实验2 MapReduce实验报告

姓名：滕子鉴

学号：231275015

实验时间：2025年10月

一、实验目的

1. 掌握MapReduce编程模型的基本原理
2. 学习使用Hadoop平台进行分布式数据处理
3. 通过真实数据分析优惠券使用规律

二、实验环境

- **操作系统**：Ubuntu 22.04 LTS
- **Hadoop版本**：Hadoop 3.4.0
- **Java版本**：OpenJDK 1.8
- **运行模式**：伪分布式模式
- **开发工具**：文本编辑器 + 命令行编译

三、任务一：消费行为统计

3.1 任务目标

统计每个商家的优惠券使用情况，分为三种类型：

1. **负样本数**：领取优惠券但未使用
2. **普通消费数**：未领取优惠券直接消费
3. **正样本数**：领取优惠券并使用

输出格式：`<Merchant_id> TAB <负样本数> TAB <普通消费数> TAB <正样本数>`

3.2 设计思路

核心逻辑伪代码

Mapper伪代码：

```
function Map(key, line):
    if line是表头:
        return

    解析line得到字段：Merchant_id, Coupon_id, Date_received, Date

    // 根据文件名判断是线上还是线下数据
    if 是线上数据:
        if Action == 1 且 Coupon_id == null:
            // 普通消费
            emit(Merchant_id, "0,1,0")
        else if Action == 2 且 Coupon_id != null:
            if Date != null:
```

```
                // 领取并使用
                emit(Merchant_id, "0,0,1")
            else:
                // 领取未使用
                emit(Merchant_id, "1,0,0")
    else:  // 线下数据
        if Coupon_id != null 且 Date_received != null:
            if Date != null:
                // 领取并使用
                emit(Merchant_id, "0,0,1")
            else:
                // 领取未使用
                emit(Merchant_id, "1,0,0")
        else if Coupon_id == null 且 Date != null:
            // 普通消费
            emit(Merchant_id, "0,1,0")
```

Reducer伪代码:

```
function Reduce(Merchant_id, values):
    negative_count = 0   // 负样本
    normal_count = 0     // 普通消费
    positive_count = 0   // 正样本

    for each value in values:
        解析value得到三个计数
        negative_count += value的第一个数
        normal_count += value的第二个数
        positive_count += value的第三个数

    emit(Merchant_id, negative_count + "\t" + normal_count + "\t" +
positive_count)
```

3.3 核心代码实现

Mapper核心代码:

```
@Override
protected void setup(Context context) throws IOException,
InterruptedException {
    FileSplit fileSplit = (FileSplit) context.getInputSplit();
    String fileName = fileSplit.getPath().getName();
    isOnlineData = fileName.contains("online");
}

@Override
protected void map(LongWritable key, Text value, Context context) {
    // 跳过表头
    if (key.get() == 0) return;

    String[] fields = line.split(",");

    if (isOnlineData) {
        // 线上数据处理逻辑
```

```
        String action = fields[2].trim();
        if ("1".equals(action) && "null".equals(couponId)) {
            valueOut.set("0,1,0");  // 普通消费
        } else if ("2".equals(action) && !"null".equals(couponId)) {
            if (!"null".equals(date)) {
                valueOut.set("0,0,1");  // 使用
            } else {
                valueOut.set("1,0,0");  // 未使用
            }
        }
    } else {
        // 线下数据处理逻辑
        if (!"null".equals(couponId) && !"null".equals(dateReceived)) {
            if (!"null".equals(date)) {
                valueOut.set("0,0,1");
            } else {
                valueOut.set("1,0,0");
            }
        } else if ("null".equals(couponId) && !"null".equals(date)) {
            valueOut.set("0,1,0");
        }
    }

    context.write(merchantId, valueOut);
}
```

Reducer核心代码：

```
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) {
    int negativeCount = 0, normalCount = 0, positiveCount = 0;

    for (Text value : values) {
        String[] counts = value.toString().split(",");
        negativeCount += Integer.parseInt(counts[0]);
        normalCount += Integer.parseInt(counts[1]);
        positiveCount += Integer.parseInt(counts[2]);
    }

    result.set(negativeCount + "\t" + normalCount + "\t" + positiveCount);
    context.write(key, result);
}
```

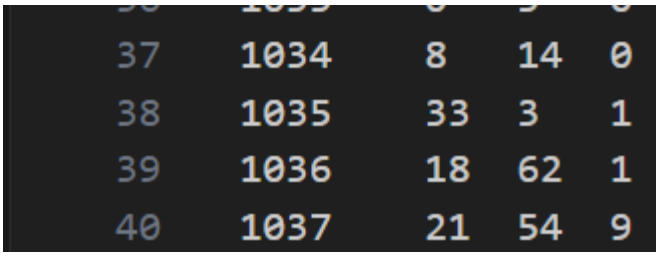
3.4 运行结果

线下数据结果（部分）

```
商家ID    负样本数    普通消费数    正样本数
```

task1_offline_result.txt

```
1    0   12  0
100 1    1   0
1000     0   5   0
1001     7   20  14
1002     5   8   0
1003     0   3   0
1004     82  281 1
1005     30  19  1
1007     3   22  0
1008     0   11  0
1009     0   1   0
101 0    4   0
1010     3   10  2
1011     63  59  6
1013     29  64  5
1015     19  16  1
1016     10  5   1
1017     8   9   0
1018     4   6   5
1019     0   4   0
102 1    3   0
1020     0   3   0
1021     0   5   0
1022     0   7   1
1023     4   41  8
1024     0   14  0
1025     10  6   2
1026     1   17  1
1027     0   13  0
1028     4   24  2
1029     2   0   1
103 0    4   0
1030     0   11  0
1031     2   2   0
1032     2   0   0
1033     0   3   0
```



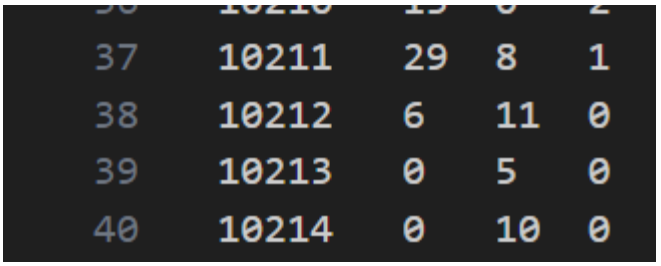
37	1034	8	14	0
38	1035	33	3	1
39	1036	18	62	1
40	1037	21	54	9

线上数据结果（部分）

```
商家ID	负样本数	普通消费数	正样本数
```

task1_online_result.txt

1	10001	118	53	11
2	10002	0	6	0
3	10003	0	1	0
4	10004	0	3	0
5	10005	0	3	0
6	10006	95	2	2
7	10007	0	172	0
8	10008	0	36	0
9	10009	0	12	0
10	10010	0	1	0
11	10011	0	4	0
12	10012	13	2	1
13	10014	31	14	0
14	10100	0	4	0
15	10102	5	2	2
16	10103	0	28	0
17	10104	0	4	0
18	10105	0	5	0
19	10106	0	19	0
20	10107	4	9	0
21	10108	0	1	0
22	10109	0	2	0
23	10110	2	11	1
24	10111	11	2	3
25	10112	13	3	19
26	10113	0	7	0
27	10114	5	150	0
28	10201	0	4	0
29	10202	0	2	0
30	10203	0	5	1
31	10205	0	5	0
32	10206	0	11	0
33	10207	0	7	0
34	10208	0	1	0
35	10209	0	16	0
36	10210	13	0	2



37	10211	29	8	1
38	10212	6	11	0
39	10213	0	5	0
40	10214	0	10	0

3.5 可能的改进

1. 统计不同时间段的消费行为变化
2. 结合User_id分析不同用户群体的行为差异

四、任务二：商家周边活跃顾客数量统计

4.1 任务目标

统计每个商家与周边消费者的距离分布，给出不同距离的活跃消费者人数。

输出格式：`<Merchant_id> TAB Distance=<距离>:<消费者人数>`

4.2 设计思路

核心逻辑伪代码

Mapper伪代码：

```
function Map(key, line):
    if line是表头:
        return

    解析line得到: User_id, Merchant_id, Distance

    if Distance == null:
        Distance = "null"

    // 组合key: Merchant_id,Distance
    composite_key = Merchant_id + "," + Distance

    emit(composite_key, User_id)
```

Reducer伪代码：

```
function Reduce(composite_key, user_ids):
    // composite_key格式：Merchant_id,Distance
    解析得到：Merchant_id, Distance

    // 使用HashSet去重统计不同用户
    unique_users = HashSet()
    for each user_id in user_ids:
        unique_users.add(user_id)

    user_count = unique_users.size()

    emit(Merchant_id, "Distance=" + Distance + ":" + user_count)
```

4.3 核心代码实现

Mapper核心代码：

```
@Override
protected void map(LongWritable key, Text value, Context context) {
    String[] fields = line.split(",");

    String userId = fields[0].trim();
    String merchantId = fields[1].trim();
    String distance = fields[4].trim();

    // 处理null值
    if ("null".equals(distance) || distance.isEmpty()) {
        distance = "null";
    }

    // 组合键：Merchant_id,Distance
    outputKey.set(merchantId + "," + distance);
    outputValue.set(userId);

    context.write(outputKey, outputValue);
}
```

Reducer核心代码：

```
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) {
    // 使用HashSet去重
    HashSet<String> uniqueUsers = new HashSet<>();

    for (Text value : values) {
        uniqueUsers.add(value.toString());
    }

    // 解析key
    String[] keyParts = key.toString().split(",");
    String merchantId = keyParts[0];
    String distance = keyParts[1];

    outputKey.set(merchantId);
```

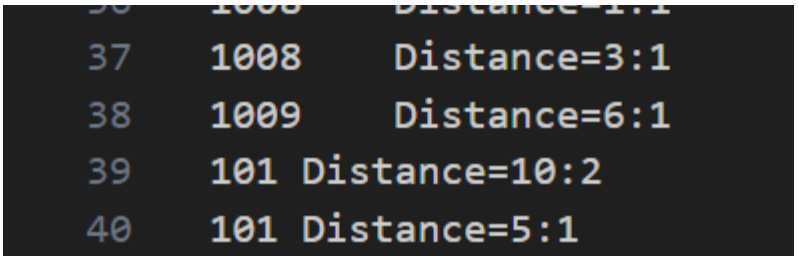
```
        outputValue.set("Distance=" + distance + ":" + uniqueUsers.size());

        context.write(outputKey, outputValue);
    }
```

4.4 运行结果（部分）

```
商家ID    距离分布
```

task2_result.txt

```
1    Distance=0:2
100  Distance=1:1
1000     Distance=0:1
1001     Distance=0:4
1001     Distance=10:2
1001     Distance=3:1
1001     Distance=5:1
1001     Distance=7:1
1001     Distance=null:2
1002     Distance=0:3
1003     Distance=0:1
1003     Distance=1:1
1004     Distance=0:30
1004     Distance=1:5
1004     Distance=10:1
1004     Distance=2:2
1004     Distance=5:2
1004     Distance=7:2
1004     Distance=null:3
1005     Distance=0:9
1005     Distance=1:4
1005     Distance=10:3
1005     Distance=2:3
1005     Distance=3:1
1005     Distance=4:1
1005     Distance=5:1
1005     Distance=null:9
1007     Distance=0:4
1007     Distance=1:1
1007     Distance=10:5
1007     Distance=3:4
1007     Distance=4:2
1007     Distance=5:1
1007     Distance=8:1
1007     Distance=9:1
1008     Distance=1:1
```



4.5 可能的改进

1. 生成商家周边消费者分布热力图
2. 结合时间维度分析不同时段的距离分布变化

五、任务三：优惠券使用时间统计

5.1 任务目标

统计每种优惠券的使用次数，对于使用次数大于总使用次数1%的优惠券，计算从领取到使用的平均间隔天数并排序。

输出格式：`<Coupon_id> TAB <平均消费间隔天数>`

5.2 设计思路

本任务需要**两阶段MapReduce**：

- **第一阶段**：统计每个优惠券的使用次数和总间隔天数
- **第二阶段**：过滤使用次数>1%阈值的优惠券，计算平均间隔并排序

第一阶段伪代码

Mapper1伪代码：

```
function Map1(key, line):
    if line是表头:
        return

    解析line得到：Coupon_id, Date_received, Date

    if Coupon_id == null 或 Date == null 或 Date_received == null:
        return  // 只统计被使用的优惠券

    // 计算日期间隔
    interval_days = Date - Date_received

    emit(Coupon_id, interval_days)
```

Reducer1伪代码：

```
function Reduce1(Coupon_id, interval_days_list):
    count = 0
    total_days = 0

    for each interval in interval_days_list:
        count += 1
        total_days += interval

    emit(Coupon_id, count + "," + total_days)
```

第二阶段伪代码

Mapper2伪代码:

```
function Setup():
    // 读取第一阶段输出，计算总使用次数
    total_usage_count = 0
    for each line in stage1_output:
        解析得到count
        total_usage_count += count

    threshold = total_usage_count * 0.01

function Map2(key, line):
    解析line得到：Coupon_id, count, total_days

    if count > threshold:
        avg_days = total_days / count

        // 使用avg_days作为key实现排序
        emit(格式化的avg_days, Coupon_id + "\t" + avg_days)
```

Reducer2伪代码:

```
function Reduce2(avg_days, coupon_info_list):
    // 直接输出，已经按avg_days排序
    for each info in coupon_info_list:
        解析得到：Coupon_id, avg_days
        emit(Coupon_id, avg_days)
```

5.3 核心代码实现

第一阶段Mapper核心代码:

```
private SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

@Override
protected void map(LongWritable key, Text value, Context context) {
    String[] fields = line.split(",");

    String coupon = fields[2].trim();
    String dateReceived = fields[5].trim();
    String date = fields[6].trim();
```

```
10
11      // 只统计被使用的优惠券
12      if ("null".equals(coupon) || "null".equals(date) ||
13          "null".equals(dateReceived)) {
14          return;
15      }
16
17      // 计算日期间隔
18      Date receivedDate = dateFormat.parse(dateReceived);
19      Date usedDate = dateFormat.parse(date);
20      long diffInMillis = usedDate.getTime() - receivedDate.getTime();
21      long diffInDays = diffInMillis / (1000 * 60 * 60 * 24);
22
23      couponId.set(coupon);
24      outputValue.set(String.valueOf(diffInDays));
25      context.write(couponId, outputValue);
26  }
```

第二阶段Mapper核心代码:

```
1   @Override
2   protected void setup(Context context) throws IOException {
3       // 读取第一阶段输出
4       Configuration conf = context.getConfiguration();
5       String tempPath = conf.get("temp.output.path");
6       FileSystem fs = FileSystem.get(conf);
7
8       BufferedReader br = new BufferedReader(
9           new InputStreamReader(fs.open(new Path(tempPath + "/part-r-00000")))
10      );
11
12      long totalUsageCount = 0;
13      String line;
14      while ((line = br.readLine()) != null) {
15          String[] parts = line.split("\t");
16          String[] countAndTotal = parts[1].split(",");
17          totalUsageCount += Long.parseLong(countAndTotal[0]);
18      }
19      br.close();
20
21      threshold = totalUsageCount * 0.01;
22  }
23
24  @Override
25  protected void map(LongWritable key, Text value, Context context) {
26      String[] parts = line.split("\t");
27      String couponId = parts[0];
28      String[] countAndTotal = parts[1].split(",");
29
30      long count = Long.parseLong(countAndTotal[0]);
31      long totalDays = Long.parseLong(countAndTotal[1]);
32
33      if (count > threshold) {
34          double avgDays = (double) totalDays / count;
35
```

```
        // 格式化用于排序
        String sortKey = String.format("%010.2f", avgDays);
        Text outputKey = new Text(sortKey);
        Text outputValue = new Text(couponId + "\t" +
                                    String.format("%.2f", avgDays));

        context.write(outputKey, outputValue);
    }
}
```

5.4 运行结果

```
优惠券ID    平均间隔天数
```

task3_result.txt

1	7751	11.22
2	111	11.24
3	9776	12.30
4	10323	4.96
5	5686	4.97
6	12034	8.54
7	11539	8.65
8	4773	8.79
9	2418	9.18

5.5 可能的改进

1. 可以尝试0.5%或0.1%的阈值，获取更多样本
2. 统计不同间隔区间的优惠券数量分布
3. 除了平均值，还可以统计中位数

六、任务四：优惠券使用影响因素分析

6.1 任务目标

自行选取可能影响优惠券使用行为的因素，通过MapReduce统计分析这些因素对优惠券使用率的影响。

我选择分析三个因素：

1. **折扣率对使用率的影响**
2. **领取时间（周末vs工作日）对使用率的影响**
3. **商家发券量对使用率的影响**

6.2 分析一：折扣率对使用率的影响

设计思路伪代码

Mapper伪代码：

```
function Map(key, line):
    解析line得到: Coupon_id, Discount_rate, Date_received, Date

    if Coupon_id == null 或 Date_received == null:
        return

    // 解析折扣率并分类
    if Discount_rate包含":":
        // 满减类型: x:y
        actual_rate = (x - y) / x
    else:
        // 直接折扣
        actual_rate = Discount_rate

    // 分类
    if actual_rate >= 0.9:
        category = "0.9-1.0_低折扣"
    else if actual_rate >= 0.8:
        category = "0.8-0.9_中低折扣"
    else if actual_rate >= 0.7:
        category = "0.7-0.8_中等折扣"
    else if actual_rate >= 0.5:
        category = "0.5-0.7_中高折扣"
    else:
        category = "0.0-0.5_高折扣"

    if Date != null:
        status = "used"
    else:
        status = "unused"

    emit(category, status)
```

Reducer伪代码：

```
function Reduce(category, status_list):
    used_count = 0
    unused_count = 0

    for each status in status_list:
        if status == "used":
            used_count += 1
        else:
            unused_count += 1

    total = used_count + unused_count
    usage_rate = (used_count / total) * 100

    emit(category, "领取:" + total + ", 使用:" + used_count +
```

```
                    ", 使用率:" + usage_rate + "%")
```

核心代码实现

折扣率解析函数:

```
private String parseDiscountCategory(String discountRate) {
    if ("null".equals(discountRate) || discountRate.isEmpty()) {
        return null;
    }

    try {
        if (discountRate.contains(":")) {
            // 满减类型: x:y
            String[] parts = discountRate.split(":");
            double full = Double.parseDouble(parts[0]);
            double minus = Double.parseDouble(parts[1]);
            double actualRate = (full - minus) / full;
            return categorizeRate(actualRate);
        } else {
            // 直接折扣
            double rate = Double.parseDouble(discountRate);
            return categorizeRate(rate);
        }
    } catch (NumberFormatException e) {
        return null;
    }
}

private String categorizeRate(double rate) {
    if (rate >= 0.9) return "0.9-1.0_低折扣";
    else if (rate >= 0.8) return "0.8-0.9_中低折扣";
    else if (rate >= 0.7) return "0.7-0.8_中等折扣";
    else if (rate >= 0.5) return "0.5-0.7_中高折扣";
    else return "0.0-0.5_高折扣";
}
```

运行结果

```
折扣类别                统计结果
```

task4_discount_result.txt

```
0.0-0.5_高折扣  领取:59, 使用:3, 使用率:5.08%
0.5-0.7_中高折扣    领取:16100, 使用:1995, 使用率:12.39%
0.7-0.8_中等折扣    领取:54275, 使用:7434, 使用率:13.70%
0.8-0.9_中低折扣    领取:3867, 使用:800, 使用率:20.69%
0.9-1.0_低折扣  领取:58292, 使用:9174, 使用率:15.74%
```

结果分析

关键发现：

1. **中低折扣（0.8-0.9）使用率最高**：达到20.69%
2. **并非折扣越大使用率越高**：高折扣(0.0-0.5)使用率仅5.08%
3. **U型曲线特征**：中等力度折扣使用率反而不如中低和低折扣

可能原因：

- **高折扣门槛高**：通常需要满足较高的消费金额
- **信任度问题**：过高的折扣可能让用户怀疑产品质量
- **中等折扣最优**：0.8-0.9的折扣既有吸引力又不会让用户怀疑

6.3 分析二：领取时间对使用率的影响

设计思路伪代码

Mapper伪代码：

```
function Map(key, line):
    解析line得到：Coupon_id, Date_received, Date

    if Coupon_id == null 或 Date_received == null:
        return

    // 解析日期，判断星期几
    date_obj = parse_date(Date_received)
    day_of_week = get_day_of_week(date_obj)

    if day_of_week是周六或周日:
        day_type = "周末"
    else:
        day_type = "工作日"

    if Date != null:
        status = "used"
    else:
        status = "unused"

    emit(day_type, status)
```

Reducer伪代码：

```
function Reduce(day_type, status_list):
    // 与分析一相同，统计使用率
    ...
```

核心代码实现

```java
private SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

@Override
protected void map(LongWritable key, Text value, Context context) {
    String[] fields = line.split(",");

    String dateReceived = fields[5].trim();

    // 解析日期
    Date receivedDate = dateFormat.parse(dateReceived);
    Calendar cal = Calendar.getInstance();
    cal.setTime(receivedDate);

    int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);

    if (dayOfWeek == Calendar.SATURDAY || dayOfWeek == Calendar.SUNDAY) {
        dayType.set("周末");
    } else {
        dayType.set("工作日");
    }

    // 判断使用状态
    if (!"null".equals(date)) {
        usageStatus.set("used");
    } else {
        usageStatus.set("unused");
    }

    context.write(dayType, usageStatus);
}
```

运行结果

```
领取时间	统计结果
```

task4_weekday_result.txt

```
1	周末	领取:211411, 使用:13437, 使用率:6.36%
2	工作日	领取:418413, 使用:31529, 使用率:7.54%
```

结果分析

关键发现：

1. **工作日使用率更高**：7.54% vs 6.36%，差异约1.2个百分点
2. **工作日领取量更大**：约为周末的2倍

可能原因：

- 工作日领取的用户可能有明确的购买计划
- 周末浏览时随手领取，实际使用意愿较弱

- 工作日时间有限，领券后更倾向于快速使用

6.4 分析三：商家发券量对使用率的影响

设计思路

本分析需要**两阶段MapReduce**：

- **第一阶段**：统计每个商家的发券量和使用量
- **第二阶段**：按发券量分类，计算各类商家的整体使用率

第一阶段伪代码

Mapper1伪代码：

```
function Map1(key, line):
    解析line得到：Merchant_id, Coupon_id, Date_received, Date

    if Coupon_id == null 或 Date_received == null:
        return

    if Date != null:
        emit(Merchant_id, "1")  // 已使用
    else:
        emit(Merchant_id, "0")  // 未使用
```

Reducer1伪代码：

```
function Reduce1(Merchant_id, status_list):
    total_count = 0
    used_count = 0

    for each status in status_list:
        total_count += 1
        if status == "1":
            used_count += 1

    emit(Merchant_id, total_count + "," + used_count)
```

第二阶段伪代码

Mapper2伪代码：

```
function Map2(key, line):
    解析line得到：Merchant_id, total_count, used_count

    // 根据发券量分类
    if total_count >= 1000:
        category = "高活跃商家(>=1000券)"
    else if total_count >= 500:
        category = "中高活跃商家(500-999券)"
    else if total_count >= 100:
        category = "中等活跃商家(100-499券)"
```

```
    else if total_count >= 50:
        category = "中低活跃商家(50-99券)"
    else:
        category = "低活跃商家(<50券)"

    emit(category, total_count + "," + used_count)
```

Reducer2伪代码:

```
function Reduce2(category, count_list):
    total_coupons = 0
    used_coupons = 0
    merchant_count = 0

    for each counts in count_list:
        解析得到: total, used
        total_coupons += total
        used_coupons += used
        merchant_count += 1

    usage_rate = (used_coupons / total_coupons) * 100
    avg_coupons = total_coupons / merchant_count

    emit(category, "商家数:" + merchant_count +
                   ", 总发券:" + total_coupons +
                   ", 总使用:" + used_coupons +
                   ", 使用率:" + usage_rate + "%" +
                   ", 平均发券:" + avg_coupons)
```

核心代码实现

第二阶段Mapper核心代码:

```
private String categorizeMerchant(int couponCount) {
    if (couponCount >= 1000) {
        return "高活跃商家(>=1000券)";
    } else if (couponCount >= 500) {
        return "中高活跃商家(500-999券)";
    } else if (couponCount >= 100) {
        return "中等活跃商家(100-499券)";
    } else if (couponCount >= 50) {
        return "中低活跃商家(50-99券)";
    } else {
        return "低活跃商家(<50券)";
    }
}

@Override
protected void map(LongWritable key, Text value, Context context) {
    String[] parts = line.split("\t");
    String[] counts = parts[1].split(",");

    int totalCount = Integer.parseInt(counts[0]);
    int usedCount = Integer.parseInt(counts[1]);

```

```
        String cat = categorizeMerchant(totalCount);
        category.set(cat);
        info.set(totalCount + "," + usedCount);

        context.write(category, info);
    }
```

第二阶段Reducer核心代码：

```
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) {
    long totalCoupons = 0, usedCoupons = 0;
    int merchantCount = 0;

    for (Text value : values) {
        String[] counts = value.toString().split(",");
        totalCoupons += Long.parseLong(counts[0]);
        usedCoupons += Long.parseLong(counts[1]);
        merchantCount++;
    }

    double usageRate = totalCoupons > 0 ?
        (double) usedCoupons / totalCoupons * 100 : 0;
    double avgCoupons = merchantCount > 0 ?
        (double) totalCoupons / merchantCount : 0;

    result.set(String.format(
        "商家数:%d, 总发券:%d, 总使用:%d, 使用率:%.2f%%, 平均发券:%.1f",
        merchantCount, totalCoupons, usedCoupons, usageRate, avgCoupons
    ));

    context.write(key, result);
}
```

运行结果

```
商家活跃度分类                    统计结果
```

task4_merchant_result.txt

```
中低活跃商家(50-99券)    商家数:262, 总发券:17872, 总使用:2959, 使用率:16.56%, 平均发券:68.2
中等活跃商家(100-499券) 商家数:188, 总发券:38831, 总使用:5480, 使用率:14.11%, 平均发券:206.5
中高活跃商家(500-999券) 商家数:33, 总发券:24378, 总使用:3567, 使用率:14.63%, 平均发券:738.7
低活跃商家(<50券)    商家数:4791, 总发券:45428, 总使用:7944, 使用率:17.49%, 平均发券:9.5
高活跃商家(>=1000券)    商家数:73, 总发券:503315, 总使用:25016, 使用率:4.97%, 平均发券:6894.7
```

结果分析

1. 发券数量与商家数

- 低活跃商家占据绝对多数（4791），体现平台长尾结构。
- 高活跃商家数量稀少（73），但发券量巨大。

> 呈现典型的长尾分布结构：少数商家贡献主要资源投放。

2. 使用率对比

- 低活跃商家使用率最高：17.49%

- 中低活跃次之：16.56%
- 中等 / 中高活跃相对接近：≈14%
- 高活跃商家最低：仅 4.97%

> 表明随着发券规模上升，用户边际反应迅速下降。

3. 平均发券差异显著

- 低活跃商家仅 9.5 张/家，但利用率较好
- 高活跃商家平均 6894.7 张/家，存在明显过度投放迹象

> 反映高活跃商家在粗放式营销中可能遭遇效率瓶颈。

现象解释

（1）高活跃商家使用率低

- 发券过多导致用户资格泛化、缺乏稀缺性
- 店铺本身用户池有限

这属于供给过剩导致的转化率下滑。

（2）低活跃商家使用率高

- 用户更稀缺的优惠券有“珍惜感”
- 场景更明确、定位更明确
- 发券数量有限，能够精确触达忠实的顾客

（3）中段发券区间表现稳定

50–999 券区间商家表现较为均衡：

- 使用率 ~14–16%
- 增长较为平稳

6.5 任务四可能的改进之处

1. **考虑多因素交叉分析**：
    - 分析折扣率+领取时间的组合效果
    - 分析商家规模+折扣率的交互影响
2. **可以考虑机器学习**：
    - 使用逻辑回归预测优惠券使用概率
    - 特征工程：提取更多特征（用户活跃度、历史使用率等）
3. **考虑加入时间序列分析**：
    - 分析不同月份、季节的使用率变化
    - 研究节假日效应