实验四

林恺越 181098163

1、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）；

mapreduce部分是在bdkit上完成的，由于担心数据表过大被集群驱逐，我先用python做了数据预处理，即先选出除点击外的log数据并实现log和info表的连接保存为result.csv（之后在spark中有尝试使用dataframe进行同样操作，结果相同）：

```
import csv
import os

file_old = 'user_log_format1.csv'
file_temp = 'user_log.csv'
with open(file_old, 'r', newline='', encoding='utf-8') as f_old, \
    open(file_temp, 'w', newline='', encoding='utf-8') as f_temp:
    f_csv_old = csv.reader(f_old)
    f_csv_temp = csv.writer(f_temp)
    for i, rows in enumerate(f_csv_old):    # 保留header
        if i == 0:
            f_csv_temp.writerow(rows)
            print(rows)
            break
    for rows in f_csv_old:
        if (rows[6] in ['1','2','3']) and rows[5]=='1111':
            f_csv_temp.writerow(rows)
            print(rows)

import pandas
source_filename = "user_info_format1.csv"
aim_filename    = "user_log.csv"
source_file = pandas.read_csv(source_filename)
aim_file = pandas.read_csv(aim_filename)
source = pandas.DataFrame({
            'user_id':source_file['user_id'],
            'age_range':source_file['age_range'],
            'gender':source_file['gender'],
    })
aim = pandas.DataFrame({
            'user_id':aim_file['user_id'],
            'item_id':aim_file['item_id'],
            'seller_id':aim_file['seller_id'],
            'action_type':aim_file['action_type'],
    })
aim['age_range'] = aim.merge(source, how = 'left', on = 'user_id')['age_range']
aim['gender'] = aim.merge(source, how = 'left', on = 'user_id')['gender']
aim.to_csv('result.csv')
```

第一步mapreduce因为之前已经实现过了，编程部分基本没有什么问题，具体代码见仓库

运行前需要使用命令 sed -i '1d' input.csv进行去表头

过程中遇到了一个很玄学的bug:

```
//if(fields[3]=="1"| fields[3]=="2"|fields[3]=="3") {
 String str="";
 for (int i=0;i<fields.length;i++)
    if(i==3)
        if(Integer.parseInt(fields[i])<4&Integer.parseInt(fields[i])>0){
                str=fields[i-1];
                 k.set(str);
                 context.write(k, v);
}
```

这里被注释掉的部分fields[3]直接使用会报错“index out of range ”但是我尝试直接把fields输出到文件确实有4个值，访问fields[3]应该没有越界，而改成for循环当i=3时访问fields[3]是可以正常跑的，没有搞懂是什么原因。

spark部分我使用的是pyspark， 配置过程参考了https://blog.csdn.net/lemon_zl/article/details/107209819的教程，配置成功截图：

In [1]:
```
import os
import sys
spark_home = os.environ.get('SPARK_HOME', None)
if not spark_home:

    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.10.4-src.zip'))
comm=os.path.join(spark_home, 'python/lib/py4j-0.10.4-src.zip')
print ('start spark....',comm)
exec(open(os.path.join(spark_home, 'python/pyspark/shell.py')).read())
```

```
start spark.... C:\Users\953031197\Downloads\spark-3.0.1-bin-hadoop2.7\python/lib/py4j-0.10.4-src.zip
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Python version 3.7.4 (default, Aug  9 2019 18:34:13)
SparkSession available as 'spark'.
```

spark代码（item统计部分,seller类似具体见仓库）：

主要步骤：

1、先筛选出除单击以外、双十一当天的数据

2、提取item_id这一列

3、对其进行词频统计，并倒序输出取前一百的值

```
import sys
from operator import add
from pandas.core.frame import DataFrame
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType

from pyspark.sql import SQLContext
from pyspark import SparkContext

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("MostPopularItemCount")\
        .getOrCreate()

    lines=spark.read.csv(r"C:\\Users\\953031197\\Desktop\\data_format1\\user_log_format1.csv", encoding='gbk', header=True, inferSchema=True)
    lines=lines.filter((lines.action_type!=0) & (lines.time_stamp==1111))
    lines=lines.withColumn("item_id", lines["item_id"].cast(StringType()))
    lines=lines[['item_id']].rdd

    counts =lines.map(lambda x: (x, 1))\
                .reduceByKey(add)\
                .sortBy(keyfunc=(lambda x:x[1]),ascending=False)\
                .take(100)
    output = counts
    with open('item.txt', 'w') as f:
        for (word, count) in output:
            print("%s: %i" % (word, count))
            f.write(str(word)+","+str(count)+"\r")
    spark.stop()
```

结果：

item前五，具体见仓库pyspark/output文件夹:

Row(item_id='191499'),2494
Row(item_id='353560'),2250
Row(item_id='1059899'),1917
Row(item_id='713695'),1754
Row(item_id='655904'),1674

seller前五，具体见仓库pyspark/output文件夹:

Row(seller_id='4044'),7278
Row(seller_id='3491'),3661
Row(seller_id='1102'),3588
Row(seller_id='3828'),3434
Row(seller_id='4173'),3348

2、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；

男女比例统计部分代码（年龄比例类似，具体见仓库）：

主要步骤：

1、先筛选出“购买”了商品的数据（result.csv是上面提到的预先经过python处理的数据）

2、对user_id去重，重复值保留一个

3、去除gender值无效的数据

4、对gender词频统计，输出统计的不同性别的人数和占比（因为之前统计item和seller时的输出是直接输出Row的信息，不太好看，这次输出格式换了一下）

```
from __future__ import print_function

import sys
from operator import add
from pandas.core.frame import DataFrame
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType,IntegerType

from pyspark.sql import SQLContext
from pyspark import SparkContext

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("GenderCount")\
        .getOrCreate()

    lines=spark.read.csv(r"C:\\Users\\953031197\\Desktop\\data_format1\\result.csv", encoding='gbk', header=True, inferSchema=True)
    lines=lines.filter((lines.action_type==2) )
    lines=lines.dropDuplicates(subset=["user_id"])#去重
    lines=lines.filter((lines.gender==0) | (lines.gender==1))
    lines.show()
    lines=lines.withColumn("gender", lines["gender"].cast(IntegerType()))
    lines=lines[['gender']].rdd
    gender=["female","male"]
    counts =lines.map(lambda x: (x, 1))\
                .reduceByKey(add)\
                .sortBy(keyfunc=(lambda x:x[1]),ascending=False)\
                .take(2)
    output = counts
    sum=0
    with open('gender.txt', 'w') as f:
        for (word, count) in output:
            print(gender[word.gender]+","+str(count)+"\r")
            f.write(gender[word.gender]+","+str(count)+"\r")
    with open('gender_rate.txt', 'w') as f1:
        for (word, count) in output:
            sum+=count
        for (word, count) in output:
            print(gender[word.gender]+str(round(count*100/sum,2)+"%"))
            f1.write(gender[word.gender]+","+str(round(count*100/sum,2))+"%"+"\r")
    spark.stop()
```

结果：

gender人数和占比：

female,285638	female:70.13%
male,121670	male:29.87%

age_range人数和占比：

[25,29],111654	[25,29]:33.93%
[30,34],79991	[30,34]:24.31%
[18,24],52871	[18,24]:16.07%
[35,39],40777	[35,39]:12.39%
[40,49],35464	[40,49]:10.78%
>=50,8258	>=50:2.51%
<18,24	<18:0.01%

3、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；

基于sparkSQL查询男女比例（年龄比例类似，具体见仓库）：

主要步骤：

1、先筛选出“购买”了商品的数据（result.csv是上面提到的预先经过python处理的数据）

2、对user_id去重，重复值保留一个

3、去除gender值无效的数据

4、按gender分组查询并计数，输出查询结果

```
from __future__ import print_function

import sys
from operator import add
from pandas.core.frame import DataFrame
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType
import pyspark.sql.functions as F
from pyspark.sql import SQLContext
from pyspark import SparkContext

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("GenderSQL")\
        .getOrCreate()

    df=spark.read.csv(r"C:\\Users\\953031197\\Desktop\\data_format1\\result.csv", encoding='gbk', header=True, inferSchema=True)
    df=df.filter(df.action_type==2)
    df=df.dropDuplicates(subset=["user_id"])#去重
    df=df.filter((df.gender==0) | (df.gender==1))
    df=df.groupby('gender').agg(F.count(df['gender']))
    df.show()
    gender=["female","male"]
    list=df.collect()
    sum=list[0]['count(gender)']+list[1]['count(gender)']
    print(gender[int(list[0].gender)]+":"+str(round(list[0]['count(gender)']*100/sum,2))+"%")
    print(gender[int(list[1].gender)]+":"+str(round(list[1]['count(gender)']*100/sum,2))+"%")
```

Spark SQL的代码相对比task2的代码要简练很多，因为避免了dataframe和rdd的转换。

就使用感觉来说，dataframe类似SQL使用起来更好上手，读写文件不局限于hdfs会更加自由，类似pd.dataftame且两者可以转换。

结果：

gender人数和占比:

```
+------+-------------+
|gender|count(gender)|
+------+-------------+
|   0.0|       285638|
|   1.0|       121670|
+------+-------------+

female:70.13%
male:29.87%
```

age_range人数和占比:

age_range	count(age_range)
7.0	8258
1.0	24
4.0	79991
3.0	111654
2.0	52871
6.0	35464
5.0	40777

>=50:2.51%
<18:0.01%
[30,34]:24.3%
[25,29]:33.92%
[18,24]:16.06%
[40,49]:10.77%
[35,39]:12.39%

4、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。

（1）特征提取，参考了天池baseline中的特征：

想要建立的特征

需要根据user_id，和merchant_id（seller_id）,从用户画像表以及用户日志表中提取特征，填写到df_train这个数据框中，从而训练评估模型 需要建立的特征如下：

- 用户的年龄(age_range)
- 用户的性别(gender)
- 某用户在该商家日志的总条数(total_logs)
- 用户浏览的商品的数目，就是浏览了多少个商品(unique_item_ids)
- 浏览的商品的种类的数目，就是浏览了多少种商品(categories)
- 用户浏览的天数(browse_days)
- 用户单击的次数(one_clicks)
- 用户添加购物车的次数(shopping_carts)
- 用户购买的次数(purchase_times)
- 用户收藏的次数(favourite_times)

以此为基础建立了13个特征：

('age_range',年龄)
('gender',性别)
('total_count',在该商家的总log数)
('total_unique_item',浏览的总商品数)
('unique_item',浏览的该商家商品数)
('total_unique_cat',浏览的总商品种类数)

('unique_cat',浏览的该商家商品种类数)
('total_browse_days',浏览的总天数)
('browse_days',浏览该商家的天数)
('one_clicks',在该商家单击的次数)
('shopping_carts',在该商家添加购物车的次数)
('purchase_times',在该商家购买的次数)
('favourite_times',在该商家添加收藏夹的次数)

并对train_format1和test_format1进行处理（代码见task4-data_process.py）得到processed_train_data.csv和processed_test_data.csv两个文件并保存。

主要步骤：

1、提取user_id,seller_id,和想统计的数据，去除重复值

2、按user_id(或[user_id,seller_id],根据特征提取的需要)分组，分别计数

3、更改列名，与train_data(或test_data)进行连接，这里需要注意的是需要指明是左连接，'left'，一开始没有指定，默认是inner连接，最后的结果只有600多行

```
total_unique_item = log_data.dropDuplicates(subset=[c for c in log_data.columns if c in ["user_id","seller_id","item_id"]])
                                            .groupBy("user_id")
                                            .agg({"item_id": "count"})
                                            .withColumnRenamed("count(item_id)", "total_unique_item")#浏览的总商品数
train_data=train_data.join(total_unique_item,["user_id"],'left')
```

（2）训练模型

之前需要处理的数据通常只有一列，所以一般使用.cast(StringType())　对类型进行限定，这次因为涉及的列较多，所以直接设定schema规定类型会更方便。

```
schema_train=StructType(
  [
    StructField('user_id',DoubleType()),
    StructField('seller_id',DoubleType()),
    StructField('label',DoubleType()),
    StructField('age_range',DoubleType()),
    StructField('gender',DoubleType()),
    StructField('total_count',DoubleType()),
    StructField('total_unique_item',DoubleType()),
    StructField('unique_item',DoubleType()),
    StructField('total_unique_cat',DoubleType()),
    StructField('unique_cat',DoubleType()),
    StructField('total_browse_days',DoubleType()),
    StructField('browse_days',DoubleType()),
    StructField('one_clicks',DoubleType()),
    StructField('shopping_carts',DoubleType()),
    StructField('purchase_times',DoubleType()),
    StructField('favourite_times',DoubleType())
  ]
)
```

读取之前处理好的训练数据，对表中null值进行填充（主要针对计数的变量，例如单击数、收藏数统计），gender和age_range 中的null 值无法处理，所以舍弃。

```
train_data=spark.read.format('csv').option('sep',',').option('header','true').load('C:\\Users\\953031197\\Desktop\\processed_train_data.csv',schema=schema_train)
train_data.show(5)
train_data=train_data.fillna(0, subset=['shopping_carts', 'favourite_times', 'one_clicks','browse_days','purchase_times']) #填充null值
train_data.show(5)
train_data = train_data.filter(train_data.gender. isNotNull())
train_data = train_data.filter(train_data.age_range. isNotNull())
train_data.show(5)
train_data=train_data.rdd
```

如果训练数据中还有null值的话会报错：

```
requirement failed: init value should <= bound
```

（这是网上的图，自己忘了截图）一开始从字面意思理解，以为是训练设置的迭代数太大导致超过bound，但是怎么调都是这个错，最后搜了半天找到这个，保证所有null值都填充之后就可以正常跑了：

原因：run()中给定的训练集数据有问题，比如说出现NAN等情况

这个问题导致模型无法通过训练构造，预测时不会报错。（可以通过加载已经持久化的模型得到）
说明了构造模型的过程对数据要求比较严格，而测试不严格

解决方法：出现NAN一般是由infinity转换而来，infinity在scala中为无穷大，可以由除以0产生，数据做好控制即可。

scala本身不够完善，基本由java衍生，依赖JVM，异常处理不够清晰。

模型训练选择了两个，支持向量机SVM和逻辑回归LR，按照7：3划分训练集和测试集：

```
(trainData, testData) = train_data.randomSplit([7,3])#划分训练集和测试集
trainData=trainData.map(lambda line: LabeledPoint(line[2],[line[3:]]))
testData=testData.map(lambda line: LabeledPoint(line[2],[line[3:]]))
trainData.cache()
trainCount=float(trainData.count())
testData.cache()
testCount=float(testData.count())
LR=LogisticRegressionWithLBFGS()
LR=LR.train(trainData,iterations=1000)#逻辑回归
SVM=SVMWithSGD.train(trainData,iterations=1000)#支持向量机
predictedSVM=trainData.map(lambda x:(x.label,SVM.predict(x.features)))
predictedLR=trainData.map(lambda x:(x.label,LR.predict(x.features)))
accuracySVM=predictedSVM.filter(lambda p:p[0]==p[1]).count()/trainCount
accuracyLR=predictedLR.filter(lambda p:p[0]==p[1]).count()/trainCount
print("SVM training accuracy="+str(round(accuracySVM*100,2))+"%")#SVM训练集准确度
print("LR training accuracy="+str(round(accuracyLR*100,2))+"%")#LR训练集准确度
te_predictedSVM=testData.map(lambda x:(x.label,SVM.predict(x.features)))
te_predictedLR=testData.map(lambda x:(x.label,LR.predict(x.features)))
te_accuracySVM=te_predictedSVM.filter(lambda p:p[0]==p[1]).count()/testCount
te_accuracyLR=te_predictedLR.filter(lambda p:p[0]==p[1]).count()/testCount
print("SVM testing accuracy="+str(round(te_accuracySVM*100,2))+"%")#SVM测试集准确度
print("LR testing accuracy="+str(round(te_accuracyLR*100,2))+"%")#LR测试集准确度
```

结果：

```
SVM training accuracy=93.81%
LR training accuracy=93.78%
SVM testing accuracy=93.66%
LR testing accuracy=93.65%
```

（3）进行预测

```
result=spark.read.csv(r"C:\\Users\\953031197\\Desktop\\data_format1\\test_format1.csv", encoding='gbk', header=True, inferSchema=True).withColumnRenamed("merchant_
result=result.drop('prob')
predict_data=spark.read.format('csv').option('sep',',').option('header','true').load('C:\\Users\\953031197\\Desktop\\processed_test_data.csv',schema=schema_predict)
predict_data.show(5)
predict_data=predict_data.fillna(0, subset=['shopping_carts','prob', 'favourite_times', 'one_clicks','browse_days','purchase_times']) #填充null值
predict_data.show(5)
predict_data = predict_data.filter(predict_data.gender. isNotNull())
predict_data =predict_data.filter(predict_data.age_range. isNotNull())
predict_data.show(5)
predictData=predict_data.rdd
predictData=predictData.map(lambda line:(line[0],line[1], LabeledPoint(line[2],[line[3:]])))
predictData.cache()
LR.clearThreshold()#可能性预测
#pr_predictedSVM=predictData.map(lambda p:(p[0],p[1],SVM.predict(p[2].features)))
pr_predictedLR=predictData.map(lambda p:(p[0],p[1],LR.predict(p[2].features)))
#dfSVM=pr_predictedSVM.toDF().withColumnRenamed("_1", "user_id").withColumnRenamed("_2", "seller_id").withColumnRenamed("_3", "SVM")
dfLR=pr_predictedLR.toDF().withColumnRenamed("_1", "user_id").withColumnRenamed("_2", "seller_id").withColumnRenamed("_3", "prob")
#result=result.join(dfSVM,["user_id","seller_id"],'left')
result=result.join(dfLR,["user_id","seller_id"],'left').withColumnRenamed("seller_id", "merchant_id")
result.toPandas().to_csv('prediction.csv')
```

对预测数据的处理类似之前的训练数据，除了预测集的prob为空，这个我一开始没在意，就按之前的步骤做，然后会报错：

TypeError: float() argument must be a string or a number, not 'NoneType'

填充完后可以正常运行， 我是先尝试了只保存预测的一列值，然后发现无法转成dataframe，会报错

Can not infer schema for type: <type 'unicode'>

查询发现如果只有一列值的话需要使用：

```
dfSVM=pr_predictedSVM.map(lambda x: (x, )).toDF()
```

不过因为这样单列保存的话没办法和test_format1进行连接，所以在map时加上了user_id和seller_id，最后的rdd就有3列，可以直接toDF()，上面的命令没用到。

最后的结果是，SVM将所有用户都预测成了不会复购，LR则预测出一部分用户会复购。考虑是不会复购的人数远多于会复购的人数，两类数据不太平衡以及特征选取不够完善的原因导致的。

由于需要给出预测的可能性，所以使用LR.clearThreshold()保证输出的是原始概率，最终结果见prediction.csv

(4)提交结果：原始的数据提交是633/0.444956

然后我尝试对原始prediction做一些处理再提交：

大于0.5就预测为1，小于0.5就预测为0：**score:** 0.5016117

全部预测为0：**score:** 0.5000000

大于0.4就预测为1，小于0.4就预测为0：**score:** 0.5020465

以0.3和0.2为判断标准结果相同：**score:** 0.5029915

最高的排名是493/0.5029915