

《金融大数据处理技术》作业7报告

191870068 嵯澤同


《金融大数据处理技术》作业7报告

- 一、题目要求
- 二、结果截图
- 三、设计思路
 - 1.基本思路
 - 2.距离计算
 - 3.自定义数据类型
 - 4.map阶段
 - 5.Combine阶段
 - 6.Reduce阶段
- 四、可视化

一、题目要求

Iris数据集是常用的分类实验数据集，由Fisher, 1936收集整理。Iris也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含150个数据，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。在MapReduce上任选一种分类算法（KNN, 朴素贝叶斯或决策树）对该数据集进行分类预测，采用留出法对建模结果评估，70%数据作为训练集，30%数据作为测试集，评估标准采用精度accuracy。可以尝试对结果进行可视化的展示（可选）。

二、结果截图



All Applications

Cluster

[About](#)
[Nodes](#)
[Node Labels](#)
[Applications](#)

[NEW](#)
[NEW_SAVING](#)
[SUBMITTED](#)
[ACCEPTED](#)
[RUNNING](#)
[FINISHED](#)
[FAILED](#)
[KILLED](#)
[Scheduler](#)

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total
1	0	0	1	0	0 B	20 GB

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
1	0	0	0


Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:2048, vCores:1>

Show 20 ▼ entries


ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Container
application_1636945448041_0001	jzt	KNN	MAPREDUCE		default	0	Mon Nov 15 21:13:43 +0800 2021	Mon Nov 15 21:13:46 +0800 2021	Mon Nov 15 21:14:54 +0800 2021	FINISHED	SUCCEEDED	N/A

Showing 1 to 1 of 1 entries

 part-r-00000 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
143:virginica——predicted:virginica
144:virginica——predicted:virginica
145:virginica——predicted:virginica
146:virginica——predicted:virginica
147:virginica——predicted:virginica
148:virginica——predicted:virginica
149:virginica——predicted:virginica
150:virginica——predicted:versicolor
距离计算方式: euclid
k: 3
accuracy: 0.9555555555555556
```

 part-r-00000 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
143:virginica——predicted:versicolor
144:virginica——predicted:virginica
145:virginica——predicted:virginica
146:virginica——predicted:versicolor
147:virginica——predicted:versicolor
148:virginica——predicted:virginica
149:virginica——predicted:virginica
150:virginica——predicted:versicolor
距离计算方式: manhattan
k: 5
accuracy: 0.8666666666666667
```

```
143:virginica——predicted:versicolor
144:virginica——predicted:virginica
145:virginica——predicted:virginica
146:virginica——predicted:virginica
147:virginica——predicted:versicolor
148:virginica——predicted:versicolor
149:virginica——predicted:versicolor
150:virginica——predicted:versicolor
距离计算方式: chebyshev
k: 10
accuracy: 0.8
```

![image-20211115223322292](C:\Users\jzt\AppData\Roaming\Typora\typora-user-images\image-20211115223322292.png)

三、设计思路

1.基本思路

KNN算法中，可以将训练集放到内存中，也可以将测试集放到内存中。本次作业中我参照讲义上的实现，将训练集放到内存中，自定义数据类型`IdDistance`，在`map`阶段将每一行数据（以`id`标识）分别与训练集（内存中）中的每一行数据对比，计算对应的距离，并且将得到的信息保存到自定义的数据类型中并作为`key`输出。在`combine`阶段则在每个`id`对应的所有`map`输出中只保留距离前`k`小的`k`个，以减少发送到`reducer`的信息量。在`reduce`阶段则根据每个`id`对应的`k`条信息，根据多数原则“票选”出分类结果，并且记录该分类结果与实际结果是否一致，用于最后的`accuracy`计算。

2.距离计算

本次作业中我考虑了三种较简单的距离计算方式：欧氏距离、曼哈顿距离以及切比雪夫距离，并且允许用户通过传入参数来决定使用哪种距离计算方式：

```
public class Distance {
    static public double CalcuDistance(String method, double[] a, double[] b) {
        int length = a.length;
        double result = 0;
        if (method.equals("manhattan")) { // 曼哈顿距离
            for(int i = 0; i < length; i++) {
                result = result + Math.abs(a[i]-b[i]) ;
            }
            return result;
        }
        else if (method.equals("chebyshev")) { // 切比雪夫距离
            for(int i = 0; i < length; i++) {
                if (Math.abs(a[i]-b[i]) > result)
                    result = Math.abs(a[i]-b[i]);
            }
        }
    }
}
```

```

    }
    return result;
}
else {
    for(int i = 0; i < length; i++) { // 欧氏距离，默认情况下所采用的方式
        result = result + (a[i]-b[i]) * (a[i] - b[i]);
    }
    return Math.sqrt(result);
}
}
}
}

```

3.自定义数据类型

在map阶段进行距离计算后，需要把得到的信息传输到reducer，用自带的数据类型作为key/value较难实现，因此我自定义了一个数据类型IdDistance，用以保存测试样本的id、与其比较的训练样本的id、两者间的距离。为了后续shuffle过程中排序顺利，自定义该数据类型的compareTo方法使得优先比较测试样本id，在测试样本id相同的情况下再比较距离。这样的处理使得combine/reduce阶段接收到的前k个数据就是前k小的k个数据。值得注意的是，由于自定义了数据类型作为key，因此需要自定义partitioner以实现将具有相同测试样本id的数据传输到同一个reducer节点。

```

public static class IdDistance implements WritableComparable<IdDistance> {
    private String id; // 测试样本的id
    private double distance; // 测试样本和训练样本间的距离
    private String target; // 训练样本的id

    public IdDistance() { }

    public IdDistance(String id, double distance, String target) {
        this.id = id;
        this.distance = distance;
        this.target = target;
    }

    public void set(String id, double distance, String target) {
        this.id = id;
        this.distance = distance;
        this.target = target;
    }

    public void readFields(DataInput in) throws IOException {
        this.id = in.readUTF();
        this.distance = in.readDouble();
        this.target = in.readUTF();
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(this.id);
        out.writeDouble(this.distance);
        out.writeUTF(this.target);
    }

    public String toString() {
        return this.id + ":" + this.distance + "-" + this.target;
    }
}

```

```

        public int compareTo(IdDistance o) {
            if (!this.id.equals(o.id)) return Integer.parseInt(this.id) <
Integer.parseInt(o.id) ? -1:
                (Integer.parseInt(this.id)==Integer.parseInt(o.id)?0 : 1);
            else {
                return this.distance < o.distance ? -1 : (this.distance ==
o.distance ? 0 : 1);
            }
        }
    }
}

```

```

public class IdPartitioner extends Partitioner<IdDistance, Text> {
    @Override
    public int getPartition(IdDistance idDistance, Text text, int i) {
        return (idDistance.id.hashCode() & 2147483647) % i;
    }
}

```

4.map阶段

首先在setup()方法中读取CacheFile（通过命令行参数传入），并将其内容按行存入ArrayList中，同时读取conf中的计算距离的方法（通过命令行参数传入）。

map()方法的实现也较为简单，只需读取测试样本中的一条数据a，split提取出四个特征，并对于内存中的每一条训练样本b，都计算a的特征和b的特征间的距离，将计算得到的距离、a的id和b的id都存入自定义数据类型IdDistance的实例中，并将其作为key传出，同时将a实际对应的分类作为value传出。伪代码如下：

```

class Mapper{
    ArrayList trainData;
    setup(...){
        ... // 读取CacheFile存入内存trainData中，同时读取其他后续所需要的信息
    }
    map(key, value, context){
        testData = value.split(",");
        testData -> testId, testCategory;
        for(trainSample:trainData){
            sampleData = tranData.split(",");
            sampleData -> trainId;
            distance = calcuDistance(testData,sampleData);
            testId,trainId,distance -> newKey;
            testCategory -> newValue;
            emit(newKey, newValue);
        }
    }
}

```

5.Combine阶段

combine阶段接收来自maper的数据，由于这些数据已经按照测试样本id和距离排好序，因此只需设置一个k（从命令行参数传入）用于计数，然后对于每个测试样本，只保留接收到的前k条数据即可。伪代码如下：

```

class Combiner{
    int k;
}

```

```

int left; // 记录还有几条需要保留的数据
String curId; // 记录当前正在处理的测试样本的id
setup(...){
    ... // 从conf中读取k值并记录
}
reduce(key, values, context){
    if key.id != curId {
        curId = key.id;
        left = k;
    }
    if left != 0 {
        for(value:values){
            emit(key, value);
            left--;
            if left == 0
                break;
        }
    }
}
}

```

6.Reduce阶段

由于combine阶段已经只保留了前k小的数据，因此reduce阶段只需要对于每个测试样本a，根据接收到的对应的k条数据，“票选”出模型给出的a的分类结果，并且与a的实际分类对比并记录，若模型结果与实际结果一致，则correctNumber++。最后根据correctNumber/totalNumber即可得出accuracy。伪代码如下：

```

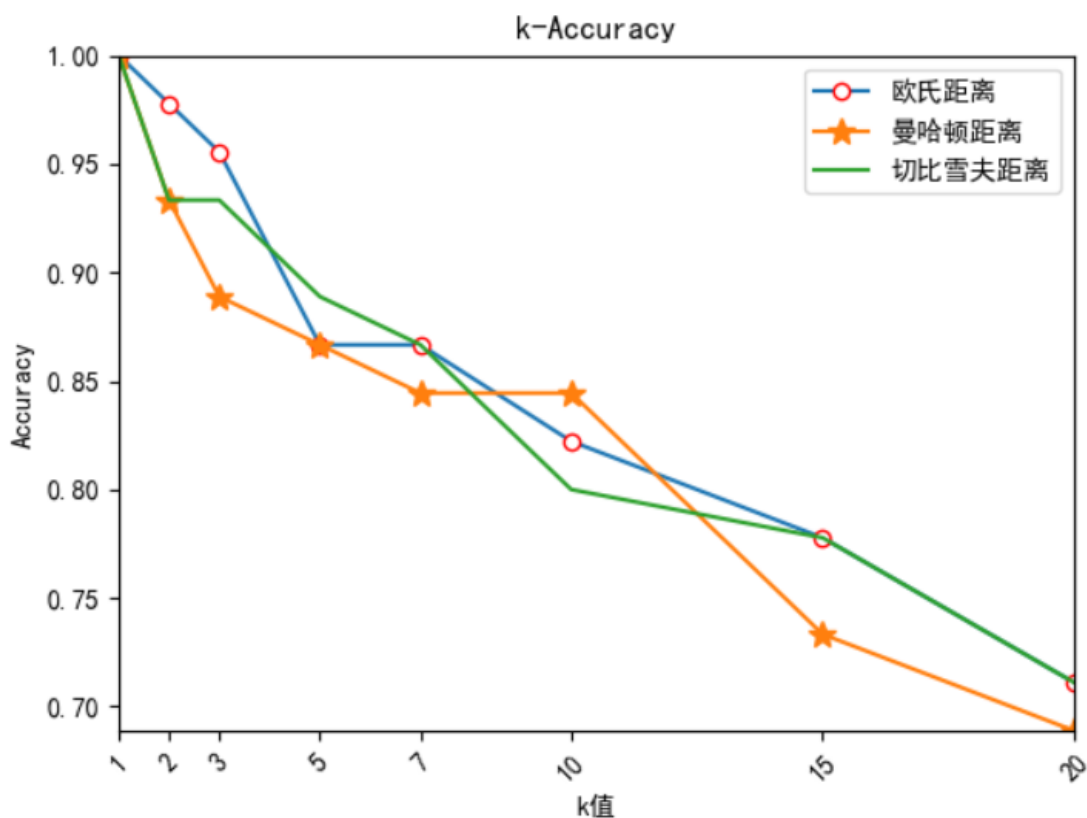
class Reducer{
    int k;
    int left; // 用于记录还要读取多少条数据
    int correctNum; // 记录被正确分类的测试样本的数目
    int totalNum; // 记录测试样本总数
    HashMap<String, Integer> hm; // 用于保存投票结果
    setup(...){
        ... // 从context中读取k值，并初始化上述变量的值
    }
    reduce(key, values, context) {
        for(value:values){
            if left != 0{
                hm.update; // hm中对应分类的票数+1
                left--;
            }
            if left == 0 { // 所有k条数据已经处理完毕，进行最终票选
                resultCategory = getVoteResult(hm);
                if resultCategory == trueCategory {
                    correctNum++;
                }
                totalNum++;
                hm.clear();
                left = k;
                emit(resultText);
            }
        }
    }
    cleanup(...){
        ... // 输出相关信息，如k值、计算距离方法等
    }
}

```

```
emit(correctNum/totalNum); // 输出accuracy
    }
}
```

四、可视化

在本次作业中，我依次采用了欧氏距离、曼哈顿距离和切比雪夫距离，设置不同的k值，得到结果绘制图像如下：



可以看到，不同的距离计算方式总体而言差异不大。值得注意的是，随着k值的增大，accuracy整体呈下降趋势。按道理而言，大多数情况下准确度应该随着k值的增大呈现先上升后下降趋势，因为k值太小则随机性过大，k值过大则考虑了过多其他分类的样本。本次作业中准确度随k值单调下降可能是由于样本因素，比如训练集和测试集都过小，在这种情况下，k=1时，即直接取距离测试样本最近的分类作为该样本的分类时，accuracy达到100%，而这在通常情况下是不大可能的。