Kdtree-scala design document

- 1.Build kdtree
- 2.Search tree

Kdtree一般实现流程

- **构建k-d树的一般算法实现**
  算法：构建k-d树（createKDTree）
- 输入：数据点集Data-set和其所在的空间Range
  输出：Kd，类型为k-d tree
  1、If Data-set为空，则返回空的k-d tree
  2、调用节点生成程序：
  （1）确定split域：对于所有描述子数据（特征矢量），统计它们在每个维上的数据方差。数据方差大表明沿该坐标轴方向上的数据分散得比较开，在这个方向上进行数据分割有较好的分辨率；
  （2）确定Node-data域：数据点集Data-set按其第split域的值排序。位于正中间的那个数据点被选为Node-data。此时新的Data-set' = Data-set\Node-data（除去其中Node-data这一点）。
  3、dataleft = {d属于Data-set' && d[split] ≤ Node-data[split]}
  Left_Range = {Range && dataleft}
  dataright = {d属于Data-set' && d[split] > Node-data[split]}
  Right_Range = {Range && dataright}
  4、left节点 = 由（dataleft，Left_Range）建立的k-d tree，即递归调用createKDTree（dataleft，Left_Range）。并设置left的parent域为Kd；
  right 节点= 由（dataright，Right_Range）建立的k-d tree，即调用createKDTree（dataright，Right_Range）。并设置right的parent域为Kd。

kdtree的构建实现

每个节点的数据类型： TODO:根据以后spark并行还要做更改

KDNode.scala:

- **class** KDNode (
  val label: Int, //点的标签
  val pointData: Array[Double], //点的数据
  val splitAxis: Int, //选择的分割维
  val range: Int, //暂时没用，以后会删掉
  var isLeaf: Boolean, //是否是叶节点
  var leftNode: Option[KDNode], //左节点
  var rightNode: Option[KDNode],//右节点
  var parentNode: Option[KDNode]) **extends** Serializable **with** Logging {

KDTree.scala： createKDTree如下：

1.其中输入Array[LabeledPoint]采用spark milb里面

2.获得分割维(getSplitAxis),并不是用计算各维的方差的方式，而是采用当前节点的深度和数据的维度来获取(网上版本都是这样)，以后有待确认。

最后建好树后，获得了kdtree root节点，用KDTreeModel.scala 形成kdtree模型。

```
def createKDTree(subInput: Array[LabeledPoint], depth: Int = 0, dim: Int): Option[KDNode] = {
  subInput.length match {
    case 0 => Option.empty
    case 1 => Some(KDNode(subInput.head.label.toInt, subInput.head.features.toArray, 0, 1, true, None, None, None))

    case subInputLength =>
      val splitAxis:Int = KDTree.getSplitAxis(depth, dim)
      val (left, rightWithMedian) = subInput.sortBy(_.features(splitAxis)).splitAt(subInputLength/2)
      val newDepth = depth+1
      val current = rightWithMedian.head
      val leftNode = createKDTree(left, newDepth, this.dim)
      val rightNode = createKDTree(rightWithMedian.tail, newDepth, this.dim)
      Option(KDNode(current.label.toInt, current.features.toArray, splitAxis, 1, false, leftNode, rightNode, None))

  }
}
```

```
def getSplitAxis(depth: Int, dim: Int): Int = {
  if (depth==2)
    depth & 1
  else
    depth % dim
}
```

Search tree

- KDTree.scala: findNeighbours方法：
- (目标点：要分类的点)
- 搜索步骤：
- 1.把目标点从root节点划分，一直到叶节点。并记住搜索路径。
- 2.根据搜索路径从最后面开始依次判断K近邻点，在此建立一个K大小的堆，计算当前点和目标点的距离，1>当此距离小于堆的第一个存储点到目标点距离时就入堆并更新堆，此外当前节点如果有左右子节点(且未访问则增加到搜索路径中)。2>如果距离大于堆的第一个存储点则扫描路径的下一个点直到路径为空。堆的点的结构采用类searchedPoint.scala。
- 注：检测节点是否被访问还未加。

并行

是否可以类似于millb中决策树的那种方式？
(没有完全看懂不能给出决策树的并行流程)