

# 实验 4 报告

## 一、实验过程

### (一) 第一题

分别编写 MapReduce 程序和 Spark 程序统计双十一最热门的商品和最受年轻人(age<30)关注的商家 (“添加购物车+购买+添加收藏夹”前 100 名)

#### 1. MapReduce

##### (1) 最热门的商品

代码:

```
mr1/lab4_mr_1
```

运行命令:

先将输入文件放入 hdfs://user/<username>/input 文件夹下,再运行 hadoop jar task1-1.jar

输出:

```
mr1/output_mr1_2
1: 631714,      6205
2: 67897,       5850
3: 15207,       4490
4: 783997,      4316
5: 1059899,     3684
6: 186456,      3678
7: 191499,      3502
8: 822352,      3499
9: 454937,      3351
10: 441588,     3285
11: 159310,     3173
12: 353560,     3125
13: 655904,     2978
14: 764906,     2732
15: 195714,     2606
16: 668220,     2600
17: 1073970,    2541
18: 649596,     2493
19: 951042,     2489
20: 489523,     2474
21: 1039919,    2446
```

思路:

该题较为简单,之前作业出现过。分成两个 mapreduce 过程,第一个阶段统计商品出现的次数,输出一个词频统计的文件 output\_mr1\_1,第二个阶段对该输出进行排序,输出结果 output\_mr1\_2。

##### (2) 最受年轻人(age<30)关注的商家

代码:

```
mr2/lab4_mr_2
```

运行命令：

先将输入文件放入 `hdfs://user/<username>/input` 文件夹下，再运行 `hadoop jar task1-2.jar`

输出：

mr2/output_mr2_3	
1: 4044,	26050
2: 1102,	21416
3: 3760,	19265
4: 3828,	18963
5: 3698,	17976
6: 4173,	17490
7: 1393,	16234
8: 3491,	14430
9: 4976,	11845
10: 606,	11387
11: 173,	8889
12: 3677,	8478
13: 2385,	8261
14: 798,	7771
15: 1257,	7479
16: 184,	7124
17: 4538,	6996
18: 4752,	6930
19: 4282,	6852
20: 1480,	6808
21: 1663,	6681
22: 3734,	6677
23: 4798,	6568
24: 4160,	6498
25: 141,	6460
26: 1043,	6272
27: 4760,	6211
28: 4945,	5978
29: 2206,	5948

思路：

分为三个 `mapreduce`。

第一个阶段合并 `user_log` 和 `user_info`，Mapper 从文件名判断输入的来源，并根据来源在 `value` 前加 `#` 或 `$` 来区分，发送给 Reducer

```

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    FileSplit fileSplit = (FileSplit) context.getInputSplit();
    String name = fileSplit.getPath().getName();

    String line = value.toString();
    if (line == null || line.equals(""))
        return;

    String[] split = line.split(",", -1);

    if (name.contains("user_log")) {
        String user = split[0];
        String item = split[1];
        String cat = split[2];
        String seller = split[3];
        String brand = split[4];
        String time = split[5];
        String action = split[6];
        context.write(new Text(user),
            new Text("#" + item + "," + cat + "," + seller + "," + brand + "," + time + "," + action));
    } else if (name.contains("user_info")) {
        String user = split[0];
        String age = split[1];
        String gender = split[2];
        context.write(new Text(user), new Text("$" + age + "," + gender));
    }
}

```

Reducer 再根据 value 前的标记，将记录分别放在两个 List 中，然后循环两个 List 将记录拼接在一起

```

protected void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {
    List<String> list1 = new LinkedList<>();
    List<String> list2 = new LinkedList<>();

    for (Text text : values) {
        String value = text.toString();
        if (value.startsWith("#")) {
            value = value.substring(1);
            list1.add(value);
        } else if (value.startsWith("$")) {
            value = value.substring(1);
            list2.add(value);
        }
    }

    for (String a : list1) {
        for (String b : list2) {
            context.write(key, new Text(a + "," + b));
        }
    }
}

```

后面两个阶段和上一题相似。

## 2. Spark

(1) 最热门的商品

代码：

sp1/lab4\_sp\_1

运行命令：

```
spark-submit --class "TopCommodity" topcommodity_2.12-1.0.jar  
hdfs://localhost:9000/user/lzt/input/user_log_format1.csv
```

输出：

sp1/output\_sp1

---

```
(1,631714: 6205)  
(2,67897: 5850)  
(3,15207: 4490)  
(4,783997: 4316)  
(5,1059899: 3684)  
(6,186456: 3678)  
(7,191499: 3502)  
(8,822352: 3499)  
(9,454937: 3351)  
(10,441588: 3285)  
(11,159310: 3173)  
(12,353560: 3125)  
(13,655904: 2978)  
(14,764906: 2732)  
(15,195714: 2606)  
(16,668220: 2600)  
(17,1073970: 2541)  
(18,649596: 2493)  
(19,951042: 2489)  
(20,489523: 2474)  
(21,1039919: 2446)  
(22,514725: 2357)  
(23,81550: 2350)  
(24,209821: 2276)  
(25,221663: 2260)  
(26,678194: 2251)  
(27,843827: 2232)  
(28,28805: 2227)
```

思路：

和 MapReduce 一样，而且用 scala 的匿名函数可以让代码简单很多。代码长度跟 Mapreduce 一个 Mapper 的长度差不多。

```

object TopCommodity {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: <file>")
      System.exit(1)
    }

    val conf = new SparkConf().setAppName("Scala_TopCommodity")
    val sc = new SparkContext(conf)
    val file = sc.textFile(args(0))
    val file_sorted = file.map(line => line.split(",", -1))
                                .filter(line => line(6) != "0" && line(6) != "")
                                .map(line => (line(1), 1))
                                .reduceByKey(_ + _)
                                .sortBy(_._2)

    val result = file_sorted.zipWithIndex()
                                .map(line => (line._1, line._2 + 1))
                                .map(line => (line._2, line._1._1 + ": " + String.valueOf(line._1._2)))
                                .take(100)

    val rdd_res = sc.parallelize(result, 1)
    rdd_res.saveAsTextFile("output_sp1")

    sc.stop()
  }
}

```

(2) 最受年轻人(age<30)关注的商家

代码:

sp2/lab4\_sp\_2

运行命令:

```

spark-submit --class "TopSeller" topseller_2.12-1.0.jar
hdfs://localhost:9000/user/lzt/input/user_log_format1.csv
hdfs://localhost:9000/user/lzt/input/user_info_format1.csv

```

输出:

sp2/output\_sp2\_

---

```
(1,4044: 26050)|
(2,1102: 21416)
(3,3760: 19265)
(4,3828: 18963)
(5,3698: 17976)
(6,4173: 17490)
(7,1393: 16234)
(8,3491: 14430)
(9,4976: 11845)
(10,606: 11387)
(11,173: 8889)
(12,3677: 8478)
(13,2385: 8261)
(14,798: 7771)
(15,1257: 7479)
(16,184: 7124)
(17,4538: 6996)
(18,4752: 6930)
(19,4282: 6852)
(20,1480: 6808)
(21,1663: 6681)
(22,3734: 6677)
(23,4798: 6568)
(24,4160: 6498)
(25,141: 6460)
(26,1043: 6272)
(27,4760: 6211)
(28,4945: 5978)
(29,2206: 5948)
```

思路:

思路也和 MapReduce 一样。之前的难点在于合并两个表，而 Scala 的 join 函数可以简单地做到这一点。

```
val file1_pair = file1.map(line => (line.split(", ", -1)(0), line.split(", ", -1)(3), line.split(", ", -1)(6)))
    .filter(line => line._3 != "0")
    .map(line => (line._1, line._2))
val file2_pair = file2.map(line => (line.split(", ", -1)(0), line.split(", ", -1)(1)))
val file_join = file1_pair.join(file2_pair)
```

## (二) 第二题

编写 Spark 程序统计双十一购买了商品的男女比例，以及购买了商品的买家年龄段的比例

### 1. 男女比例

代码:

sp3/lab4\_sp\_3

运行命令:

```
spark-submit --class "MF_Proportion" mf_proportion_2.12-1.0.jar
hdfs://localhost:9000/user/lzt/input/user_log_format1.csv
hdfs://localhost:9000/user/lzt/input/user_info_format1.csv
```

输出:

在终端输出结果

```
*****输出结果*****
男性占总人数比例：0.2987
女性占总人数比例：0.7013
*****输出结果*****
```

思路：

先用 map 将两个表的 Rdd 转化为 pairRdd，筛选出购买的记录，再去掉重复的记录（在第一题我认为重复的购买和加购可以表明热门程度，而这一题统计男女比例就需要去重了）。

```
val file1 = sc.textFile(args(0))
    .map(line => (line.split(",",-1)(0), line.split(",",-1)(6)))
    .filter(line => line._2 == "2")
    .distinct()

val file2 = sc.textFile(args(1))
    .map(line => (line.split(",",-1)(0), line.split(",",-1)(2)))
    .filter(line => line._2 == "0" || line._2 == "1")
    .distinct()
```

然后合并两表，使用 countByValue 可以简单得出数量，除以总数得到比例。

```
val file_join = file1.join(file2)
    .flatMap(line => line._2._2)
    .countByValue()

var sum = 0.0
for (value <- file_join.values) {
    sum = sum + value
}

println("*****输出结果*****")
print("男性占总人数比例：")
println((file_join('1') / sum).formatted("%.4f"))
print("女性占总人数比例：")
println((file_join('0') / sum).formatted("%.4f"))
println("*****输出结果*****")
```

## 2. 年龄段比例

代码：

sp4/lab4\_sp\_4

运行命令：

```
spark-submit --class "Age_Proportion" age_proportion_2.12-1.0.jar
hdfs://localhost:9000/user/lzt/input/user_log_format1.csv
hdfs://localhost:9000/user/lzt/input/user_info_format1.csv
```

输出：



在终端输出结果

```
*****输出结果*****
<18岁占总人数比例：0.0001
[18,24]占总人数比例：0.1607
[25,29]占总人数比例：0.3393
[30,34]占总人数比例：0.2431
[35,39]占总人数比例：0.1239
[40,49]占总人数比例：0.1078
>=50岁占总人数比例：0.0251
*****输出结果*****
```

思路：

与男女比例类似，不再赘述。

### （三）第三题

基于 Hive 或者 Spark SQL 查询双十一购买了商品的男女比例，以及购买了商品的买家年龄段的比例

#### 1. 男女比例

代码：

```
sql1/lab4_sql_1
```

运行命令：

```
spark-submit --class "SparkSQL_MF_Proportion"
sparksq1_mf_proportion_2.12-1.0.jar
hdfs://localhost:9000/user/lzt/input/user_log_format1.csv
hdfs://localhost:9000/user/lzt/input/user_info_format1.csv
```

输出：

在终端输出结果

```
+-----+-----+
|gender|Proportion|
+-----+-----+
|      0|      0.7013|
|      1|      0.2987|
+-----+-----+
```

思路：

我没有在 spark-shell 中完成，是写在程序里的。需要注意的就是在用 SQL 语言操作数据时，需要先将数据集用 createOrReplaceTempView 创建视图。实现的思路和上面是一致的，都是筛选、去重、计算。



```

val info = spark.read.format("csv").option("header", "true")
    .load(args(1))

val log = spark.read.format("csv").option("header", "true")
    .load(args(0))

log.createOrReplaceTempView("df1")
var df1 = spark.sql("SELECT user_id,action_type FROM df1 WHERE action_type = 2")

df1.createOrReplaceTempView("df1")
df1 = spark.sql("SELECT DISTINCT user_id,action_type FROM df1")

info.createOrReplaceTempView("df2")
var df2 = spark.sql("SELECT user_id,gender FROM df2 WHERE gender = 0 or gender = 1")

df2.createOrReplaceTempView("df2")
df2 = spark.sql("SELECT DISTINCT user_id,gender FROM df2")

df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")
var df_join = spark.sql("SELECT * FROM df1 INNER JOIN df2 on df1.user_id=df2.user_id")

df_join.createOrReplaceTempView("df1")
var res = spark.sql("SELECT df1.gender, round(count(*)/(SELECT count(df1.gender) AS count FROM df1), 4) AS Proportion FROM df1 GROUP
res.show()

```

## 2. 年龄段比例

代码：

sql2/lab4\_sql\_2

运行命令：

spark-submit --class "SparkSQL\_Age\_Proportion"

sparksql\_age\_proportion\_2.12-1.0.jar

hdfs://localhost:9000/user/lzt/input/user\_log\_format1.csv

hdfs://localhost:9000/user/lzt/input/user\_info\_format1.csv

输出：

age_range	Proportion
7	0.0212
3	0.3393
8	0.0038
5	0.1239
6	0.1078
1	1.0E-4
4	0.2431
2	0.1607

思路：

和上面是一样的，下面是主要程序

```

val info = spark.read.format("csv").option("header", "true")
    .load(args(1))

val log = spark.read.format("csv").option("header", "true")
    .load(args(0))

log.createOrReplaceTempView("df1")
var df1 = spark.sql("SELECT user_id,action_type FROM df1 WHERE action_type = 2")

df1.createOrReplaceTempView("df1")
df1 = spark.sql("SELECT DISTINCT user_id,action_type FROM df1")

info.createOrReplaceTempView("df2")
var df2 = spark.sql("SELECT user_id,age_range FROM df2 WHERE age_range = 1 or age_range = 2 or age_range = 3 or age_range = 4 or age_range = 5")

df2.createOrReplaceTempView("df2")
df2 = spark.sql("SELECT DISTINCT user_id,age_range FROM df2")

df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")
var df_join = spark.sql("SELECT * FROM df1 INNER JOIN df2 on df1.user_id=df2.user_id")

df_join.createOrReplaceTempView("df1")
var res = spark.sql("SELECT df1.age_range, round(Count(*)/(SELECT count(df1.age_range) AS count FROM df1), 4) AS Proportion FROM df1")
res.show()

```

#### （四）第四题

预测给定的商家中，哪些新消费者在未来会成为忠实客户，即需要预测这些新消费者在 6 个月内再次购买的概率。基于 Spark MLlib 编写程序预测回头客，评估实验结果的准确率。

说明：

该题我分为两部分完成。首先我使用 Spark MLlib 编写了一个简单的程序，因为在 spark 中对 dataframe 的操作不熟悉，花费了很长的时间，还是未能构建较复杂的特征，因此实现的结果得分很低。这也导致我本次实验提交时间很晚。

然后我直接用 python 对该题构造特征，用 lightGBM 进行预测，效果较好，在天池评测得分 0.6914395，当天还未显示排名，应该在 29 名。



**日期:** 2020-12-27 18:42:53 **排名:** 无

**score:** 0.6914395

23	GuoQi	电子科技大学	0.692345	2019-06-23
24	buzhidao	河海大学	0.692086	2017-08-22
25	xupengrun	csi	0.692071	2019-01-22
26	xdwayne	复旦大学	0.691810	2020-12-02
27	CharlesOu	Oyfs	0.691565	2018-09-11
28	yellow_flash	Others-IIT(ISM) dhanbad, I...	0.691482	2020-05-26
29	杉崎键2233	暨南大学	0.691225	2020-12-24
30	sixnu	无	0.690826	2020-12-06
31	feifeiawuwuwu	东华理工大学	0.690673	2020-12-24
32	banfenqian	京东	0.690380	2018-12-29
33	Conggo	开闾海	0.690333	2018-11-09
34	科近邻	无	0.690308	2020-12-19
35	邪恶的小恐龙	南京大学	0.690239	2020-12-23
36 <sup>19</sup>	名称未设定_li	[]	0.689962	2020-12-27

代码：（为了方便运行和调试，我使用 pyspark 和 jupyter notebook）

MLlib/MLlib.ipynb

竞赛/特征.ipynb, 训练、预测.ipynb

输出：

MLlib/lr\_res\_df, svm\_res\_df

竞赛/submission\_lgb.csv

思路：

在 MLlib 部分，我构造了 5 个特征：

①用户年龄、性别

```
# 添加年龄、性别
df_train = df_train.join(user_info, ["user_id"], "left")
df_train.limit(5).show()
```

user_id	merchant_id	label	age_range	gender
34176	3906	0	6	0
34176	121	0	6	0
34176	4356	1	6	0
34176	2217	0	6	0
230784	4818	0	0	0

②log\_count 为某用户在某店加入购物车、购买、收藏的总数

```
# log_count为某用户在某店加入购物车、购买、收藏的总数
user_log_exceptClick = user_log[user_log["action_type"] != 0]
log_count = user_log_exceptClick.groupby(["user_id", "seller_id"]).count()
log_count = log_count.withColumnRenamed("seller_id", "merchant_id").withColumnRenamed("count", "log_count")
df_train = df_train.join(log_count, ["user_id", "merchant_id"], "left")
df_train.limit(5).show()
```

user_id	merchant_id	label	age_range	gender	log_count
464	4718	0	6	0	1
867	3152	0	3	0	1
1882	4377	0	6	1	1
2450	2760	0	0	0	2
2766	3885	0	4	1	1

③cart\_count 为某用户在某店加入购物车的总数

```
# cart_count为某用户在某店加入购物车的总数
user_log_cart = user_log[user_log["action_type"] == 1]
cart_count = user_log_cart.groupby(["user_id", "seller_id"]).count()
cart_count = cart_count.withColumnRenamed("seller_id", "merchant_id").withColumnRenamed("count", "cart_count")
df_train = df_train.join(cart_count, ["user_id", "merchant_id"], "left")
df_train.limit(5).show()
```

user_id	merchant_id	label	age_range	gender	log_count	cart_count
464	4718	0	6	0	1	null
867	3152	0	3	0	1	null
1882	4377	0	6	1	1	null
2450	2760	0	0	0	2	null
2766	3885	0	4	1	1	null

④buy\_count 为某用户在某店购买的总数

```
# buy_count为某用户在某店购买的总数
user_log_buy = user_log[user_log["action_type"] == 2]
buy_count = user_log_buy.groupby(["user_id", "seller_id"]).count()
buy_count = buy_count.withColumnRenamed("seller_id", "merchant_id").withColumnRenamed("count", "buy_count")
df_train = df_train.join(buy_count, ["user_id", "merchant_id"], "left")
df_train.limit(5).show()
```

user_id	merchant_id	label	age_range	gender	log_count	cart_count	buy_count
464	4718	0	6	0	1	null	1
867	3152	0	3	0	1	null	1
1882	4377	0	6	1	1	null	1
2450	2760	0	0	0	2	null	2
2766	3885	0	4	1	1	null	1

⑤collect\_count 为某用户在某店收藏的总数

```
# collect_count为某用户在某店收藏的总数
user_log_collect = user_log[user_log["action_type"] == 3]
collect_count = user_log_collect.groupby(["user_id", "seller_id"]).count()
collect_count = collect_count.withColumnRenamed("seller_id", "merchant_id").withColumnRenamed("count", "collect_count")
df_train = df_train.join(collect_count, ["user_id", "merchant_id"], "left")
df_train.limit(5).show()
```

user_id	merchant_id	label	age_range	gender	log_count	cart_count	buy_count	collect_count
464	4718	0	6	0	1	null	1	null
867	3152	0	3	0	1	null	1	null
1882	4377	0	6	1	1	null	1	null
2450	2760	0	0	0	2	null	2	null
2766	3885	0	4	1	1	null	1	null

得到特征数据后，我采用逻辑回归和支持向量机模型，代码参考官网和其他示例。

下面是逻辑回归的训练结果，测试误差 0.061263329029790085

```

# 逻辑回归，参考官方代码
# Run training algorithm to build the model
lr_model = LogisticRegressionWithLBFGS.train(training)

# Compute raw scores on the test set
predictionAndLabels1 = test.map(lambda lp: (float(lr_model.predict(lp.features)), lp.label))

# Instantiate metrics object
metrics = BinaryClassificationMetrics(predictionAndLabels1)

# Area under precision-recall curve
print("Area under PR = %s" % metrics.areaUnderPR)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)

# Evaluating the model on test data
test_error = predictionAndLabels1.filter(lambda lp: lp[0] != lp[1]).count() / float(test.count())
print("Test Error = " + str(test_error))

Area under PR = 0.09310795813683474
Area under ROC = 0.50004247074441
Test Error = 0.061263329029790085

```

SVM，测试误差为 0.061205948453115284

```

# SVM
# Build the model
from pyspark.mllib.classification import SVMWithSGD, SVMModel
svm_model = SVMWithSGD.train(training, iterations=100)

# Evaluating the model on test data
predictionAndLabels2 = test.map(lambda lp: (lp.label, svm_model.predict(lp.features)))
test_error = predictionAndLabels2.filter(lambda lp: lp[0] != lp[1]).count() / float(test.count())
print("Test Error = " + str(test_error))

Test Error = 0.061205948453115284

```

但是放到官网上评测分数都很低，一个是 0.5001877，还有一个是 0.5。这是由于我的特征选取太少，而且模型不够好。因为 spark 中的 dataframe 和 pandas 中的 dataframe 操作有很大的不同，很多特征我用 pyspark 都无法实现。

后来我直接用 pandas 处理数据，用 lightGBM 来预测，效果较好，得分为 0.6914395，下面说一下实现的思路。

我先是搜索相关的资料，搜到了一篇排名比较高的文章 <https://zhuanlan.zhihu.com/p/137905297>，作者 kagging 在天池的得分为 0.685928，排名 75。在看了他的思路之后，我先是构建了以下几个特征：

### 1. 用户特征

- ① 点击、加购、购买、收藏总记录数：u1
- ② 用户有操作记录的 item\_id 的个数（去重后）：u2
- ③ 用户有操作记录的 cat\_id 的个数（去重后）：u3
- ④ 用户有操作记录的 merchant\_id 的个数（去重后）：u4
- ⑤ 用户有操作记录的 brand\_id 的个数（去重后）：u5
- ⑥ 用户有操作记录的最近时间和最晚时间的间隔天数：u6
- ⑦ 点击、加购、购买、收藏分别记录数：u7、u8、u9、u10



```

# 点击、加购、购买、收藏总记录数: u1
temp = groups.size().reset_index().rename(columns={0:'u1'})
matrix = matrix.merge(temp, on='user_id', how='left')

# 用户有操作记录的item_id的个数(去重后): u2
temp = groups['item_id'].agg(['u2', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='user_id', how='left')

# 用户有操作记录的cat_id的个数(去重后): u3
temp = groups['cat_id'].agg(['u3', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='user_id', how='left')

# 用户有操作记录的merchant_id的个数(去重后): u4
temp = groups['merchant_id'].agg(['u4', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='user_id', how='left')

# 用户有操作记录的brand_id的个数(去重后): u5
temp = groups['brand_id'].agg(['u5', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='user_id', how='left')

# 用户有操作记录的最近时间和最晚时间的间隔天数: u6
temp = groups['time_stamp'].agg(['F_time', 'min'), ('L_time', 'max')]).reset_index()
temp['u6'] = (temp['L_time'] - temp['F_time']).dt.days
matrix = matrix.merge(temp[['user_id', 'u6']], on='user_id', how='left')

# 点击、加购、购买、收藏分别记录数: u7、u8、u9、u10
temp = groups['action_type'].value_counts().unstack().reset_index().rename(columns={0:'u7', 1:'u8', 2:'u9', 3:'u10'})
matrix = matrix.merge(temp, on='user_id', how='left')

```

## 2. 商店特征

- ①商店被点击、加购、购买、收藏总记录数: m1
- ②商店有操作记录的 user\_id 的个数(去重后): m2
- ③商店有操作记录的 item\_id 的个数(去重后): m3
- ④商店有操作记录的 cat\_id 的个数(去重后): m4
- ⑤商店有操作记录的 brand\_id 的个数(去重后): m5
- ⑥商店被点击、加购、购买、收藏各自的记录数: m6、m7、m8、m9

```

# 商店被点击、加购、购买、收藏总记录数: m1
temp = groups.size().reset_index().rename(columns={0:'m1'})
matrix = matrix.merge(temp, on='merchant_id', how='left')

# 商店有操作记录的user_id的个数(去重后): m2
temp = groups['user_id'].agg(['m2', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='merchant_id', how='left')

# 商店有操作记录的item_id的个数(去重后): m3
temp = groups['item_id'].agg(['m3', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='merchant_id', how='left')

# 商店有操作记录的cat_id的个数(去重后): m4
temp = groups['cat_id'].agg(['m4', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='merchant_id', how='left')

# 商店有操作记录的brand_id的个数(去重后): m5
temp = groups['brand_id'].agg(['m5', 'nunique')).reset_index()
matrix = matrix.merge(temp, on='merchant_id', how='left')

# 商店被点击、加购、购买、收藏各自的记录数: m6、m7、m8、m9
temp = groups['action_type'].value_counts().unstack().reset_index().rename(columns={0:'m6', 1:'m7', 2:'m8', 3:'m9'})
matrix = matrix.merge(temp, on='merchant_id', how='left')

```

## 3. 用户-商店特征

- ①用户——商店点击、加购、购买、收藏总记录数: um1
- ②用户——商店有操作记录的 item\_id 的个数(去重后): um2
- ③用户——商店有操作记录的 cat\_id 的个数(去重后): um3
- ④用户——商店有操作记录的 brand\_id 的个数(去重后): um4
- ⑤用户——商店被点击、加购、购买、收藏各自的记录数: um5、um6、um7、um8
- ⑥用户——商店有操作记录的最近时间和最晚时间的间隔天数: um9

```

# 用户—商店点击、加购、购买、收藏总记录数: um1
temp = groups.size().reset_index().rename(columns={0: 'um1'})
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

# 用户—商店有操作记录的item_id的个数(去重后): um2
temp = groups['item_id'].agg(['um2', 'nunique')).reset_index()
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

# 用户—商店有操作记录的cat_id的个数(去重后): um3
temp = groups['cat_id'].agg(['um3', 'nunique')).reset_index()
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

# 用户—商店有操作记录的brand_id的个数(去重后): um4
temp = groups['brand_id'].agg(['um4', 'nunique')).reset_index()
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

# 用户—商店被点击、加购、购买、收藏各自的记录数: um5、um6、um7、um8
temp = groups['action_type'].value_counts().unstack().reset_index().rename(columns={0: 'um5', 1: 'um6', 2: 'um7', 3: 'um8'})
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

# 用户—商店有操作记录的最近时间和最晚时间的间隔天数: um9 |
temp = groups['time_stamp'].agg(['frist', 'min', 'last', 'max')).reset_index()
temp['um9'] = (temp['last'] - temp['frist']).dt.days
temp.drop(['frist', 'last'], axis=1, inplace=True)
matrix = matrix.merge(temp, on=['user_id', 'merchant_id'], how='left')

```

训练后进行预测，在天池上得分大概只有 0.65，根本达不到作者说的 0.685928，我估计是作者出于私心，没有写一些重要的特征。

通过我的思考，又添加了如下若干个特征：（缺失值填 0，性别缺失填 2）

## 1. 用户特征：

### ①每个用户在 5-11 月内的购买数量

解释：用户的购买数量和他是否会复购有极大的关联，而且时间的不同也很重要，有理由认为，靠近 11 月的购买量越多，说明他的生活条件越好，或是购物欲望越强烈，复购的可能性越大。

```

# 每个用户在5-11月内的购买数量
buy_11 = user_log_data[(user_log_data['month'] == "11") & (user_log_data['action_type'] == 2)]
buy_10 = user_log_data[(user_log_data['month'] == "10") & (user_log_data['action_type'] == 2)]
buy_9 = user_log_data[(user_log_data['month'] == "09") & (user_log_data['action_type'] == 2)]
buy_8 = user_log_data[(user_log_data['month'] == "08") & (user_log_data['action_type'] == 2)]
buy_7 = user_log_data[(user_log_data['month'] == "07") & (user_log_data['action_type'] == 2)]
buy_6 = user_log_data[(user_log_data['month'] == "06") & (user_log_data['action_type'] == 2)]
buy_5 = user_log_data[(user_log_data['month'] == "05") & (user_log_data['action_type'] == 2)]

user_buy_11 = buy_11.groupby('user_id', as_index=False)['month'].agg({'user_buy_11': 'count'}).fillna(0)
user_buy_10 = buy_10.groupby('user_id', as_index=False)['month'].agg({'user_buy_10': 'count'}).fillna(0)
user_buy_9 = buy_9.groupby('user_id', as_index=False)['month'].agg({'user_buy_9': 'count'}).fillna(0)
user_buy_8 = buy_8.groupby('user_id', as_index=False)['month'].agg({'user_buy_8': 'count'}).fillna(0)
user_buy_7 = buy_7.groupby('user_id', as_index=False)['month'].agg({'user_buy_7': 'count'}).fillna(0)
user_buy_6 = buy_6.groupby('user_id', as_index=False)['month'].agg({'user_buy_6': 'count'}).fillna(0)
user_buy_5 = buy_5.groupby('user_id', as_index=False)['month'].agg({'user_buy_5': 'count'}).fillna(0)

matrix = matrix.merge(user_buy_11, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_10, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_9, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_8, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_7, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_6, on=['user_id'], how='left')
matrix = matrix.merge(user_buy_5, on=['user_id'], how='left')

```

### ②购买——点击比

解释：这说明用户的习惯，比值越大，越容易消费。

```
matrix['r1'] = matrix['u9']/matrix['u7'] # 用户购买点击比
```

## 2. 商店特征：



### ①每个商店在 5-11 月内的用户购买数量

解释：和上面类似，如果一家店最近的销量好，说明它的经营状况好，或是商品更有吸引力。

```
# 每个商店在5-11月内的购买数量
merchant_buy_11 = buy_11.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_11': 'count'}).fillna(0)
merchant_buy_10 = buy_10.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_10': 'count'}).fillna(0)
merchant_buy_9 = buy_9.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_9': 'count'}).fillna(0)
merchant_buy_8 = buy_8.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_8': 'count'}).fillna(0)
merchant_buy_7 = buy_7.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_7': 'count'}).fillna(0)
merchant_buy_6 = buy_6.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_6': 'count'}).fillna(0)
merchant_buy_5 = buy_5.groupby('merchant_id', as_index=False)['month'].agg({'merchant_buy_5': 'count'}).fillna(0)

matrix = matrix.merge(merchant_buy_11, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_10, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_9, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_8, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_7, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_6, on=['merchant_id'], how='left')
matrix = matrix.merge(merchant_buy_5, on=['merchant_id'], how='left')
```

### ②每个商店的复购次数

解释：这一点很重要，如果一家店的复购次数大，说明这家店很容易吸引回头客。

```
# 每个商店的复购次数
temp = user_log_data[user_log_data['action_type'] == 2]
temp = temp.drop_duplicates(subset=['merchant_id', 'time_stamp', 'user_id'])
temp = temp.groupby('merchant_id').apply(lambda x: x['user_id'].count() - x['user_id'].nunique())
temp = temp.reset_index().rename(columns={'repeat_count'})
matrix = matrix.merge(temp, on='merchant_id', how='left')
```

### ③购买——点击比

解释：说明了商店的吸引力

```
matrix['r2'] = matrix['m8']/matrix['m6'] # 商家购买点击比
```

## 3. 用户——商店特征

### ①每个用户——商店在 5-11 月内的购买数量

解释：和上面类似。

```
# 每个用户——商店在5-11月内的购买数量
user_merchant_buy_11 = buy_11.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_11': 'count'})
.fillna(0)
user_merchant_buy_10 = buy_10.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_10': 'count'})
.fillna(0)
user_merchant_buy_9 = buy_9.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_9': 'count'}).fillna(0)
user_merchant_buy_8 = buy_8.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_8': 'count'}).fillna(0)
user_merchant_buy_7 = buy_7.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_7': 'count'}).fillna(0)
user_merchant_buy_6 = buy_6.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_6': 'count'}).fillna(0)
user_merchant_buy_5 = buy_5.groupby(['user_id', 'merchant_id', as_index=False)['month'].agg({'user_merchant_buy_5': 'count'}).fillna(0)

matrix = matrix.merge(user_merchant_buy_11, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_10, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_9, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_8, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_7, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_6, on=['user_id', 'merchant_id'], how='left')
matrix = matrix.merge(user_merchant_buy_5, on=['user_id', 'merchant_id'], how='left')
```

### ②购买——点击比

解释：和上面类似。

```
matrix['r3'] = matrix['um7']/matrix['um5'] #不同用户不同商家购买点击比
```

得到特征数据后，保存在 train\_data.csv, test\_data.csv 中。然后读取 train\_data.csv，将其分为训练集和测试集，使用 lightgbm 模型训练，最后预测 test\_data.csv 中的概率，得到 submission\_lgb.csv。（这里偷了点小懒，直接用了 Kagging 的参数，发现效果比他做得还好，就没有继续调参了）

```
model_lgb = lgb.LGBMClassifier(  
    max_depth=10,  
    n_estimators=1000,  
    min_child_weight=200,  
    colsample_bytree=0.8,  
    subsample=0.8,  
    eta=0.3,  
    seed=42  
)  
  
model_lgb.fit(  
    X_train[i],  
    y_train[i],  
    eval_metric='auc',  
    eval_set=[(X_train[i], y_train[i]), (X_valid[i], y_valid[i])],  
    verbose=False,  
    early_stopping_rounds=10  
)  
  
print(model_lgb.best_score_['valid_1']['auc'])  
  
pred = model_lgb.predict_proba(test_data)  
pred = pd.DataFrame(pred[:,1])  
pred_lgbms.append(pred)
```

## 二、问题总结

本次实验在配置环境上没有遇到太大的问题，主要是学习 spark 编程耗时间。（上次配好虚拟机确实一劳永逸）

### 1. spark 无法读取 hdfs 文件

解决方法：

编辑 spark-env.sh 文件(vim ./conf/spark-env.sh)，在第一行添加以下：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

有了上面的配置信息以后，Spark 就可以把数据存储到 Hadoop 分布式文件系统 HDFS 中，也可以从 HDFS 中读取数据。如果没有配置上面信息，Spark 就只能读写本地数据，无法读写 HDFS 数据。

### 2. sbt 使用国内镜像下载很慢

解决方法：

这个要用 aliyun 的源，huaweiyun 的还是很慢，不过我一开始的解决方法是

让电脑开了一晚。

### 3. scala 分割遇到缺失值的处理

解决方法：

一开始我用一个数组存储，每次都要判断一下数组的大小来查看是否有缺失值，后来直接使用 `split("", -1)`，可以将缺失值留空，比如 `1,2,,` 这个字符串，如果用 `split(",")` 只会读到 “1 2 ” 两个值，用 `split("", -1)` 会读到 “ 1 2 空 空 ” 四个值。