标注点LabeledPoint是一种带有标签（Label/Response）的本地向量，它可以是稠密或者是稀疏的。在MLlib中，标注点在监督学习算法中被使用。由于标签是用双精度浮点型来存储的，故标注点类型在回归（Regression）和分类（Classification）问题上均可使用。例如，对于二分类问题，则正样本的标签为1，负样本的标签为0，而对于多类别的分类问题来说，标签则应是一个以0开始的索引序列:0, 1, 2 ...

标注点的实现类是org.apache.spark.mllib.regression.LabeledPoint，请注意它与前面介绍的本地向量不同，并不位于linalg包下，标注点的创建如下所示：

|  |
| --- |
| scala> import org.apache.spark.mllib.linalg.Vectors  import org.apache.spark.mllib.linalg.Vectors  scala> import org.apache.spark.mllib.regression.LabeledPoint  import org.apache.spark.mllib.regression.LabeledPoint  //创建一个标签为1.0（分类中可视为正样本）的稠密向量标注点  scala> val pos = LabeledPoint(1.0, Vectors.dense(2.0, 0.0, 8.0))  pos: org.apache.spark.mllib.regression.LabeledPoint = (1.0,[2.0,0.0,8.0])  //创建一个标签为0.0（分类中可视为负样本）的稀疏向量标注点  scala> val neg = LabeledPoint(0.0, Vectors.sparse(3, Array(0, 2), Array(2.0, 8.0)))  neg: org.apache.spark.mllib.regression.LabeledPoint = (0.0,(3,[0,2],[2.0,8.0])) |

在实际的机器学习问题中，稀疏向量数据是非常常见的，MLlib提供了读取**LIBSVM**格式数据的支持，该格式被广泛用于**LIBSVM**、**LIBLINEAR**等机器学习库。在该格式下，每一个带标注的样本点由以下格式表示：

label index1:value1 index2:value2 index3:value3 ...

其中label是该样本点的标签值，一系列index:value对则代表了该样本向量中所有非零元素的索引和元素值。这里需要特别注意的是，index是以1开始并递增的。  
MLlib在org.apache.spark.mllib.util.MLUtils工具类中提供了读取**LIBSVM**格式的方法loadLibSVMFile，其使用非常方便。

|  |
| --- |
| scala> import org.apache.spark.mllib.util.MLUtils  import org.apache.spark.mllib.util.MLUtils    // 用loadLibSVMFile方法读入LIBSVM格式数据  // sample\_libsvm\_data.txt为spark自带的一个示例，在以下地址可以找到：  // $SPARK\_HOME$/data/mllib/sample\_libsvm\_data.txt  scala> val examples = MLUtils.loadLibSVMFile(sc, "/data/mllib/sample\_libsvm\_data.txt")  //返回的是组织成RDD的一系列LabeledPoint  examples: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[6] at map at MLUtils.scala:108 |

这里，sc是Spark-shell自动建立的SparkContext。我们可以查看下加载进来的标注点的值：

|  |
| --- |
| scala> examples.collect().head  res7: org.apache.spark.mllib.regression.LabeledPoint = (0.0,(692,[127,128,129,130,131,154,155,156,157,158,159,181,182,183,184,185,186,187,188,189,207,208,209,210,211,212,213,214,215,216,217,235,236,237,238,239,240,241,242,243,244,245,262,263,264,265,266,267,268,269,270,271,272,273,289,290,291,292,293,294,295,296,297,300,301,302,316,317,318,319,320,321,328,329,330,343,344,345,346,347,348,349,356,357,358,371,372,373,374,384,385,386,399,400,401,412,413,414,426,427,428,429,440,441,442,454,455,456,457,466,467,468,469,470,482,483,484,493,494,495,496,497,510,511,512,520,521,522,523,538,539,540,547,548,549,550,566,567,568,569,570,571,572,573,574,575,576,577,578,594,595,596,597,598,599,600,601,602,603,604,622,623,624,625,626,627,628,629,630,651,652,653,654,655,656,657],[51.0,159.0,253.0,159.0,50... |

这里，examples.collect()把rdd转换为了向量，并取第一个元素的值。每个标注点共有692个维，其中第127列对应的值是51.0，第128列对应的值是159.0，依此类推。

|  |
| --- |
| 将RDD转换为LabeledPoint RDD |
| Object RDDToLPTest {  def main(args: Array[String]): Unit = {  val sparkConf = new SparkConf().setAppName("RDDToLPTest ")  val sc = new SparkContext(sparkConf)  //val data:RDD[LabeledPoint] = MLUtils.loadLibSVMFile(sc,"/data/sample.txt")  val data:RDD[LabeledPoint] = sc.textFile("/data/sample.txt").map(line=>create\_label\_point(line))  val numclasses = 2  val featureSubsetStrategy = "auto"  val numTrees = 3  val model:RandomForestModel = RandomForest.trainClassifier(data, Strategy.defaultStrategy("classification"),numTrees,  featureSubsetStrategy,new java.util.Random().nextInt())  //val input:RDD[LabeledPoint] = MLUtils.loadLibSVMFile(sc,"/data/input.txt")  val input:RDD[LabeledPoint] = sc.textFile("/data/input.txt").map(line=>create\_label\_point(line))  val predictResult = input.map { point =>  val prediction = model.predict(point.features)  (point.label,prediction)  }  predictResult.collect().foreach(  x=>println("res"+x)  )  sc.stop()  }  def create\_label\_point(line:String):LabeledPoint = {  //字符串去空格，以逗号分隔转为数组  val linearr = line.trim().split(" ")  val linedoublearr = linearr.map(x=>x.toDouble)  //定长数组转可变数组  val linearrbuff = linedoublearr.toBuffer  //移除label元素（将linedoublearr的第一个元素作为标签）  linearrbuff.remove(0)  //将剩下的元素转为向量  val vectorarr = linearrbuff.toArray  val vector = Vectors.dense(vectorarr)  //返回标签向量  LabeledPoint(linedoublearr(0),vector)  }  } |