实验四

姜春妮
181250057

Task1 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:

1.1 最热门商品 hotsell:

1.1.1 设计思路

与 wordcount 思路相似，在 map 阶段写入 context 前加上 “双十一” 和 “操作为 1/2/3” 的条件判断

```
String line = value.toString();
String[] data = line.split(",");
String date = data[5];
String action = data[6];
String product = data[1];
if(date.equals("1111")){
    if(action.equals("1")||action.equals("2")||action.equals("3")){
        context.write(new Text(product), one);
    }
}
```

1.1.2 运行结果

```
排名1:191499,共计2494次
排名2:353560,共计2250次
排名3:1059899,共计1917次
排名4:713695,共计1754次
排名5:655904,共计1674次
排名6:67897,共计1572次
排名7:221663,共计1547次
排名8:1039919,共计1511次
排名9:454937,共计1387次
排名10:81360,共计1361次
排名11:514725,共计1356次
排名12:783997,共计1351次
排名13:823766,共计1343次
排名14:107407,共计1319次
排名15:889095,共计1272次
排名16:936203,共计1270次
排名17:770668,共计1257次
排名18:698879,共计1235次
排名19:349999,共计1218次
排名20:671759,共计1167次
排名21:186456,共计1162次
排名22:315345,共计1067次
排名23:729259,共计1021次
排名24:946001,共计1015次
排名25:181387,共计1002次
排名26:926069,共计1002次
排名27:28895,共计983次
排名28:89953,共计975次
排名29:413046,共计965次
排名30:944554,共计948次
排名31:617878,共计927次
排名32:676215,共计873次
排名33:213297,共计864次
```

1.1.3 遇到的问题

Q1 运行成功，但 part-r-00000 没有东西

```
root@jcn181250057-master:/workspace/TMall/jcn# hadoop fs -cat output/part-r-00000
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so which might have di
sabled stack guard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
20/12/08 11:28:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java class
es where applicable
root@jcn181250057-master:/workspace/TMall/jcn#
```

查看日志，第一个 job 中，map 的输出为空，所以第一个 job 的 map 代码出错

```
    Map-Reduce Framework
            Map input records=54925331
            Map output records=0
            Map output bytes=0
            Map output materialized bytes=90
            Input split bytes=1995
            Combine input records=0
            Combine output records=0
            Reduce input groups=0
            Reduce shuffle bytes=90
            Reduce input records=0
            Reduce output records=0
            Spilled Records=0
```

查资料发现，java 中对于 float, int, char, boolean 等数据类型的变量，变量直接存储的是“值”，因此在用关系操作符==来比较的时候，比较的是值本身。然而，对于非基本数据类型的变量，也就是引用类型的变量，变量存储的是对象在内存中的地址。因此，我在 action 这个 string 和“1”的比较中，直接使用==，将筛选不到任何数据，所以 map 的 output 是 0

Q2 命令行创建 maven

```
[INFO] ------------------------------------------------------------------------
[ERROR] No plugin found for prefix 'archetype' in the current project and in the plugin groups [org.apac
he.maven.plugins, org.codehaus.mojo] available from the repositories [local (/root/.m2/repository), cent
ral (http://repo.maven.apache.org/maven2)] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles
```

原因在于 maven 需要换源，参考这个网址：

https://blog.csdn.net/weixin_40992982/article/details/104087472?utm_medium=distribute.pc_relevant_bbs_down.none-task--2~all~first_rank_v2~rank_v29-1.nonecase&depth_1-utm_source=distribute.pc_relevant_bbs_down.none-task--2~all~first_rank_v2~rank_v29-1.nonecase

加入：

```
<mirrors>
    <mirror>
        <id>alimaven</id>
        <name>aliyun maven</name>
```

```
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
    <mirror>
        <id>UK</id>
        <name>UK Central</name>
        <url>http://uk.maven.org/maven2</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
    <mirror>
        <id>ibiblio.org</id>
        <name>ibiblio Mirror of http://repo1.maven.org/maven2/</name>
        <url>http://mirrors.ibiblio.org/pub/mirrors/maven2</url>
        <mirrorOf>central</mirrorOf>
        <!-- United States, North Carolina -->
    </mirror>
    <mirror>
        <id>jboss-public-repository-group</id>
        <mirrorOf>central</mirrorOf>
        <name>JBoss Public Repository Group</name>
        <url>http://repository.jboss.org/nexus/content/groups/public</url>
    </mirror>
</mirrors>
```

1.2 最受年轻人关注商家 youngsell:

1.2.1 设计思路

类似 wordcount, wordsToParse 在本题情景下等同于符合 “age<30” 这一条件的 user。parseUsersFile 中读取 info，筛选符合条件的年轻用户

```
private void parseUsersFile(String fileName){
    try{
        fis = new BufferedReader(new FileReader(fileName));
        String line = null;
        while((line = fis.readLine())!=null){

            String[] userInfo = line.split(",");
            if(userInfo.length == 3){
                String user = userInfo[0];
                String age = userInfo[1];

            if(age.equals("1")||age.equals("2")||age.equals("3")){
                youngUsers.add(user);
            }
```

setup 中设置读取 info

```
public void setup(Context context) throws IOException,InterruptedException{
    this.conf = context.getConfiguration();
    URI[] patternsURIs = Job.getInstance(this.conf).getCacheFiles();
    URI patternURI = patternsURIs[0];
    Path patternPath = new Path(patternURI.getPath());
    String patternFileName = patternPath.getName().toString();
    parseUsersFile(patternFileName);
}
```

map 写入 context 前加入条件筛选：是年轻用户且在双十一购买

```
//@Override
public void map(Object key, Text value, Context context) throws IOException, Interrupte
    String line = value.toString();
    String[] data = line.split(",");
    String date = data[5];
    String action = data[6];
    String seller = data[3];
    String user = data[0];

    if(youngUsers.contains(user)==true && date.equals("1111")){
        if(action.equals("1")||action.equals("2")||action.equals("3")){
            context.write(new Text(seller), one);
        }
```

1.1.2 运行结果

```
排名1:4044,共计7248次
排名2:3491,共计3634次
排名3:1102,共计3565次
排名4:3828,共计3416次
排名5:4173,共计3333次
排名6:3734,共计3277次
排名7:2385,共计3209次
排名8:4976,共计3048次
排名9:798,共计2988次
排名10:422,共计2874次
排名11:1892,共计2779次
排名12:1393,共计2756次
排名13:4282,共计2713次
排名14:1535,共计2706次
排名15:4760,共计2643次
排名16:3760,共计2595次
排名17:4644,共计2582次
排名18:184,共计2391次
排名19:598,共计2371次
排名20:3698,共计2103次
排名21:4043,共计2070次
排名22:375,共计2061次
排名23:2537,共计1937次
排名24:1760,共计1924次
排名25:2482,共计1910次
排名26:2138,共计1833次
排名27:4659,共计1825次
排名28:606,共计1815次
排名29:1257,共计1771次
排名30:4218,共计1746次
排名31:141,共计1628次
排名32:4538,共计1623次
排名33:4918,共计1588次
排名34:420,共计1577次
排名35:3826,共计1577次
排名36:2031,共计1568次
```

1.2.3 遇到的问题

运行报错，在尝试 print 以及查看 log 都无果后，对可能出现问题的代码块进行逐一注释排查，发现问题在于分割 user 取数组里的每一位时，没有考虑到空值的情况。即如果年龄或性别空缺，那么索引为 2 的 string 是不存在的。解决方案是首先通过 user 长度判断有无缺失值，再进行赋值。

```
String[] userInfo = line.split(",");
if(userInfo.length == 3){
    String user = userInfo[0];
    String age = userInfo[1];
```

1.2.4 问题反思

这一小题中，耗费时间最多的就是空值对结果的影响。这带来的启示是，处理实际问题的过程中，数据往往是不完善、需要再处理的，包括空值、不规范等问题。因此数据的预处理十分重要。

2. Spark

2.1 最热门商品 hotsell:

2.1.1 设计思路

- 依次筛选操作日期、操作
- 做 user_id 和 1 的 MapReduce
- 交换键值对，排序后再交换键值对
- take 语句取前 100 个

```
val conf = new SparkConf().setAppName("Scala_HotSell")
val sc = new SparkContext(conf)
val input = sc.textFile(args(0))

val rdd = input.filter(x=>x.split(",")(5).equals("1111"))

val rdd2 = rdd.filter(x=>x.split(",")(6).equals("0")==false)
val counts = rdd2.map(x=>(x.split(",")(1),1)).reduceByKey(_+_)


val result = counts.map(x=>(x._2,x._1)).sortByKey(false).map(x=>(x._2,x._1)).take(100)
val r = sc.parallelize(result)
r.saveAsTextFile(args(1))
sc.stop()
```

2.1.2 运行结果

同 java 写的 mapreduce 运行结果

2.1.3 遇到的问题

除了 sbt 安装的问题之外，编写 scala 程序时最大的问题在于没有理解 scala 的逻辑结构。举例来说，我把 java 中做 mapreduce 时读取的是一行一行的数据的想法迁移到了 scala 中，从而在第一步操作的时候就产生了许多问题。rdd 操作中，最关键的部分在于 key 和 value 的映射对，理解了这一点后，后续 scala 的编写就很轻松了。

这一问题带来的启示在于，对于一门新的语言，入门的关键不在于语法的细节，而是在于对这门语言底层逻辑结构和关键点的理解，java 的关键在于对象的概念，scala 的关键在于键值对等等。

Task2 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

1.1 设计思路

- 处理 info 中的空缺值，填补 null 为 0 或 2
- 使用 filter 筛选 man 和 woman 用户，分别形成<user_id, 1>的键值对
- 筛选 log 中 “双十一” 购买的用户为 users
- 从 log 中 subtractByKey 去掉 users，剩下的是不符合条件的用户
- man, woman 中分别 subtractByKey 去掉不符合条件的用户，分别形成<man,1>和<woman,1>的键值对
- 作<man,1>和<woman,1>的 reduce

```
val conf = new SparkConf().setAppName("Sex")
val sc = new SparkContext(conf)
val inputLog = sc.textFile(args(0))
val inputInfo = sc.textFile(args(1))

//sex
val infoClean = inputInfo.na.fill(value="0".toInt, Array("gender"))
val infoClean = inputInfo.na.fill(value="2".toInt, Array("age_range"))


val man = infoClean.filter(x=>(x.split(",")(2).equals("1"))).map(x=>(x.split(",")(0),1))
val woman = infoClean.filter(x=>(x.split(",")(2).equals("0"))).map(x=>(x.split(",")(0),1))


val rdd = inputLog.filter(x=>x.split(",")(5).equals("1111"))
val rdd2 = rdd.filter(x=>x.split(",")(6).equals("2")) //purchase
val users = rdd2.map(x=>(x.split(",")(0),1))//<user_id, 1>
val log = inputLog.map(x=>(x.split(",")(0),1))
val notUsers = log.subtractByKey(users)

val manUsers = man.subtractByKey(notUsers).map(x=>("man",1))
val manCounts = manUsers.reduceByKey(_+_).map(x=>x._2)

val womanUsers = woman.subtractByKey(notUsers).map(x=>("woman",1))
val womanCounts = womanUsers.reduceByKey(_+_).map(x=>x._2)
```

这里设计的重点在于，使用 subtractByKey 进行两个表的合并。使用 subtractByKey 的原因在于，题目中需要筛选的条件众多，取非运算会更加方便。

1.2 运行结果

```
man:121670
woman:285638
```

2. 年龄比例

2.1 设计思路

- 处理 info 中的空缺值，填补 null 为 0 或 2

```
val infoClean = inputInfo.na.fill(value="0".toInt, Array("gender"))
val infoClean = inputInfo.na.fill(value="2".toInt, Array("age_range"))
```

- info 中使用 filter 筛选各个年龄段的用户，形成<user_id, 1>键值对

```
val ageLess18 = infoClean.filter(x=>(x.split(",")(1).equals("1"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val age18to24 = infoClean.filter(x=>(x.split(",")(1).equals("2"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val age25to29 = infoClean.filter(x=>(x.split(",")(1).equals("3"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val age30to34 = infoClean.filter(x=>(x.split(",")(1).equals("4"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val age35to39 = infoClean.filter(x=>(x.split(",")(1).equals("5"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val age40to49 = infoClean.filter(x=>(x.split(",")(1).equals("6"))).map(x=>(x.split(",")(0),1))//<user_id, 1>
val ageMore50 = infoClean.filter(x=>(x.split(",")(1).equals("7") || x.split(",")(1).equals("8"))).map(x=>(x.split(",")(0),1))//<user_id, 1
```

- log 中使用 filter 筛选双十一购买的用户（同统计性别中的操作），形成<user_id, 1>键值对

```
val rdd = inputLog.filter(x=>x.split(",")(5).equals("1111"))
val rdd2 = rdd.filter(x=>x.split(",")(6).equals("2"))
val users = rdd2.map(x=>(x.split(",")(0),1))//<user_id, 1>
val log = inputLog.map(x=>(x.split(",")(0),1))
```

- log 中 subtractByKey 实际用户得到不符合条件的用户 notUsers

```
val notUsers = log.subtractByKey(users)
```

- 分别在各年龄段的用户中 subtractByKey 得到该年龄段内符合条件的用户，形成<category, 1>的键值对

```
val ageLess18Users = ageLess18.subtractByKey(notUsers).map(x=>("ageLess18",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate1, 1>
val age18to24Users = age18to24.subtractByKey(notUsers).map(x=>("age18to24",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate2, 1>
val age25to29Users = age25to29.subtractByKey(notUsers).map(x=>("age25to29",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate3, 1>
val age30to34Users = age30to34.subtractByKey(notUsers).map(x=>("age30to34",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate4, 1>
val age35to39Users = age35to39.subtractByKey(notUsers).map(x=>("age35to39",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate5, 1>
val age40to49Users = age40to49.subtractByKey(notUsers).map(x=>("age40to49",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate6, 1>
val ageMore50Users = ageMore50.subtractByKey(notUsers).map(x=>("ageMore50",1)).reduceByKey(_+_).map(x=>x._2)//<user_id, 1> -> <cate7, 1>
```

- 合并输出

2.2 输出结果

```
24
52871
111654
79991
40777
35464
8258
```

Task3 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比例

本题使用 Hive

1. 男女比例

1.1 设计思路

- 创建对应数据格式的表格 Log
- 将数据载入 log 表格
- insert overwrite 写入 log 双十一购买的数据行
- log 中筛选 action=2，创建并写入表格 trueusers，代表计入统计的真实用户信息
- 创建对应数据格式的表格 info，并将数据载入 info 表格
- info 中筛选 gender=1，创建并写入表格 manusers，代表所有男性用户信息
- info 中筛选 gender=0，创建并写入表格 womanusers，代表所有女性用户信息
- select intersect 取 manusers 和 trueusers 的交集，得到符合条件的男性用户
- 女性用户同上
- 对 user_id 进行 count 操作，得到男女人数

1.2 输出结果

同 Task2

1.3 遇到的问题

hive 对于数据的类型不敏感，需要在创建表格时首先规定各数据的类型，如：

```
hive> create table log(
    > user_id int,
    > item_id int,
    > cat_id int,
    > seller_id int,
    > brand_id int,
    > time_stamp string,
    > action_type int)
    > row format delimited
    > fields terminated by ',';
```

2. 年龄段比例

2.1 设计思路（在 1 的基础上操作）

- 从 info 中筛选 age_range，形成各个年龄段的表格
- 各个年龄段的表格分别和 trueusers 表格就 user_id 取交集
- 对各个年龄段表格中的 user_id 作 count 操作，得到计数

2.2 输出结果

同 Task2

Task4 预测给定的商家中，哪些新消费者会在未来成为忠实客户，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 MLib 编写程序预测回头客，评估实验结果的准确率

1. 设计思路

该 task 分为以下几步：

- 数据预处理：空值
- 特征选取：综合 log, info 交叉提取特征，得到包含属性和标签的 train.csv
- 数据特征再处理：建立特征向量、标准化、调节不均衡样本……
- 算法选取：逻辑回归、随机森林……
- 调参
- 测试集带入模型

2. 实现细节

2.1 数据预处理

数据的预处理主要在于 info 中 gender 和 age_range 的空值填充。

这里我想到了几个可行的方案，一是删去有空值的整条数据，二是向前/后填充，三是将空值也划分成新的一类。我最后选择的方案是方案三，即以性别为例，划分成男、女、未知共三类

这样做的原因在于，用户设置性别与否可以一定程度上作为用户画像的一部分，例如，设置性别的用户相对不注重隐私、使用天猫更多等等。

此外，空值的填充不仅出现在数据的预处理中，特征提取时，有些特征也可能为空，例如加入购物车的次数，如果用户从来没有加入购物车，那么他的这一条数据为 null，是不合理的，应该在正式训练前预处理为 0

```
train = train.withColumn("shopping_carts",
    when(train.shopping_carts.isNull(),lit("0")).otherwise(train.shopping_carts))
```

2.2 特征选取

判断用户是否在一家商店回购，取决于三方面：

- 用户与商家的关系
- 用户自身购买习惯
- 商家自身是否能吸引人回购

因此，我将从这三个方面来设计特征：

2.2.1 用户与商家关系

特征	定义	涵义
total_gender	用户在该商家所有日志总数	用户对该商家关注程度
clicks	用户在该商家单击的操作数量	
shopping_carts	用户在该商家添加购物车的操作次数	
purchase_times	用户在该商家购买的操作次数	
favourite_times	用户在该商家收藏的操作次数	
unique_item_times	用户在该商家买过多少个不同的产品	用户对该商家整体兴趣
categories	用户在该商家买过多少个不同的产品类别	
browse_days	用户在该商家浏览过多少天	用户对该商家的长期关注程度
browse_time_interval	用户在该商家每季度浏览次数	用户对该商家关注程度的变化

2.2.2 用户自身偏好

特征	定义	涵义
gender	用户性别	基本信息
age_range	用户年龄	
user_total_log	用户所有日志数量	历史购买频率
user_purchase_log	用户所有购买次数	
user_habit	用户所有日志总数/所有购买次数	
dedicated_on_brand	用户买过的不同的牌子数量之和	对品牌的专注程度
dedicated_on_seller	用户买过的不同商家数量之和	对商家的专注程度
dedicated_on_cat	用户买过的不同类别商品数量之和	对商品类别的专注程度

2.2.3 商家自身能力

特征	定义	涵义
seller_size	商家卖出的商品总数	商家总的卖货能力
old_customer	商家拥有的会回购的客人数量	商家吸引回头客的能力
sell_item	商家售卖的不同商品的个数	商家售卖的商品丰富程度
sell_cate	商家售卖的不同类别商品的个数	
sell_brand	商家售卖的不同品牌商品的个数	

实验发现，在 baseline 的基础上，添加用户自身偏好特征可以将准确率提升 1-2 个百分点，添加商家自身能力特征可以提升 2 个百分点。

这一差距的原因，除了用户偏好的特征选择不佳以外，还在于，用户的差异性较大，而商家的差异性相对较小，特征概括会比较容易且准确。

2.3 数据特征再处理

2.3.1 VectorAssembler

它将给定的列列表组合到一个向量列中，将原始特征和由不同特征变换器生成的特征组合成单个特征向量非常有用，以便训练 ML 模型。

```
train = VectorAssembler(inputCols=feature_colums_train, outputCol="features").transform(train)
test = VectorAssembler(inputCols=feature_colums_test, outputCol="features").transform(test)
```

2.3.2 indexer

将标签索引转换为原始的标签。

```
# indexer
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(train)

featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures",
    maxCategories=5).fit(train)
featureIndexer_test = VectorIndexer(inputCol="features", outputCol="indexedFeatures",
    maxCategories=5).fit(test)


# indexedLabel -> label
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
    labels=labelIndexer.labels)
```

2.3.3 standard scale

若用回归分析算法时，必须将数值特征字段进行标准化，这是因为数值特征字段单位不同，数字差异很大，所以无法彼此比较，这时，就需要使用标准化，使得数值特征字段具有共同的标准。

```
# lr standard scale
standardscaler = StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
train = standardscaler.fit(train).transform(train)
test = standardscaler.fit(test).transform(test)
```

2.3.4 modify unbalance

由于训练集中样本极度不均衡，若不预处理，则会预测所有样本都是占多数的那类样本，这样的准确率虽然很高，但是没有意义。

因此，需要给占比少的样本加大权重，使其在训练中拥有和另一主导样本相似的“影响力”。

```
# modify unbalance
train_size = train.select("label").count()
negative_num = train.select("label").where("label==0").count()
balance_ratio =float(float(negative_num)/float(train_size))
train = train.withColumn("classWeights",when(train.label==1, balance_ratio).otherwise(1-
    balance_ratio))
```

2.4 算法选取

由于原训练集是不均衡样本，因此首先我选择了随机森林，后来尝试了样本加权的逻辑回归。实验发现，逻辑回归的 acc 和 auc 更高，因此选择逻辑回归。

3. 预测效果与超参设置

准确率	0.6543749
算法	样本加权的逻辑回归
最大迭代轮数	50
regParam	0.02

4. 遇到的问题与反思

4.1 不均衡样本

机器学习理论课上讲到，随机森林可以很好地处理不均衡样本，因此我首先选取了随机森林作为算法。

但是在实际操作中，不给样本加权的随机森林效果远远不如加权的逻辑回归，且给随机森林的样本加权后，随机森林的效果有 3%的提升。

我把这次用随机森林的体验和机器学习课上的随机森林实验进行了对比。在机器学习课程的作业中，我用随机森林实现工资的二分类预测，其训练集和测试集也都是不均衡样本。但是，在工资的二分类预测中，随机森林准确率可以达到 80%以上，auc 也可以达到 0.75 以上。也就是说，同样是处理不均衡样本，随机森林在工资这一场景的预测上比回购上表现好很多。

分析原因，我认为有两点。

其一，工资的预测实验中，所有的特征都是已经给出的，不需要自己来提炼。这就避免了差、乃至无效特征对预测结果的干扰。由于随机森林在建立基分类器的时候，特征的选取是随机的，因此如果我的特征里有一些不好的特征，那么对于那些不幸选到这些特征的基分类器来说，他们做出的判断是无效、乃至有反作用的。

其二，预测回购的基础信息有限。以自己网购的想法带入，我的购买习惯、商家的特性等等因素都不是我是否决定回购的关键性指标，给第一次购买的评分和评价是更为重要的反映指标。因此，原本就不算强有效的指标进行再提取，产生有效特征的可能性就更低了，这就又回到了原因的第一点。

因此，回顾随机森林，我认为这个算法对特征的选取要求较高、对差特征的容忍度较低。

4.2 训练集和测试集分布不一致

在使用逻辑回归的时候，我惊喜地发现，regParam 参数越高，在 validation 集上的准确率就越高。然而运行出来的结果在提交时准确率却很低。

后来细想才明白过来，其实这是因为模型随着 regParam 的升高，把越来越多的样本分为占比大的那个标签。如果 regParam 无限升高，那么和全部打成 “不回购” 的标签效果是一样的。

那么，为什么 validation 和真实 test 集上的准确率差异会这么大呢？

看着排行榜一堆的 0.5 准确率，我推测是因为测试集中 0&1 标签是对半分的，也就是说，测试集数据是均衡样本，和 validation，也就是训练集的样本分布不一致。

针对这个问题，我上网找了资料，但基本上都是关于 “如何判断测试集与训练集是否同分布”，没有该问题的解决方法。也就是说，除了机械地把 validation 调成 test 的同分布，无法解决 validation 不起作用的问题，这给调参带来了很大困难。