

221275028-张伊璐-实验4

0.准备

启动hadoop

```
zhangyilu@zhangyilu-VMware-Virtual-Platform:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as zhangyilu in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [zhangyilu-VMware-Virtual-Platform]
Starting resourcemanager
Starting nodemanagers
zhangyilu@zhangyilu-VMware-Virtual-Platform:~$ jps
14449 Jps
13411 NameNode
13971 ResourceManager
14088 NodeManager
13545 DataNode
13755 SecondaryNameNode
```

启动spark

```
zhangyiliu@zhangyiliu-VMware-Virtual-Platform: /usr/local/spark$ bin/pyspark
Python 3.12.3 (main, Sep 11 2024, 14:17:37) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Welcome to

      _
     / _/ _ \ _ _ _ _ _ _ _ \ _/ _
    _ \ V _ \ V _ ' / _/ ' _/
   / _ / _ _ \ _ _ _ / _ \ _ \   version 3.5.4
    / _/

Using Python version 3.12.3 (main, Sep 11 2024 14:17:37)
Spark context Web UI available at http://192.168.20.128:4042
Spark context available as 'sc' (master = local[*], app id = local-1734974417721
).
SparkSession available as 'spark'.
>>> sc = SparkContext(appName="UserBalanceAnalysis")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/spark/python/pyspark/context.py", line 201, in __init__
    SparkContext.ensure_initialized(self, gateway=gateway, conf=conf)
```

1. SparkRDD

1.1 查询特定日期的资金流入和流出情况

核心代码：

```

# 解析CSV文件，提取日期、资金流入和流出量
✓ def parse_line(line):
    parts = line.split(',')
    date = parts[1].strip()
    try:
        purchase_amt = float(parts[4].strip() or 0)
    except ValueError:
        purchase_amt = 0.0
    try:
        redeem_amt = float(parts[8].strip() or 0) # total_redeem_amt, default to 0 if missing
    except ValueError:
        redeem_amt = 0.0
    return (date, purchase_amt, -redeem_amt) # 资金流入和流出量

# 转换为RDD并去除空行
balance_rdd = data.map(parse_line).filter(lambda x: x[0] is not None)

# 分别计算资金流入和流出
inflow_rdd = balance_rdd.map(lambda x: (x[0], x[1])).reduceByKey(lambda a, b: a + b)
outflow_rdd = balance_rdd.map(lambda x: (x[0], x[2])).reduceByKey(lambda a, b: a + b)

# 合并流入和流出量
daily_balance = inflow_rdd.join(outflow_rdd).mapValues(lambda x: (x[0], -x[1]))

sorted_daily_balance = daily_balance.sortByKey()

```

问题：一直报错ValueError: could not convert string to float: 'total_purchase_amt'

解决：报错的意思是我尝试将字符串转换为浮点数，原因为没有去掉首行（标题行），因此去掉标题行即可

```

lines = sc.textFile("file:///home/zhangyilu/downloads/user_balance_table.csv")
header = lines.first() # 获取第一行，即标题行
data = lines.filter(lambda line: line != header)
# 解析CSV文件，提取日期、资金流入和流出量
def parse_line(line):

```

运行结果：

```
26
27 # 分别计算资金流入和流出
28 inflow_rdd = balance_rdd.map(lambda x: (x[0], x[1])).reduceByKey(lambda a, b: a + b)
29 outflow_rdd = balance_rdd.map(lambda x: (x[0], x[2])).reduceByKey(lambda a, b: a + b)
30
31 # 合并流入和流出量
32 daily_balance = inflow_rdd.join(outflow_rdd).mapValues(lambda x: (x[0], -x[1]))
33
34 sorted_daily_balance = daily_balance.sortByKey()
35
36 # 收集结果并打印
37 results = sorted_daily_balance.collect()
38 for data, (inflow, outflow) in results:
    print(data, inflow, outflow)
    1
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
20130730 47481243.0 13048582.0
20130731 54569637.0 9534040.0
```

1.2 统计每个城市总流量前 3 高的用户

核心代码：

解析CSV文件，转换为RDD并解析数据，按用户ID分组，并计算每个用户在2014年8月的活跃天数，过滤出活跃天数至少为5天的用户，计算去重后的活跃用户总数。

```
# 解析CSV文件，提取user_id, report_date
def parse_line(line):
    parts = line.split(',')
    user_id = parts[0]
    report_date = parts[1]
    # 检查用户当天是否活跃
    is_active = int(parts[5]) > 0 or int(parts[8]) > 0
    return (user_id, report_date, is_active)

# 转换为RDD并解析数据
user_activity_rdd = data.map(parse_line).filter(lambda x: x[2])

# 按用户ID分组，并计算每个用户在2014年8月的活跃天数
active_days_rdd = user_activity_rdd.filter(lambda x: x[1].startswith('201408')) \
    .map(lambda x: (x[0], 1)) \
    .reduceByKey(lambda a, b: a + b)

# 过滤出活跃天数至少为5天的用户
active_users_rdd = active_days_rdd.filter(lambda x: x[1] >= 5) \
    .map(lambda x: x[0])

# 计算活跃用户总数
active_user_count = active_users_rdd.distinct().count()
```

```
28 # 计算活跃用户总数
29
30 # 计算活跃用户总数
31 active_user_count = active_users_rdd.distinct().count()
32
33 print(f"<活跃用户总数>{active_user_count}<活跃用户总数>")
34
35 # 保存结果到文件
```

问题 输出 调试控制台 终端 端口

+ v bash [] [] ... ^

● zhangyilu@zhangyilu-VMware-Virtual-Platform:~/bigdata/task4\$ python3 /home/zhangyilu/bigdata/task4/p2.py
<活跃用户总数>3175<活跃用户总数>

2.1 按城市统计 2014 年 3 月 1 日的平均余额

创建一个SparkSession，读取数据，过滤出3月1的数据，将用户城市信息和余额信息通过er_id 进行连接，对连接后的数据集按城市分组，并计算每个城市的平均余额，降序排列。

问题：把地址识别成hdfs地址然后报错说找不到

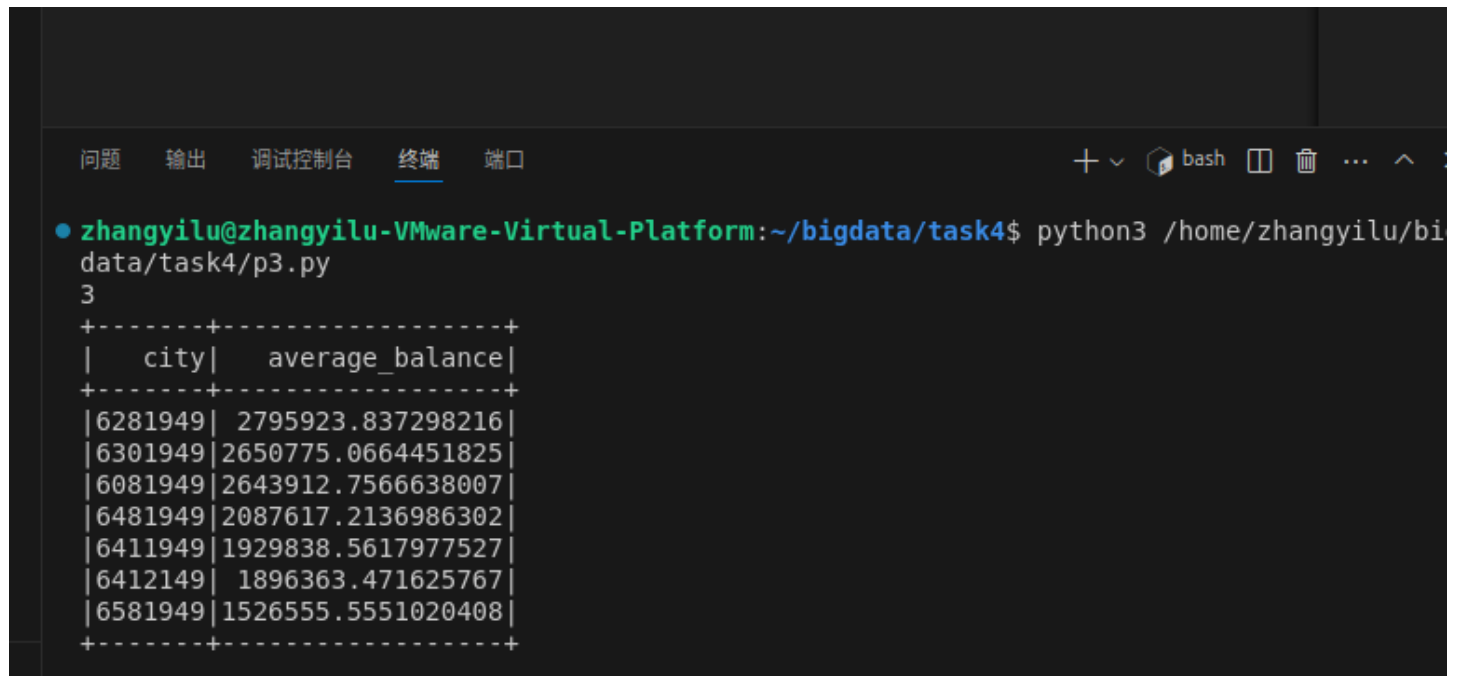
```

* zhangyilu@zhangyilu-VMware-Virtual-Platform:~/bigdata/task4$ python3 /home/zhangyilu
data/task4/p3.py
Traceback (most recent call last):
  File "/home/zhangyilu/bigdata/task4/p3.py", line 7, in <module>
    user_profiles = spark.read.csv("user_profile_table.csv", header=True, inferSchem
ue)
    ~~~~~
    ^^^^^
File "/usr/local/spark/python/pyspark/sql/readwriter.py", line 740, in csv
    return self._df(self._jreader.csv(self._spark._sc._jvm.PythonUtils.toSeq(path)))
    ~~~~~
    ~~~~~
File "/usr/local/spark/python/lib/py4j-0.10.9.7-src.zip/py4j/java_gateway.py", lin
22, in _call
File "/usr/local/spark/python/pyspark/errors/exceptions/captured.py", line 185, in

```

```
1 user_profiles =  
  spark.read.csv("file:///home/zhangyilu/downloads/user_profile_table.csv",  
    header=True, inferSchema=True)  
2 user_balances =  
  spark.read.csv("file:///home/zhangyilu/downloads/user_balance_table.csv",  
    header=True, inferSchema=True)
```

运行结果：



The terminal window shows the execution of a Python script. The prompt is `zhangyilu@zhangyilu-VMware-Virtual-Platform:~/bigdata/task4$`. The command executed is `python3 /home/zhangyilu/bigdata/task4/p3.py`. The output is a table with 2 columns: `city` and `average_balance`. The table contains 8 rows of data, including a header row and 7 data rows.

```
zhangyilu@zhangyilu-VMware-Virtual-Platform:~/bigdata/task4$ python3 /home/zhangyilu/bigdata/task4/p3.py  
3  
+-----+-----+  
|  city|  average_balance|  
+-----+-----+  
|6281949| 2795923.837298216|  
|6301949|2650775.0664451825|  
|6081949|2643912.7566638007|  
|6481949|2087617.2136986302|  
|6411949|1929838.5617977527|  
|6412149| 1896363.471625767|  
|6581949|1526555.5551020408|  
+-----+-----+
```

2.2 统计每个城市总流量前 3 高的用户

核心代码：

```

spark = SparkSession.builder.appName("TopUsersByTraffic").getOrCreate()

# 读取数据
user_profile_df = spark.read.csv("file:///home/zhangyilu/downloads/user_profile_table.csv", header=True, inferSchema=True)
user_balance_df = spark.read.csv("file:///home/zhangyilu/downloads/user_balance_table.csv", header=True, inferSchema=True)
user_profile_df = user_profile_df.alias("user_profile")

# 过滤出2014年8月的数据
user_balance_august_df = user_balance_df.filter((user_balance_df.report_date >= "20140801") & (user_balance_df.report_date <= "20140831"))

# 计算每个用户在2014年8月的总流量
user_traffic_df = user_balance_august_df.groupBy("user_id").agg(sum("total_purchase_amt").alias("total_purchase_amt"),
                                                                sum("total_redeem_amt").alias("total_redeem_amt"))

user_traffic_df = user_traffic_df.withColumn("total_traffic", col("total_purchase_amt") + col("total_redeem_amt"))
user_traffic_df = user_traffic_df.alias("user_traffic")
#user_traffic_df.show(20)
user_traffic_with_city_df = user_traffic_df.join(user_profile_df, user_traffic_df.user_id == user_profile_df.user_id, "inner") \
    .select(
        "user_traffic.user_id", # 明确指定user_traffic中的user_id
        "user_traffic.total_traffic",
        "user_profile.city",    # 明确指定user_profile中的city
    )

#user_traffic_with_city_df.show(20)
# 定义窗口规范
windowSpec = Window.partitionBy("city").orderBy(col("total_traffic").desc())

# 使用窗口函数来获取每个城市的前3名
top_users_by_city_df = user_traffic_with_city_df.withColumn(
    "row_num", row_number().over(windowSpec)
).filter(col("row_num") <= 3).select("city", "user_id", "total_traffic").show(30)

```

问题：报错有重复user_id，没法正确选择

```

File "/usr/local/spark/python/pyspark/errors/exceptions/captured.py", line 132, in __getattr__
    raise converted from None
pyspark.errors.exceptions.captured.AnalysisException: [AMBIGUOUS_REFERENCE] Reference `user_id` is ambiguous, could be: [`user_id`, `user_id`].

```

解决：明确指定

```

1 user_traffic_with_city_df = user_traffic_df.join(user_profile_df,
2 user_traffic_df.user_id == user_profile_df.user_id, "inner") \
3     .select(
4         "user_traffic.user_id", # 明确指
5         "user_traffic.total_traffic",
6         "user_profile.city",    # 明确指
    )

```

运行结果：

```

• zhangyilu@zhangyilu-VMware-Virtual-Platform
.py
+-----+-----+-----+
|   city|user_id|total_traffic|
+-----+-----+-----+
|6081949|  27235|   108475680|
|6081949|  27746|    76065458|
|6081949|  18945|    55304049|
|6281949|  15118|   149311909|
|6281949|  11397|   124293438|
|6281949|  25814|   104428054|
|6301949|   2429|   109171121|
|6301949|  26825|    95374030|
|6301949|  10932|    74016744|
|6411949|   662|    75162566|
|6411949|  21030|   49933641|
|6411949|  16769|   49383506|
|6412149|  22585|   200516731|
|6412149|  14472|   138262790|
|6412149|  25147|    70594902|
|6481949|  12026|    51161825|
|6481949|   670|    49626204|
|6481949|  14877|   34488733|
|6581949|   9494|   38854436|
|6581949|  26876|   23449539|

```

3. SparkML

核心代码：

1. 初始化SparkSession：

2. 读取数据：

3. 数据预处理：

- 将 `report_date` 列转换为日期格式。
- 选择 `report_date`、`total_purchase_amt`（总购买金额）和 `total_redeem_amt`（总赎回金额）三列作为后续分析的基础。
- 对数据进行聚合，按 `report_date` 分组，并计算每天的总购买金额和总赎回金额。
- 添加 `day_of_month`（月份中的第几天）和 `day_of_week`（一周中的第几天）作为特征。
- 使用 `na.fill` 方法填充缺失值，将缺失的购买金额和赎回金额设为0。

4. 特征向量化：

- 使用 `VectorAssembler` 将 `day_of_month` 和 `day_of_week` 两个特征合并为一个特征向量，这是机器学习模型所需的输入格式。

5. 分割数据集：

- 根据 `report_date` 将数据集分割为训练集（2014年9月1日之前的数据）和测试集（2014年9月1日及之后的数据）。然而，在代码中实际使用的是训练集来训练模型，并没有展示如何使用测试集进行模型评估。

6. 训练随机森林模型：

- 使用训练集分别训练两个随机森林回归模型，一个用于预测总购买金额，另一个用于预测总赎回金额。

7. 创建未来日期的特征数据：

- 生成一个包含未来30天日期的列表，并将其转换为DataFrame格式。然后，计算这些日期的 `day_of_month` 和 `day_of_week` 特征，并使用 `VectorAssembler` 将它们转换为特征向量。

8. 预测：

- 使用训练好的模型对未来30天的总购买金额和总赎回金额进行预测。

9. 处理预测结果：

- 将预测结果中的日期格式调整回 `yyyyMMdd` 格式，并将预测值转换为 `bigint` 类型
- 将购买金额和赎回金额的预测结果合并为一个DataFrame，并按日期排序。

10. csv文件存放到指定路径

```
# 转换日期格式
daily_data = daily_data.withColumn("report_date", to_date("report_date", "yyyyMMdd"))

# 选择需要的列
daily_data = daily_data.select("report_date", "total_purchase_amt", "total_redeem_amt")

# 聚合数据
user_data = daily_data.groupBy("report_date") \
    .agg(F.sum("total_purchase_amt").alias("total_purchase_amt"), \
         F.sum("total_redeem_amt").alias("total_redeem_amt"))

# 特征工程
user_data = user_data.withColumn("day_of_month", F.dayofmonth("report_date"))
user_data = user_data.withColumn("day_of_week", F.dayofweek("report_date"))

# 处理缺失值
user_data = user_data.na.fill({'total_purchase_amt': 0, 'total_redeem_amt': 0})

# 特征向量化
assembler = VectorAssembler(inputCols=["day_of_month", "day_of_week"], outputCol="features")
user_data = assembler.transform(user_data)


# 分割数据集
train_data = user_data.filter(user_data["report_date"] < "2014-09-01")
test_data = user_data.filter(user_data["report_date"] >= "2014-09-01")

# 训练随机森林模型
rf_model_purchase = RandomForestRegressor(featuresCol="features", labelCol="total_purchase_amt")
rf_model_redeem = RandomForestRegressor(featuresCol="features", labelCol="total_redeem_amt")
purchase_fit = rf_model_purchase.fit(train_data)
redeem_fit = rf_model_redeem.fit(train_data)

# 创建未来日期的特征数据
future_dates = [{"date": f"2014-09-{i+1}"} for i in range(30)]
future_data = spark.createDataFrame(future_dates)
future_data = future_data.withColumn("day_of_month", F.dayofmonth("date"))
future_data = future_data.withColumn("day_of_week", F.dayofweek("date"))
future_data = assembler.transform(future_data)


# 预测
predictions_purchase = purchase_fit.transform(future_data)
predictions_redeem = redeem_fit.transform(future_data)
```

运行结果：

tc_comp_predict_table.csv >  part-00000-61893d88-d61f-493a-8d80-10093f78d9d1-c000.csv

	report_date	purchase	redeem
1	20140901	383132799	230148959
2	20140902	357262438	259911823
3	20140903	394419372	270283307
4	20140904	390134283	254920640
5	20140905	331917654	267985287
6	20140906	247745925	167740002
7	20140907	266813514	202748688
8	20140908	428683423	281444833
9	20140909	423259412	299067162
10	20140910	419706975	332669272
11	20140911	415535118	316883024
12	20140912	343033315	298314023
13	20140913	280366378	179640019
14	20140914	300685922	201548009
15	20140915	431960405	328818359
16	20140916	438055071	339577060
17	20140917	420402480	349455486
18	20140918	417379513	330408335
19	20140919	327240025	300946226
20	20140920	276840821	186974484
21	20140921	301966742	232978363
22	20140922	419994568	366274609
23	20140923	429068786	335827342
24	20140924	424766277	355229543
25	20140925	414627936	344712079
26	20140926	340729469	361623958
27	20140927	279620975	185430826
28	20140928	276752391	244830783
29	20140929	438289876	366303821
30	20140930	311995155	310673584

长期赛

 日期: 2024-12-26 19:23:44

分数: 81.7557