FBDP第四次实验报告

姓名：席晓阳

学号：171840013

实验步骤及截图

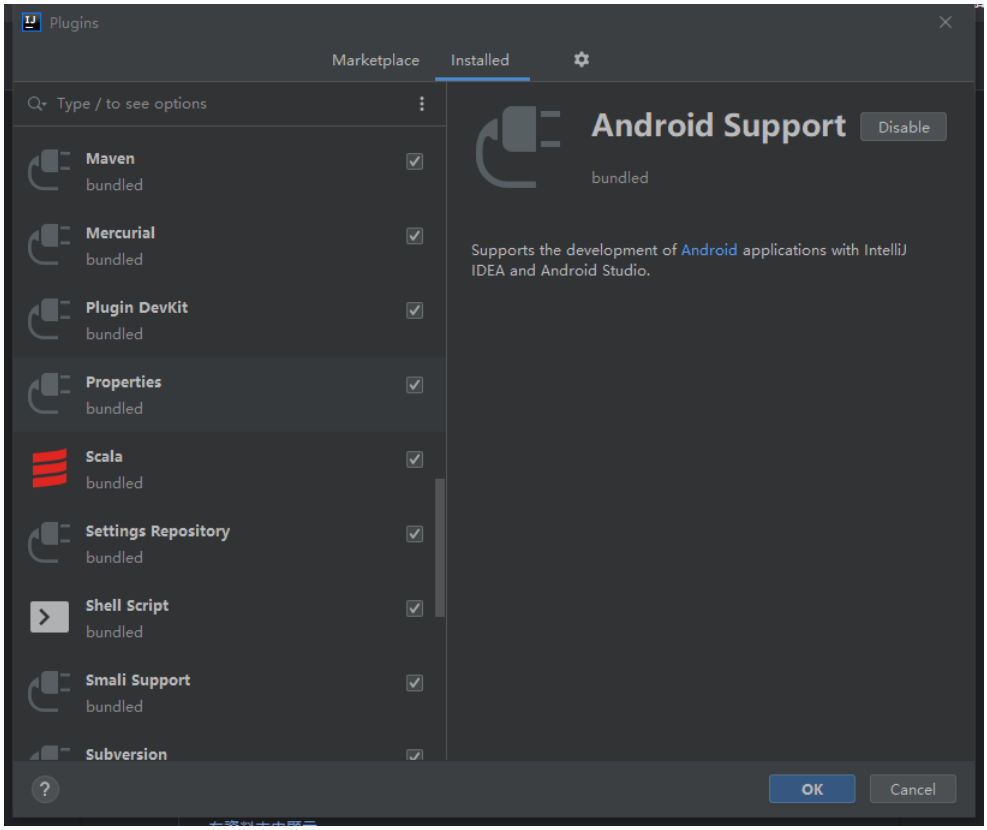
Spark的安装及Intelij IDEA的配置

说明：因为虚拟机性能有所限制，所以直接在本机上配了spark、scala和IntelilJ IDEA

步骤参考：https://blog.csdn.net/xujingpilot/article/details/104322151

Windows下Spark安装成功：

Intelij IDEA中plugins配置完成：



任务一

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

MapReduce思路：共一个mapreduce，类似于wordcount的思路，在setup时读取用户信息表，为减少运行时间，只记录年轻人（age_range=1/2/3）的数据，在map阶段记录时间为“1111”，用户操作不为“点击”的user_id和匹配用户为年轻人的merchant_id，将（word,one）传给reduce，再在reduce中排序并保留前100条记录

核心代码：

```
//建立用户信息表
        @Override
        public void setup(Context context) throws
IOException,InterruptedException{
            Configuration conf = context.getConfiguration();
            localFiles  = conf.getStrings("user_info")[0];
            System.out.println(localFiles);
            task=conf.getInt("task_type",1);
            FileSystem fs = FileSystem.get(URI.create(localFiles), conf);
            FSDataInputStream hdfsInStream = fs.open(new Path(localFiles));
            //从hdfs中读取user_info
            InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-
8");
            String line;
            BufferedReader br = new BufferedReader(isr);
            while ((line = br.readLine()) != null) {
                String[] str=line.split(",");
                String user_id,age_range;
                user_id=str[0];
```

```
                if(str.length==1 || (str.length==2 &&
!line.substring(line.length() - 1).equals(",")))
                    age_range="-1";
                else
                    age_range=str[1];

                // <18岁为1;[18,24]为2;[25,29]为3;[30,34]为4;[35,39]为5;[40,49]为
6;>=50时为7和8;0和NULL表示未知
                // 为了减少运行时间，仅记录年轻人的user_info
                if(age_range.equals("1") || age_range.equals("2") ||
age_range.equals("3"))
                {
                    User u = new User(user_id, age_range);
                    users.add(u);
                }
            }
            System.out.println("Setup Succeed!");
        }

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] str=line.split(",");
            String user_id=str[0];
            String item_id=str[1];
            String cat_id=str[2];
            String merchant_id=str[3];
            String brand_id,time_stamp,action_type;
            if(str.length==7){
                brand_id=str[4];
                time_stamp=str[5];
                action_type=str[6];
            }
            else{
                time_stamp=str[4];
                action_type=str[5];
            }
            if(!time_stamp.equals("1111"))
                return;
            switch (task){
                case 1: //双十一最热门商品
                    if(!action_type.equals("0")){
                        id.set(item_id);
                        System.out.println("item id:"+id+" recorded!");
                        context.write(id, one);
                        return;
                    }
                    break;
                case 2: //最受年轻人(age<30)关注的商家
                    if(!action_type.equals("0")){
                        for(User u:users){
                            if(u.user_id.equals(user_id)){
                                id.set(merchant_id);
                                context.write(id, one);
                                return;
                            }
                        }
                    }
```

```
                    break;
            }
        }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

        private TreeSet<Item> tree = new TreeSet<Item>();
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            Item item= new Item(key.toString(), (long)sum);
            if (tree.size()<100||tree.last().num<item.num){
                tree.add(item);
            }
            if (tree.size()>100){
                tree.pollLast();
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException,
InterruptedException{
            while(!tree.isEmpty()){
                result.set((int)tree.first().num);
                context.write(new Text(tree.first().id), result);
                tree.pollFirst();
            }
        }
    }
```

MapReduce结果：（左：最热门商品，右：最受年轻人关注商家）

1	191499	2494	1	4044	7278
2	353560	2250	2	3491	3661
3	1059899	1917	3	1102	3588
4	713695	1754	4	3828	3434
5	655904	1674	5	4173	3348
6	67897	1572	6	3734	3303
7	221663	1547	7	2385	3214
8	1039919	1511	8	4976	3064
9	454937	1387	9	798	2997
10	81360	1361	10	422	2893

Spark思路：统计最热门商品时只是简单的map和reduce，统计最受年轻人关注的商家时先利用join函数合并RDD，再进行了mapreduce

核心代码：

```
// 统计最受欢迎商品
var itemCount = textFile2.map(line => (line.split(",", -1)(1),line.split(",",
-1)(5), line.split(",", -1).last)).filter(row => row._2 == "1111").filter(row =>
(row._3 != "0")).map(word => (word._1, 1)).reduceByKey((a, b) => a +
b).sortBy(_._2, false)
itemCount=sc.parallelize(itemCount.take(100))
itemCount.saveAsTextFile("hdfs://localhost:9000/output")
// 统计最受年轻人欢迎商家
val textFile3 = textFile2.map(line => (line.split(",", -1)(0),(line.split(",",
-1)(3),line.split(",", -1)(5), line.split(",", -1).last))).filter(row =>
row._2._2 == "1111").filter(row=>row._2._3!="0")
val textFile4=textFile1.map(line=> (line.split(",",-1)(0), line.split(",", -1)
(1))).filter(row=>row._2=="1"||row._2=="2"||row._2=="3")
// textFile的结构为(user_info,((seller_id,time_stamp,action_type),age_range)
val textFile=textFile3.join(textFile4)
var sellerCount=textFile.map(word => (word._2._1._1, 1)).reduceByKey((a, b) => a
+ b).sortBy(_._2, false)
sellerCount=sc.parallelize(sellerCount.take(100))
```

Spark结果：（左：最热门商品，右：最受年轻人关注商家）

1	(191499,2494)	1	(4044,7278)
2	(353560,2250)	2	(3491,3661)
3	(1059899,1917)	3	(1102,3588)
4	(713695,1754)	4	(3828,3434)
5	(655904,1674)	5	(4173,3348)
6	(67897,1572)	6	(3734,3303)
7	(221663,1547)	7	(2385,3214)
8	(1039919,1511)	8	(4976,3064)
9	(454937,1387)	9	(798,2997)
10	(81360,1361)	10	(422,2893)

任务二

任务描述：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

思路：user_info和user_log两张表Inner Join后用filter("属性"==某值)计数，再计算比例（筛去了gender是2或null，age_range是0或null的用户）

核心代码：

```
// 合并两张表，使数据结构为(user_id,((time_stamp,action_type),(age_range,gender))，此
处采用Inner Join
val textFile3 = textFile2.map(line => (line.split(",", -1)(0),(line.split(",",
-1)(5), line.split(",", -1).last))).filter(row => row._2._1 ==
"1111").filter(row=>row._2._2=="2")
val textFile4=textFile1.map(line=> (line.split(",",-1)(0),(line.split(",",-1)
(1),line.split(",",-1).last)))
val textFile=textFile3.join(textFile4)
```

结果：

```
双十一购买商品的男女比例为:
男: 27.67403%
女: 72.325966%

双十一购买商品的各年龄段比例为:
[0,17]: 0.005504077%
[18,24]: 12.7039175%
[25,29]: 33.407505%
[30,34]: 27.372486%
[35,39]: 13.605263%
[40,49]: 10.577408%
[50,..]: 2.3279185%
```

任务三

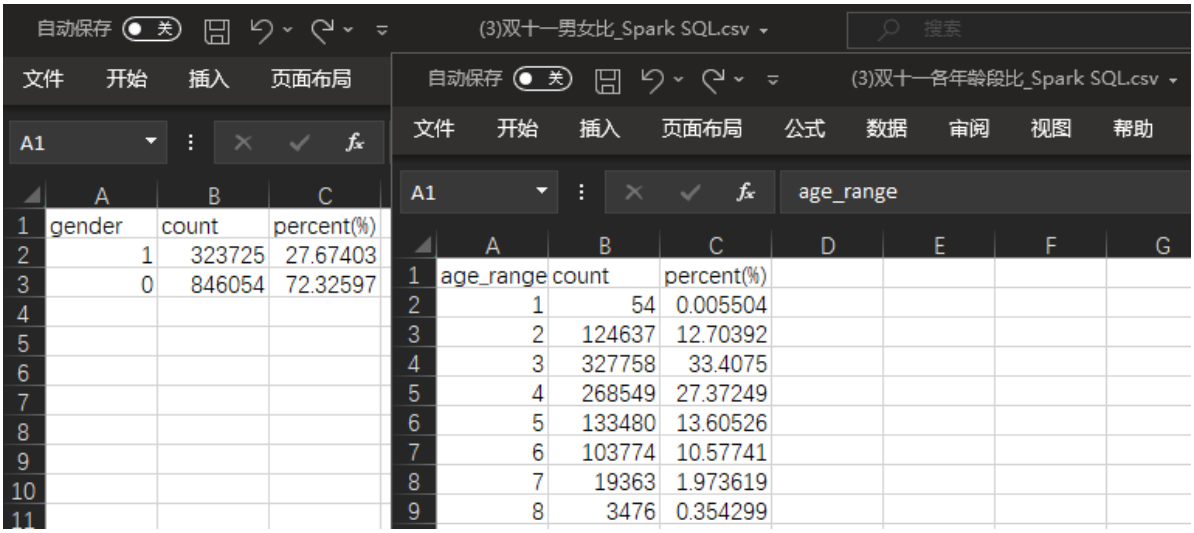
任务描述：基于Hive或Spark SQL查询双十一购买了商品的男女比例，以及购买了商品的买家年龄段的比例

思路：与任务二不同的是，使用了groupBY语句使得代码更加精简了

核心代码：

```
// 筛选时间戳和action_type
val logDF2=spark.sql("SELECT a.user_id,a.action_type FROM user_log a WHERE
a.action_type=2 AND a.time_stamp=1111")
logDF2.createTempView("filtered_log")
// 合并info和log两张表，此处采用Inner Join
val sqlDF=spark.sql("SELECT a.*,b.age_range,b.gender FROM filtered_log a JOIN
user_info b ON a.user_id = b.user_id")
sqlDF.createTempView("FinalTable")
// 计算性别、年龄比例
var Count_gender = sqlDF.groupBy($"gender").count()
Count_gender=Count_gender.filter($"gender"===1 || $"gender"===0)
Count_gender=Count_gender.withColumn("percent(%)",$"count".divide(sum($"count").
over()).multiply(100))
var Count_age_range = sqlDF.groupBy($"age_range").count()
Count_age_range=Count_age_range.filter($"age_range">0 && $"age_range"<9)
Count_age_range=Count_age_range.withColumn("percent(%)",$"count".divide(sum($"co
unt").over()).multiply(100))
Count_age_range=Count_age_range.sort($"age_range")
```

结果：



(3)双十一男女比_Spark SQL.csv

	A	B	C
1	gender	count	percent(%)
2	1	323725	27.67403
3	0	846054	72.32597

(3)双十一各年龄段比_Spark SQL.csv

	A	B	C	D	E	F	G
1	age_range	count	percent(%)				
2	1	54	0.005504				
3	2	124637	12.70392				
4	3	327758	33.4075				
5	4	268549	27.37249				
6	5	133480	13.60526				
7	6	103774	10.57741				
8	7	19363	1.973619				
9	8	3476	0.354299				

任务四

任务描述：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。

思路：

- 新增特征：
  - 用户×商家：
    - user_view_seller_count：该用户对该商家有过几次action
    - view_count/addcart_count/ buy_count/ star_count：该用户对该商家各种action_type的计数
    - buy_ratio：购买次数/action次数
    - buy_ratio2：购买次数/加购物车+购买+收藏次数
    - time_var：该用户在该商家购买时间戳方差
    - item_viewed_count/cat_viewed_count/brand_viewed_count：该用户在该商家浏览过的商品/商品类别/品牌种类
  - 商家：
    - seller_viewed_count：该商家总共被action过几次
    - female_count/male_count：该商家的女/男性买家人数
    - female_ratio/male_ratio：该商家的女/男性买家人数比例
- 特征工程：下采样，随机取训练集中label=0的十五分之一的数据+训练集中label=1的全部数据，使两种标签数目相近
- 模型训练：采用XGBoostClassifier进行模型训练，并调参

代码（太长了略）

结果：天池score：0.6011080

遇到的问题及解决方案

问题一

问题描述&解决方案：xgboost4j对windows系统十分不友好，需要另外下载对应版本的xgboost4j.dll，再复制到xgboost4j的jar包里去；然而现成的xgboost4j.dll只有scala为2.11版本的，我之前配置的scala是2.12版本的，和2.11有冲突，scala版本之间的冲突又会导致更多的问题（包括一直报XGBoostModel trainning failed的错），所以为了解决版本不一致的问题，又把spark从头装了一遍，将原先的3.0.1版本换成了2.4.7版本，对应2.11.12版本的scala。

吐槽：虽然scala语言比较简洁，但是各种各样的版本问题实在是非常浪费时间，比如说这个问题浪费了我整整一天，python相对来说语句没那么简洁，但不会出现版本或者配置问题。

```
C:\Users\Alienware>spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/12/30 05:40:48 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Spark context Web UI available at http://windows10.microdone.cn:4041
Spark context available as 'sc' (master = local[*], app id = local-1609278048598).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.7
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_152)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

问题二

问题：

```
20/12/21 23:27:09 WARN namenode.FSEditLog: No class configured for D, dfs.namenode.edits.journal-plugin.D is empty
20/12/21 23:27:09 ERROR namenode.NameNode: Failed to start namenode.
java.lang.IllegalArgumentException: No class configured for D
        at org.apache.hadoop.hdfs.server.namenode.FSEditLog.getJournalClass(FSEditLog.java:1615)
        at org.apache.hadoop.hdfs.server.namenode.FSEditLog.createJournal(FSEditLog.java:1629)
        at org.apache.hadoop.hdfs.server.namenode.FSEditLog.initJournals(FSEditLog.java:282)
        at org.apache.hadoop.hdfs.server.namenode.FSEditLog.initJournalsForWrite(FSEditLog.java:247)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.format(NameNode.java:985)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.createNameNode(NameNode.java:1429)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.main(NameNode.java:1554)
20/12/21 23:27:09 INFO util.ExitUtil: Exiting with status 1
20/12/21 23:27:09 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
```

解决方案：在hadoop配置文件中D:/some_path前面加上/，变为/D:/some_path

core-site.xml - 记事本

```
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/D:/hadoop/hadoop-2.7.1/data/tmp</value>
        <description>Abase for other temporary directories.</description>
    </property>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

hdfs-site.xml - 记事本

```
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
      <name>dfs.replication</name>
      <value>1</value>
  </property>
  <property>
      <name>dfs.namenode.name.dir</name>
      <value>/D:/hadoop/hadoop-2.7.1/data/namenode</value>
  </property>
  <property>
      <name>dfs.datanode.data.dir</name>
      <value>/D:/hadoop/hadoop-2.7.1/data/datanode</value>
  </property>
  <property>
      <name>dfs.http.address</name>
      <value>127.0.0.1:50070</value>
  </property>
  <property>
      <name>dfs.permissions.enabled</name>
      <value>false</value>
  </property>
</configuration>
```

问题三

问题：权限不够

```
Diagnostics: Exception from container-launch.
Container id: container_1608564909588_0001_02_000001
Exit code: 1
Exception message: CreateSymbolicLink error (1314): ??????????



Stack trace: ExitCodeException exitCode=1: CreateSymbolicLink error (1314): ??????????
```

解决方案：

1. win+R gpedit.msc
2. 计算机配置->windows设置->安全设置->本地策略->用户权限分配->创建符号链接。
3. 把用户添加进去，重启或者注销

思考

改进方向

数据其实还可以挖出来很多，也可以再增加只与用户有关的数据，比如说某用户买得最多的商品/店铺/品牌/类别；成绩较好的选手的经验贴上有写道似乎用上处理过（类似于转成稀疏向量）的user_id和seller_id效果会很好，所以尝试着编了一下独热，但最后因为XGBoost对features格式的限制没有用上，也考虑再换个模型试试。特征工程部分目前的采样方法比较简单，后期可以尝试一下别的采样方法，然后因为数据的维度也比较多，也考虑进行降维/特征提取。

心得体会

从一堆看起来不是都能用的数据里，提取有效信息并且得出结果，是一件很有成就感的事情，虽然score没有特别好，不过也大概能说明并不是在做无用功。并且因为并行处理，速度快了很多。不过调参真的不太有意思，以后可能还是倾向于用pyspark接口写代码，用GridSearchCV自动调参比较省心。