《云计算项目实践》课程设计报告

17341147 王桂豪

GitHub 项目 url：https://github.com/VictorWang99/Hadoop-KNN

一、 简介

本次课程设计，我选择了基于 Hadoop 平台设计编写一个 MapReduce 程序这个题目。我的 MapReduce 程序实现了 KNN（K-近邻分类）算法，使用的数据集是非常经典的 KNN 算法数据集—Iris 数据集。

Iris 数据集也称鸢尾花卉数据集，由 Fisher, 1936 收集整理。数据集包含 150 个数据样本，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

KNN 算法是对于一个测试样本，分别计算与所有训练样本属性的距离，选取 K 个距离最小的样本，以多数投票原则选择这 K 个样本中最多的类别作为测试样本的分类。

二、 关键代码

1、Iris 类的封装

实现一个 Iris 类，包含四个属性以及一个标签共 5 个数据成员。同时还实现了一个方法 distance，用来计算两个 Iris 属性之间的欧式距离。

```
public static class Iris {
    // 属性
    public double sepalLength, sepalWidth, petalLength, petalWidth;
    // 类别
    public String label;
    public Iris(String[] data) {
        sepalLength = Double.parseDouble(data[0]);
        sepalWidth = Double.parseDouble(data[1]);
        petalLength = Double.parseDouble(data[2]);
        petalWidth = Double.parseDouble(data[3]);
        if(data.length==5){
            label = data[4];
        }
    }
    // 计算两个样本的欧式距离
    public double distance(Iris another) {
        double sum = 0;
        sum += Math.pow(another.sepalLength - sepalLength, 2);
        sum += Math.pow(another.sepalWidth - sepalWidth, 2);
        sum += Math.pow(another.petalLength - petalLength, 2);
        sum += Math.pow(another.petalWidth - petalWidth, 2);
        return Math.sqrt(sum);
    }
}
```

2、Mapper 的实现

在 Mapper 中，我们每次读取训练集中的一行，即训练集的一个样本。对于测试集中的每个测试样本，计算与这个训练样本的欧式距离。之后，将测试样本的编号作为 mapper 输出的 key，将距离以及训练样本的类别作为 mapper 输出的 value。这样，我们就可以通过 mapper 过程，计算每个训练样本与每个测试样本的距离了。

```
public static class KNNMapper extends Mapper<LongWritable, Text, IntWritable, Text> {
        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
                String line = value.toString();
                // 划分以空格分隔的数据
                String[] data = line.split("\\s+");
                // 构建训练样本
                Iris trainData = new Iris(data);
                // 对测试集中的所有样本，计算与该训练样本的欧式距离，将测试样本的编号作为key，将距离与训练样本的label的组合作为value传给reducer
                for (int i = 0; i < testDatas.size(); i++) {
                    Iris testData = testDatas.get(i);
                    double distance = trainData.distance(testData);
                    Text valueInfo = new Text(Double.toString(distance)+','+trainData.label);
                    context.write(new IntWritable(i), valueInfo);
                }
        }
}
```

3、Reducer 的实现

在这里，MapReduce 框架会把 key 值相同的 value 合并。也就是说，我们得到了第 key 个测试样本与各个训练样本的距离以及训练样本的类别。那么我们需要按照距离从小到大排序。在排序这个过程，我们利用了 TreeMap 这个数据结构，该结构会对数据自动排序储存。

```
TreeMap<Double, String> disMapToLabel = new TreeMap<>();
// 对于第key个测试样本，将与所有训练样本的距离以及训练样本的label放入TreeMap中，TreeMap会默认以距离升序排列。
for (Text value : values) {
        double distance = Double.parseDouble(value.toString().split(",")[0]);
        String label = value.toString().split(",")[1];
        disMapToLabel.put(distance, label);
}
```

之后选取前 K 条记录的类别做多数投票，就可以得到这个测试样本分类的预测了。

```
// 统计距离最小的前k个label的数量，以多数投票原则得到预测的label
int setosa_count=0, versicolor_count=0, virginica_count=0;
Iterator<Double> iterator =  disMapToLabel.keySet().iterator();
int i = 0;
while(iterator.hasNext()) {
    double dis = iterator.next();
    String label = disMapToLabel.get(dis);
    if(label=="Iris-setosa") setosa_count++;
    else if(label=="Iris-versicolor") versicolor_count++;
    else if(label=="Iris-virginica") virginica_count++;
    i++;
    if(i>=K) break;
}
String testLabel;
if(setosa_count > versicolor_count && setosa_count > virginica_count){
    testLabel = "Iris-setosa";
}
else if(versicolor_count > setosa_count && versicolor_count > virginica_count){
    testLabel = "Iris-versicolor";
}
else testLabel = "Iris-virginica";
// 最后输出预测结果
context.write(key, new Text(testLabel));
```

三、 实验过程

1、编译打包结果

```
hadoop@VirtualLab:~/VirtualLab/KNN$ vim KNN.java
hadoop@VirtualLab:~/VirtualLab/KNN$ javac KNN.java
hadoop@VirtualLab:~/VirtualLab/KNN$ ls
 iris-test.txt  'KNN$Iris.class'  'KNN$KNNMapper.class'    train
 KNN.class       KNN.java         'KNN$KNNReducer.class'
hadoop@VirtualLab:~/VirtualLab/KNN$ jar cvf KNN.jar ./KNN*.class
已添加清单
正在添加: KNN.class(输入 = 2466) (输出 = 1358)(压缩了 44%)
正在添加: KNN$Iris.class(输入 = 907) (输出 = 566)(压缩了 37%)
正在添加: KNN$KNNMapper.class(输入 = 2529) (输出 = 1080)(压缩了 57%)
正在添加: KNN$KNNReducer.class(输入 = 2550) (输出 = 1186)(压缩了 53%)
hadoop@VirtualLab:~/VirtualLab/KNN$ ls
 iris-test.txt  'KNN$Iris.class'   KNN.java              'KNN$KNNReducer.class'
 KNN.class       KNN.jar          'KNN$KNNMapper.class'   train
```

2、运行结果

输入运行命令，第一个参数是训练集文件夹，第二个参数是测试集文件，第三个参数是输出文件夹，第四个参数是 KNN 中的 K（这个参数是可选的，默认为 10）

```
hadoop@VirtualLab:~/VirtualLab/KNN$ /usr/local/hadoop/bin/hadoop jar KNN.jar KNN
 train test.txt output 3
```

```
        File Input Format Counters
                Bytes Read=4108
        File Output Format Counters
                Bytes Written=182
hadoop@VirtualLab:~/VirtualLab/KNN$ ls
 KNN.class         KNN.jar   'KNN$KNNMapper.class'    output     train
'KNN$Iris.class'   KNN.java  'KNN$KNNReducer.class'   test.txt
hadoop@VirtualLab:~/VirtualLab/KNN$ ls output
part-r-00000  _SUCCESS
```

3、预测结果

选择数据集的前 140 条作为训练集，后 10 条去掉 label 作为测试集，预测结果如下，符合预期，KNN 算法运行成功。

```
hadoop@VirtualLab:~/VirtualLab/KNN$ head output/part-r-00000
0       Iris-virginica
1       Iris-virginica
2       Iris-virginica
3       Iris-virginica
4       Iris-virginica
5       Iris-virginica
6       Iris-virginica
7       Iris-virginica
8       Iris-virginica
9       Iris-virginica
```

四、 实验总结

本次实验过程中，遇到的主要问题就是如何对多个测试样本进行预

测。因为开始时构思的是 mapper 的输出 key 是距离，value 是类别，这样就无法在 reducer 中对多个测试样本进行区分。后来，将测试样本的 id 作为 mapper 输出的 key，这样就能在 reducer 中对同一个测试样本进行汇集，解决了这个问题。

实验中遇到的其他问题主要来自对于 java 语言的不熟悉，像 TreeMap 数据结构的使用、文件的 IO 等都需要现学现用。不过总体来说还算顺利。

通过本次课程设计，对于基于 hadoop 平台进行 MapReduce 程序开发有了更进一步的练习。同时学习了其在机器学习算法方面的应用，并通过一个 KNN 算法的实验进行了切身实践。