实验 4

221275029 杨珺雯

安装 spark

1.备份 apt 源，删除旧的源文件，使用 echo 命令将源写入新文件。

```
Python
#备份 apt 源
cp /etc/apt/sources.list /etc/apt/sources_init.list
#新的 apt 源
echo "
deb-src http://archive.ubuntu.com/ubuntu xenial main restricted
#Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted
multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main
restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main
restricted multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
deb http://mirrors.aliyun.com/ubuntu/ xenial multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main
restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main
restricted universe multiverse #Added by software-properties
deb http://archive.canonical.com/ubuntu xenial partner
deb-src http://archive.canonical.com/ubuntu xenial partner
deb http://mirrors.aliyun.com/ubuntu/ xenial-security main
restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main
restricted multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-security
multiverse">/etc/apt/sources.list
#更新 apt 源
apt update
```

2.安装 scala 和 java，直接输入命令

```
Python
#安装 jdk1.8
```

```
apt install openjdk-8-jdk
#安装 scala
apt install scala
```

3.安装 SSH

```
Python
apt-get install openssh-server
#安装 SSH 客户端
apt-get install openssh-client
#进入当前用户的用户根目录生成密钥
cd ~
ssh-keygen -t rsa -P ""
#将公钥追加到 authorized_keys 文件中
cat .ssh/id_rsa.pub >> .ssh/authorized_keys
#启动 SSH 服务，免密登录自己
service ssh start
ssh 127.0.0.1
#修改 ~/.bashrc 文件，启动 shell 的时候，自动启动 SSH 服务
vim ~/.bashrc
```

在 bashrc 文件中添加

```
Plain Text
service ssh start
```

4.下载 spark

从链接中下载 spark

```
Python
wget https://dlcdn.apache.org/spark/spark-3.5.3/spark-3.5.3-bin-
hadoop3.tgz
#解压到/usr/local 目录下，修改名字为 spark
tar -zxvf spark-3.5.3-bin-hadoop3.tgz  -C /usr/local/
cd /usr/local/
```

修改/etc/profile 环境变量文件，添加 spark 的环境变量，追加下述代码

```
Plain Text
export SPARK_HOME=/usr/local/spark
```

```
export PATH=$PATH:$SPARK_HOME/bin
```

使环境变量配置文件生效

```
Plain Text
/usr/local# source /etc/profile
```

5.在目录/usr/local/spark/conf 修改配置

```
Python
mv spark-env.sh.template spark-env.sh
```

在文件末尾追加

```
Python
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export SCALA_HOME=/usr/share/scala

export SPARK_MASTER_HOST=h01
export SPARK_MASTER_IP=h01
export SPARK_WORKER_MEMORY=4g
```

在 workers 文件中修改内容为

```
Python
h01
h02
h03
h04
h05
```

启动 Spark

```
Plain Text
./start-all.sh
```

任务 1：Spark RDD 编程

1、查询特定日期的资金流入和流出情况

使用 user_balance_table， 计算出所有用户在每一天的总资金流入和总资金流出量。

思路

首先创建一个 spark 环境并命名，然后读取文件 user_balance_table.csv 并返回一个包含文件每一行的 RDDdata，每一行是一个字符串；接着用 data 的 filter 函数过滤掉表头，只保留数据部分。

创建一个数据解析函数 parse_line(line)，把每一行的字符串用逗号分隔开，并提取日期、购买金额和赎回金额的数据形成一个元组返回；创建一个按日期聚合的函数 aggregagte_by_date(accumulated,current),current 是包含每一行购买金额和赎回金额的元组，accumulated 是目前计算的总购买金额和总赎回金额的元组，该函数解包 current 元组，将 accumulated 元组转换为列表，并分别加上对应的购买金额和赎回金额，最后返回计算的总值。

使用 map 转换操作将 data 的每一行数据应用到 parse_line 函数，从而得到一个新的 RDDparsed_data，包含 (date, total_purchase_amt, total_redeem_amt) 形式的元组；用 map 转换操作将 parsed_data 将每个解析后的元组转换为 (日期, (流入金额, 流出金额)) 格式，再利用 reduceByKey 操作将每一行数据应用到 aggregate_by_date 函数，累加每个日期的流入和流出金额，得到一个新的 RDDaggregated_data，包含 (日期, (总流入金额, 总流出金额)) 格式的元组；最后用 collect 操作聚合后的数据从分布式计算环境中收集到本地，并用 sorted 函数根据日期对结果排序。打印出结果，并保存到 flow_count.csv 文件中，停止 SparkContext，释放资源。

运行结果（不完整）：

```
Date,Purchase,Redeem
20130701,32488348.0,5525022.0
20130702,29037390.0,2554548.0
20130703,27270770.0,5953867.0
20130704,18321185.0,6410729.0
20130705,11648749.0,2763587.0
20130706,36751272.0,1616635.0
20130707,8962232.0,3982735.0
20130708,57258266.0,8347729.0
20130709,26798941.0,3473059.0
20130710,30696506.0,2597169.0
20130711,44075197.0,3508800.0
20130712,34183904.0,8492573.0
20130713,15164717.0,3482829.0
20130714,22615303.0,2784107.0
20130715,48128555.0,13107943.0
20130716,50622847.0,11864981.0
20130717,29015682.0,10911513.0
20130718,24234505.0,11765356.0
20130719,33680124.0,9244769.0
20130720,20439079.0,4601143.0
20130721,21142394.0,2681331.0
20130722,40448896.0,19144267.0
20130723,58136147.0,24404051.0
20130724,48422518.0,36258592.0
20130725,57433418.0,38212836.0
20130726,44721817.0,39192369.0
20130727,17194451.0,15058893.0
20130728,36255382.0,7683211.0
20130729,53512076.0,18599364.0
```

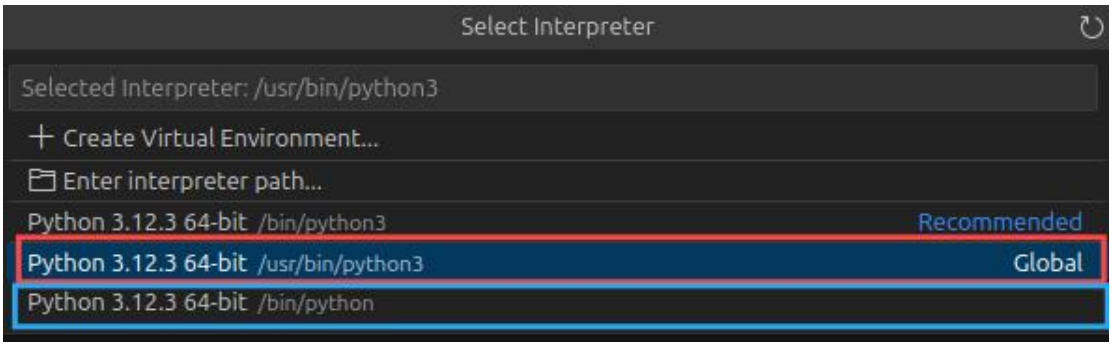
遇到问题

1）安装 python3 和 pip 之后无法安装 pyspark，按照报错提示修改命令为 pip install python3-pyspark 会报错 Unable to locate package python3-pyspark

解决方法：使用 pip install --break-system-packages pyspark 命令强制安装 pyspark

2）安装 python3 后"from pyspark import SparkContext, SparkConf"报错显示无法加载 pyspark

解决方法：python 解释器的路径选错了，根据 which python 找到 python 解释器的位置，再在设置里选择合适的 python 解释器，如下面图片，从蓝色框的 python 解释器换位红色框的 python 解释器。



2、活跃用户分析

使用 user_balance_table，定义活跃用户为在指定月份内有至少 5 天记录的用户，统计 2014 年 8月的活跃用户总数。

思路

首先创建一个 spark 环境并命名，然后读取文件 user_balance_table.csv 并返回一个包含文件每一行的 RDDdata，每一行是一个字符串；接着用 data 的 filter 函数过滤掉表头，只保留数据部分。

创建一个数据解析函数 parse_line(line)，把每一行的字符串用逗号分隔开，并提取日期和用户 ID 的数据形成一个元组返回；创建一个筛选出 2014 年 8 月记录的函数 filter_august(date)，把日期开头是"202408"的记录筛选出来。

使用 map 转换操作将 data 的每一行数据应用到 parse_line 函数，从而得到一个新的 RDDparsed_data，包含 (user_id,date) 形式的元组；用 map 转换操作将 parsed_data 将每个解析后的元组提取出日期 date，再用到 filter_august 函数，过滤出 2014 年 8 月的数据，得到一个新的 RDDaugust_data，包含 (user_id,date) 格式的元组；用 distinct 操作确保每个用户每个日期只统计一次，再用 mapvalues 操作把每个用户的每个日期设为数量 1，利用 reduceByKey 的操作统计每个用户再 2014 年 8 月出现的不同日期数量，用 filter 操作统计大于 5 的用户，打印出结果，停止 SparkContext，释放资源。

运行结果

任务 2：Spark SQL 编程

1、按城市统计 2014 年 3月1日的平均余额

计算每个城市在 2014 年 3月1日的用户平均余额 (tBalance)，按平均余额降序排列。

思路

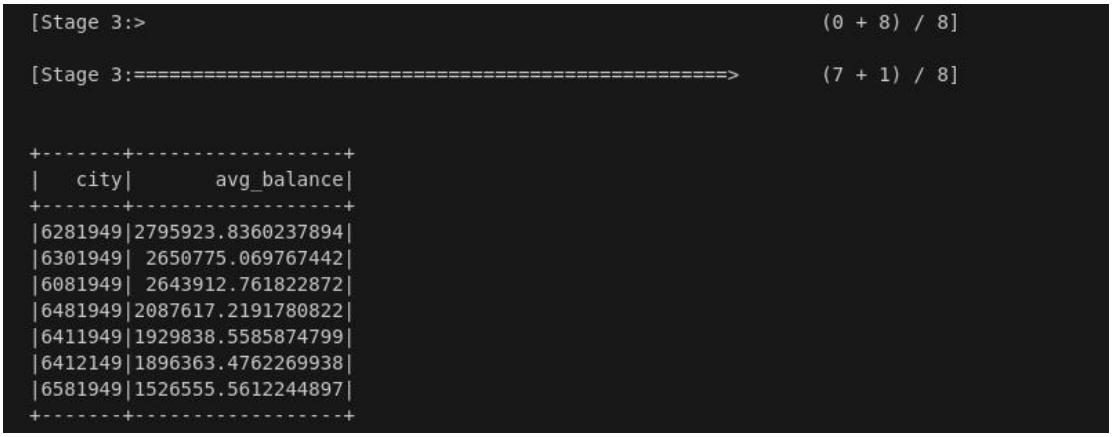
首先创建并获取一个 SparkSession 实例，加载数据文件 user_balance_table.csv 和 user_profile_table.csv 形成两个 DataFrame，把 tBalance 数据转换为 float 类型，user_id 和 city 转换为 string 类型，用 createOrReplaceTempView() 将两个 DataFrame 注册为临时视图，这样可以在 SQL 查询中使用它们。

写 SQL 语句：用 join on 通过 user_id 连接两张表，用 where 语句限定 2014 年 3 月 1 日，用 group by 语句按照城市分类，用 AVG()函数计算平均值，最后用 ORDER BY 进行排序。

```
Python
query = """
SELECT p.city, AVG(b.tBalance) AS avg_balance
FROM user_balance b
JOIN user_profile p ON b.user_id = p.user_id
WHERE b.report_date = '20140301'
GROUP BY p.city
ORDER BY avg_balance DESC
"""
```

用 spark.sql(query)执行上述的 SQL 查询，返回结果为一个 DataFrame，显示查询结果，关闭 Spark 会话，释放资源。

运行结果



2、统计每个城市总流量前 3高的用户

统计每个城市中每个用户在 2014 年 8月的总流量（定义为 total_purchase_amt + total_redeem_amt ），并输出每个城市总流量排名前三的用户 ID 及其 总流量。

思路

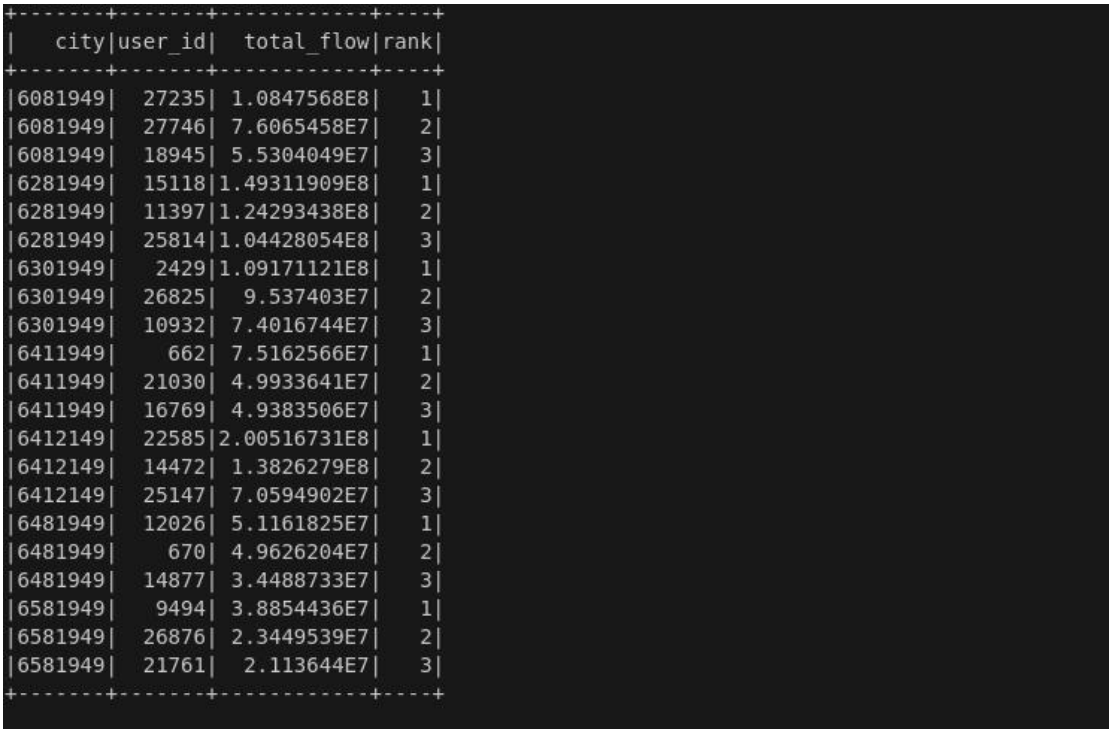
首先创建并获取一个 SparkSession 实例，加载数据文件 user_balance_table.csv 和 user_profile_table.csv 形成两个 DataFrame，把 total_purchase_amt 和 total_redeem_amt 数据转换为 float 类型，用 createOrReplaceTempView() 将两个 DataFrame 注册为临时视图，这样可以在 SQL 查询中使用它们。

写 SQL 语句：用 left join on 通过 user_id 左连接两张表，用 where 语句限定 2014 年 8 月，用 group by 语句按照城市和 user_id 分类，用 SUM()函数求和，最后用 ORDER BY 进行排序。

```
Python
query = """
    SELECT up.city, ub.user_id,
           (SUM(ub.total_purchase_amt) + SUM(ub.total_redeem_amt))
AS total_flow
    FROM user_profile AS up
    LEFT JOIN user_balance AS ub
    ON up.user_id = ub.user_id
    WHERE ub.report_date BETWEEN '20140801' AND '20140831'
    GROUP BY up.city, ub.user_id ORDER BY total_flow DESC
"""
```

用 spark.sql(query)执行上述的 SQL 查询，返回结果为一个 DataFrame，显示查询结果，关闭 Spark 会话，释放资源。

结果



city	user_id	total_flow	rank
6081949	27235	1.0847568E8	1
6081949	27746	7.6065458E7	2
6081949	18945	5.5304049E7	3
6281949	15118	1.49311909E8	1
6281949	11397	1.24293438E8	2
6281949	25814	1.04428054E8	3
6301949	2429	1.09171121E8	1
6301949	26825	9.537403E7	2
6301949	10932	7.4016744E7	3
6411949	662	7.5162566E7	1
6411949	21030	4.9933641E7	2
6411949	16769	4.9383506E7	3
6412149	22585	2.00516731E8	1
6412149	14472	1.3826279E8	2
6412149	25147	7.0594902E7	3
6481949	12026	5.1161825E7	1
6481949	670	4.9626204E7	2
6481949	14877	3.4488733E7	3
6581949	9494	3.8854436E7	1
6581949	26876	2.3449539E7	2
6581949	21761	2.113644E7	3

任务 3：Spark ML 编程

使用 Spark MLlib 提供的机器学习模型，预测 2014 年 9 月每天的申购与赎回总额。

思路

使用随机森林模型，用最基础的特征（日期）来构建模型并预测 2014 年 9 月每天的总申购金额和赎回金额。

具体步骤:

1.首先，创建 spark 会话，通过指定路径读取 csv 文件，加载为 dataframe。

2.使用 dataframe 的 groupby 函数按照 report_date 分组，agg 函数聚合，计算每一天的总申购金额和总赎回金额，和原数据一起保存到新的 dataframe daily_summary_df。

3.添加新的一列 formatted_date,这一列是 report_date 转换为 yyyymmdd 形式的字符串类型的日期。给原数据添加有关日期的特征（利用 year(),month(),dayofmonth()和 dayofweek()函数从日期 formatted_date 中提取出年、月、日和星期几的特征）并添加到 dataframe daily_summary_df 中。

4.添加滞后特征。创建 windows 窗口，使用窗口的 orderby 函数按照日期排序，再使用 lag 函数提取把预测日期 1 天、2 天...29 天之前的申购金额和赎回金额作为特征添加进去。最后使用 na.fill 函数把缺失值填充为 0。

5.使用 vectorassembler 函数把所有的特征名称合并为一个向量 features，用于模型训练，把 features 中涉及到的特征和目标（总申购金额）从原数据中提取出来，合并为一个向量；把 features 中涉及到的特征和目标（总赎回金额）从原数据中提取出来，合并为一个向量，这两个向量是训练数据。

6.用特征向量 features 和目标（total_purchase_amt 和 total_redeem_amt）分别构建两个随机森林模型，然后用 fit 函数把训练数据代入进去，形成两个预测模型。

7.已知 2014 年 9 月的日期，用这些日期按照步骤 3、4、5 构建预测的数据集。

8.把预测的数据分别代入预测模型，把结果按照 report_date 合并，保存到 csv 文件。

结果

遇到问题

1）类型不匹配：计算每一天的总申购金额和总赎回金额的时候显示 int 和 string 不能相加，说明读取数据时有数据的类型不符合要求。

解决方法：在读数据的代码 `user_balance_df = spark.read.csv("/home/hadoop/Downloads/Purchase_Redemption_Data/user_balance_table.csv", header=True, inferSchema=True)`中设置 inferSchema=True，意思是自动识别数据类型。或者读完了手动把所有的申购金额和赎回金额的数据改为 float 类型。

2）运行后在把数据代入模型的部分显示某一列不存在

解决方法：说明准备的数据和特征不匹配，可能数据少准备了一个特征，补上就可以了。