下：



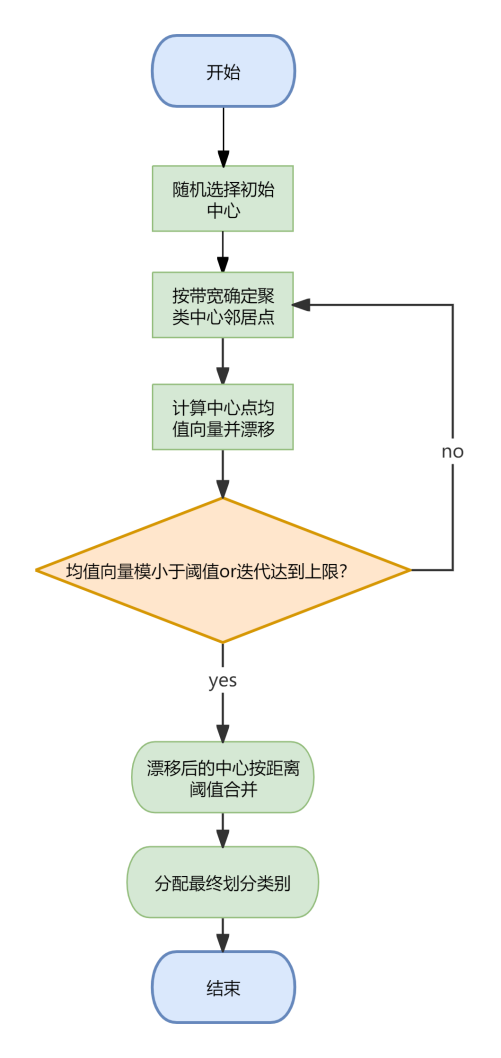
聚类中心与常见聚类指标评估结果如下：



1. Meanshift算法
2. 算法模型

Meanshift均值漂移聚类算法的思想是假设不同簇类的数据集符合不同的概率密度分布，找到任一样本点密度增大的最快方向，将样本朝着该方向不断移动，最终会在局部密度最大处达到收敛，收敛到相同局部密度最大处的点隶属于同一簇类。

1. 随机选择若干样本点作为初始中心
2. 对每个中心，按带宽（bandwidth）参数确定的范围确定中心附近的邻居点，根据均值向量公式采用高斯核函数计算每个中心此时的均值向量
3. 对中心按均值向量进行漂移。
4. 重复2和3操作，直到所有中心的均值向量小于某一阈值（移动距离足够小）或算法迭代次数达到阈值。
5. 对漂移后的中心按距离阈值合并，得到漂移后的中心，按最近邻原则对所有数据点分配类别。



1. 集成流程

为了适配oge平台所用的spark开发框架，聚类算子接收弹性分布式数据集RDD结构的n维特征点作为数据输入，算子集成后处理流程为：

1. 前端python界面调用Cluster.meanshift函数，经解析传入参数：sc(SparkContext):用于算法运行的Spark任务实例；feature:聚类点特征集JSON字符串；bandwidth:带宽，max\_itereation最大迭代次数。
2. Scala算子解析JSON字符串，提取Feature数据并构建数据的RDD表示。

val X = jsonTo2DArray(feature)

val X\_RDD = sc.parallelize(X, 4)

val TrainingData = X\_RDD.map(line => Vectors.dense(line))

1. 将Feature部分和其他参数送入meanshift算法执行函数中，默认选择所有数据点作为初始中心，迭代执行算法流程，达到收敛条件/迭代次数达到阈值时停止，获取所有聚类中心。

breakable {

**for** (iter <- 0 until maxIterations) {

        val bcCenters = data.context.broadcast(currentCenters)

        val newCenters = data.mapPartitions { points =>

          val centers = bcCenters.value

          points.map { point =>

            val neighbors = centers.filter(center => Vectors.sqdist(point, center) < bandwidth \* bandwidth)

            val weights = neighbors.map(neighbor => gaussianKernel(point, neighbor, bandwidth))

            val weightedSum = neighbors.zip(weights).map { **case** (neighbor, weight) =>

              Vectors.dense(neighbor.toArray.map(x => x \* weight))

            }.reduce((a, b) => Vectors.dense(a.toArray.zip(b.toArray).map { **case** (x, y) => x + y }))

            Vectors.dense(weightedSum.toArray.map(\_ / weights.sum))

          }

        }.collect()

        // 检查收敛条件

        var converged = **true**

        breakable {

**for** (i <- currentCenters.indices) {

**if** (Vectors.sqdist(currentCenters(i), newCenters(i)) > convergenceThreshold) {

              converged = **false**

**break**

            }

          }

        }

**if** (converged) {

          println(s"Converged after ${iter + 1} iterations")

          currentCenters = newCenters

**break**

        }

**if** (iter == maxIterations - 1) {

          println(s"Reached max iterations ($maxIterations)")

        }

        currentCenters = newCenters

      }

    }

    currentCenters

1. 合并距离小于阈值的中心，分配特征点类别，重新结合特征向量与预测标签向量，通过转换函数转换成(Feature；Label)型的JSON字符串返回。

// 合并相近的聚类中心

    val mergedClusters = mergeClusters(clusters, bandwidth)

  mergedClusters.foreach {

    center => {

      println("Clustering Center:" + center)

    }

  }

    val labelPred = TrainingData.map { point =>

      mergedClusters.minBy(center => Vectors.sqdist(point, center))

    }.collect()

    val LabelPred = sparkSession.sparkContext.parallelize(labelPred.map(x => labelPred.distinct.indexOf(x)).map(\_.toDouble))

    // 将原始特征和预测标签结合起来

    val resultRDD = X\_RDD.zip(LabelPred).map { **case** (features, label) =>

      (features, label)

    }

    // 将结果转换为Data

    val resultArray = resultRDD.collect()

    val featuresWithLabel = resultArray.map { **case** (features, label) =>

      features :+ label // 将预测标签添加到特征中

    }

    val labels = featuresWithLabel.map(\_.last) // 提取预测标签

    val featuresFinal = featuresWithLabel.map(\_.dropRight(1)) // 提取特征

    // 创建Data对象

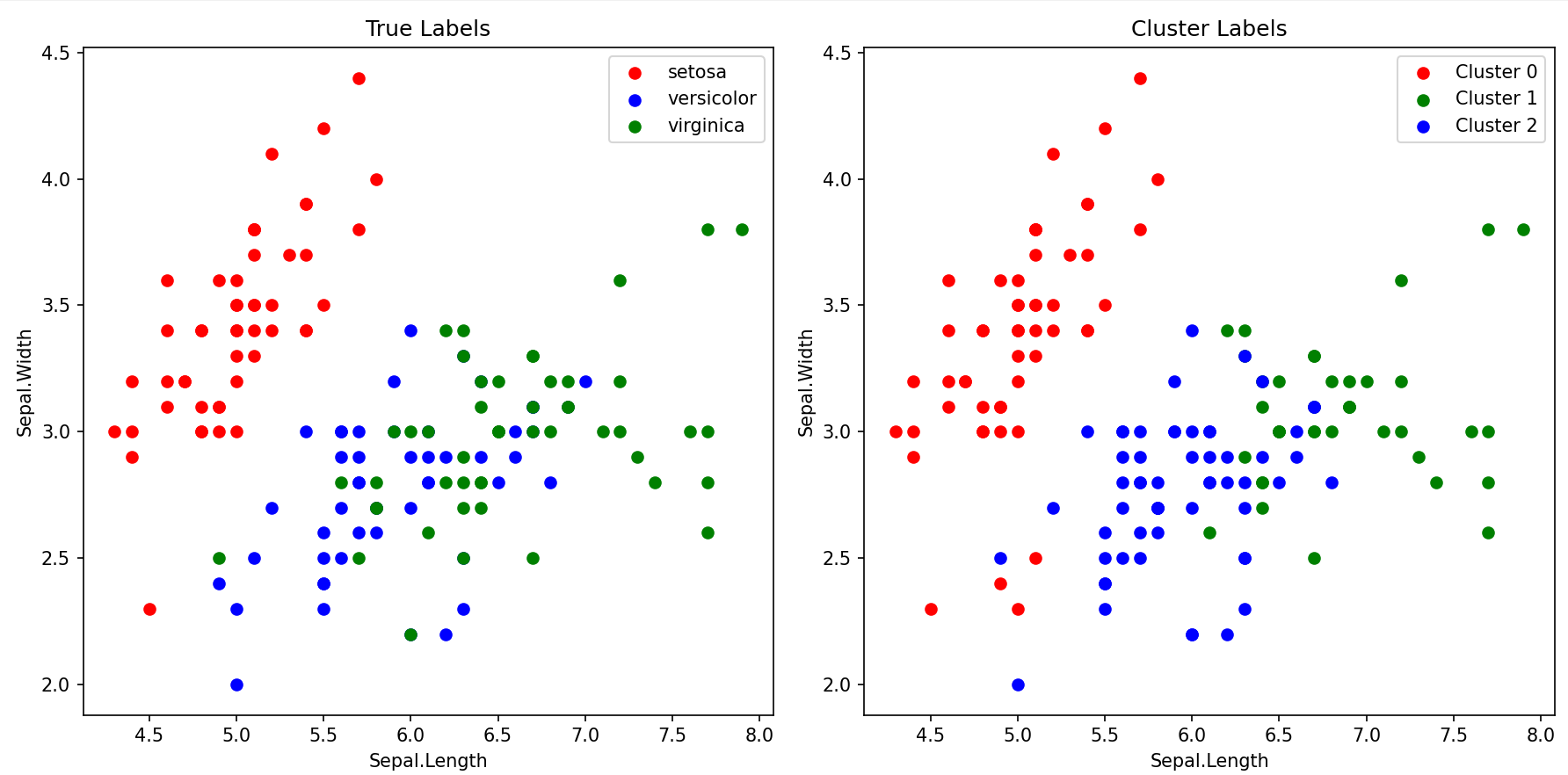
    val resultData = Data(featuresFinal, labels)

    // 将标签数组转换为JSON格式

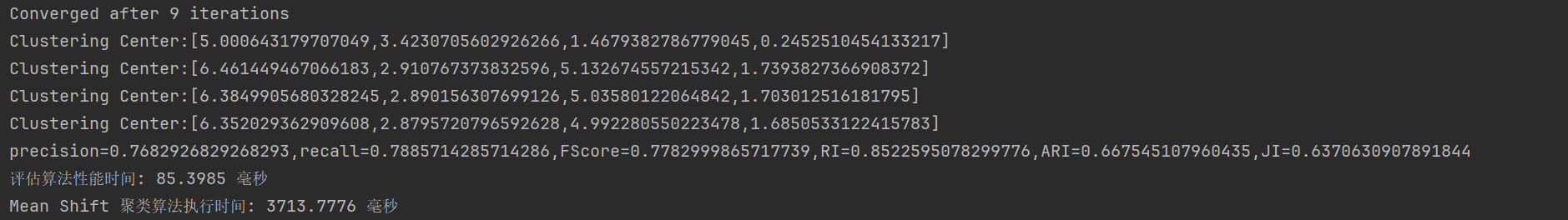
    DataTojson(resultData)

1. 示例成果

用iris数据集在meanshift上进行聚类(控制聚类数为3)结果可视化如下：



当bandwidth=1.35时聚类中心与常见聚类指标评估结果如下：



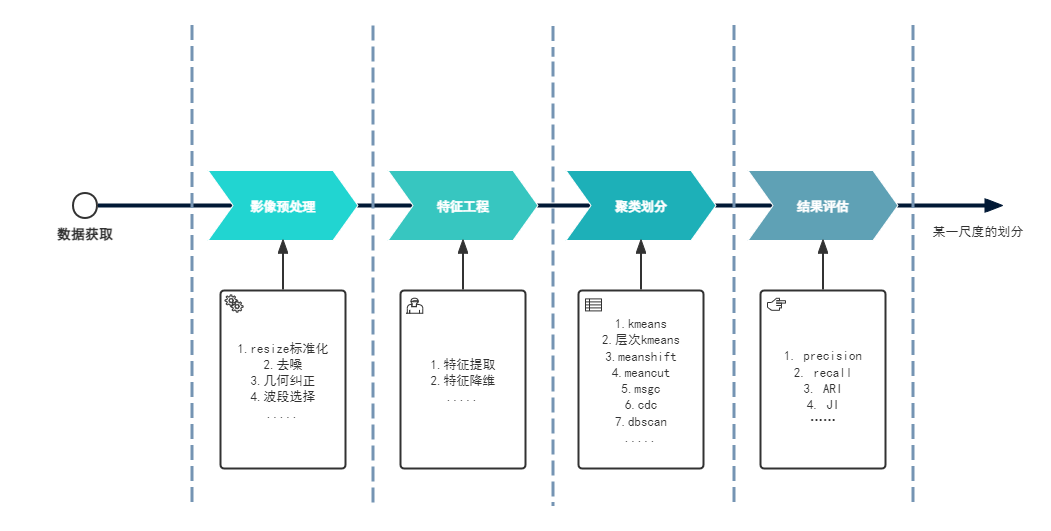
1. 项目应用说明

应用案例：影像非监督分类：

1. 背景

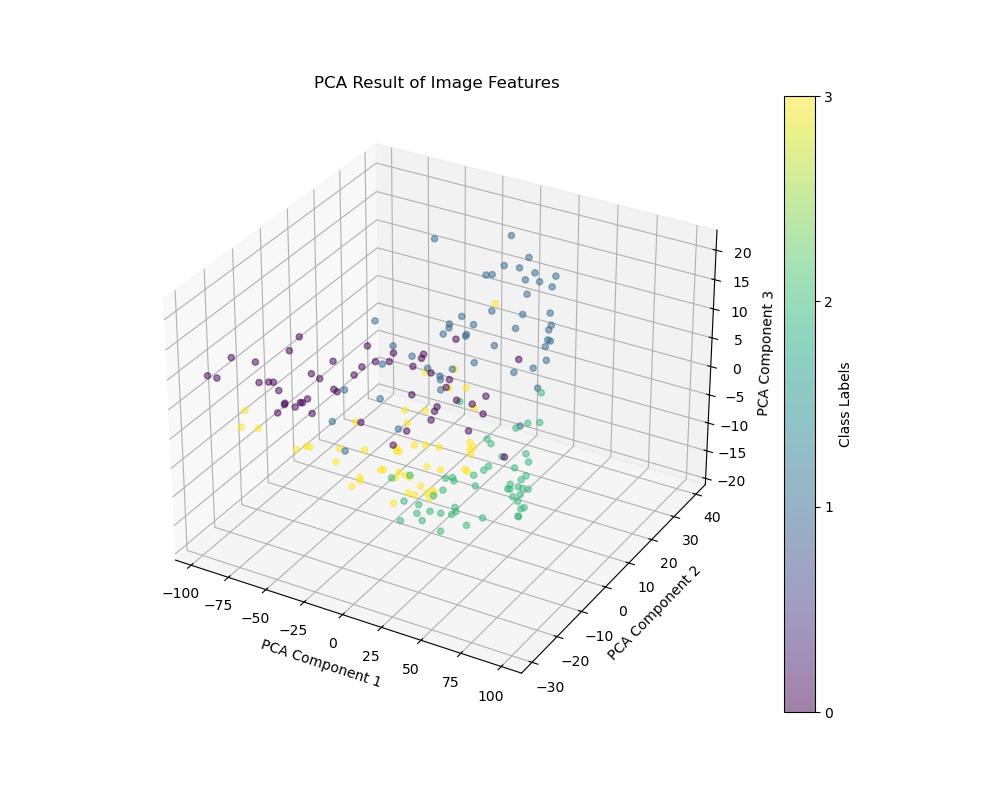
随着遥感摄影测量技术的不断发展和信息通信交互的速度的提升，遥感影像数据的数据类型和数据量正在以非常迅猛的速度增加，大量无标签影像的组织和管理成为亟待解决的问题。聚类作为经典的非监督分类算法能够通过特征空间的相似性划分类簇，给研究者进一步确定准确类别提供参照。本案例旨在比较特定需求下七种聚类算法和本组降维算法的效果和适用情况。

1. 技术路线

由于聚类划分极大的依赖于特征工程的优劣，为了有效定量验证本组聚类算子的效果，利用在ImageNet上预训练的ResNet50神经网络进行特征提取。考虑划分尺度一致性，将数据集按照3：1划分训练与测试集，用训练集微调模型再应用于测试集上，使得提特征描述类别的尺度与给定类别标签一致。提取特征后采用传统PCA和本组可伸缩流形学习算法scML对提取的特征进行特征降维，之后应用本组集成的聚类算法进行非监督聚类划分，通过四个常见的聚类评价指标比较不同降维方法和聚类算法的优劣。工作流程如下：

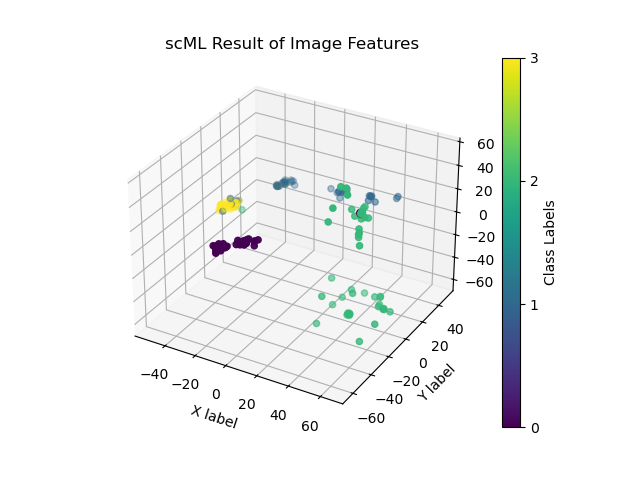
1. 结果分析
2. 降维效果比对

同时降维到3维便于可视化展示，PCA算法降维效果如下：



降维后的数据呈现明显的方向聚集性，类别间有较为明确的边界，但整体特征点分布较为松散。

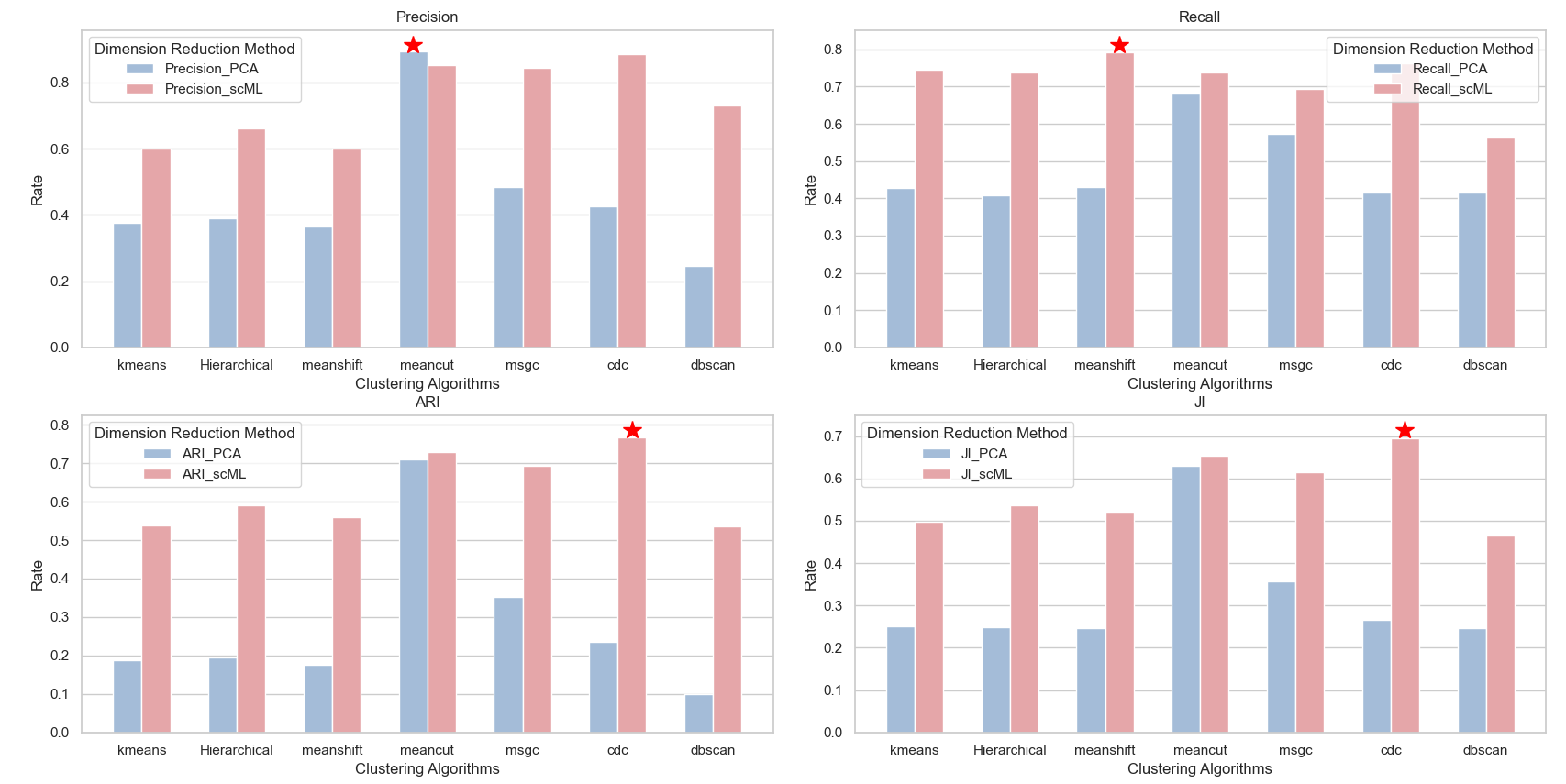
scML算法降维效果如下：



该算法降维效果较PCA更为聚集，在保留方向性特征条件下分离性更好。

1. 聚类效果比对

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PCA |  |  |  |  |
| 所用算法 | precision | recall | ARI | JI |
| kmeans | 0.376 | 0.428 | 0.187 | 0.250 |
| 层次kmeans | 0.389 | 0.408 | 0.196 | 0.249 |
| meanshift | 0.365 | 0.431 | 0.175 | 0.246 |
| meancut | 0.892 | 0.682 | 0.711 | 0.630 |
| msgc | 0.485 | 0.573 | 0.353 | 0.357 |
| cdc | 0.426 | 0.416 | 0.235 | 0.266 |
| dbscan | 0.246 | 1 | 0 | 0.246 |
| scML |  |  |  |  |
| 所用算法 | precision | recall | ARI | JI |
| kmeans | 0.600 | 0.744 | 0.539 | 0.498 |
| 层次kmeans | 0.661 | 0.739 | 0.591 | 0.536 |
| meanshift | 0.601 | 0.791 | 0.560 | 0.519 |
| meancut | 0.851 | 0.739 | 0.728 | 0.654 |
| msgc | 0.844 | 0.693 | 0.693 | 0.614 |
| cdc | 0.885 | 0.763 | 0.766 | 0.694 |
| dbscan | 0.729 | 0.562 | 0.535 | 0.465 |



由评价指标可知，整体scML降维对七种聚类算法能力的提升显著，在PCA构建的特征空间中Meancut凭借路径相似性增强了簇内关联性，在聚集效果较差的PCA算法降维特征中表现较好；而CDC算法通过识别边缘点区分簇的形态结构，在聚集性较好但仍存在弱连接的scML降维空间中表现出色。