

金融大数据实验四

22275038 杨念

一、下载安装Spark

1、安装pyspark

```
pip install pyspark
```

```
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.7 pyspark-3.5.3
hadoop@shotaroilc-virtual-machine:~$
```

2、下载配置Spark

在官网下载Spark3.5.3，放进Downloads目录。

进入Downloads目录并进行解压，放进/usr/local下：

```
sudo tar -zxvf ./spark-3.5.3-bin-hadoop3.tgz -C /usr/local
```

进入/usr/local目录，将spark-3.5.3的名字修改为spark：

```
sudo mv ./spark-3.5.3-bin-hadoop3/ ./spark
```

赋予修改权限：

```
sudo chown -R hadoop ./spark
```

配置环境变量

修改bashrc文件，修改结束后 `source ~/.bashrc`

```
hadoop@shotaroilc-virtual-machine: /usr/local
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export HIVE_HOME=/usr/local/hive
export PATH=$HIVE_HOME/bin:$PATH
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_PREFIX=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
```

配置spark

配置 `spark-env.sh`

```
cp $SPARK_HOME/conf/spark-env.sh.template $SPARK_HOME/conf/spark-env.sh

vim $SPARK_HOME/conf/spark-env.sh

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export SPARK_MASTER_HOST=localhost
export SPARK_LOCAL_IP=localhost
export SPARK_MASTER_PORT=7077
```

配置 `spark-defaults.conf`

```
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-
defaults.conf

nano $SPARK_HOME/conf/spark-defaults.conf

spark.master                spark://localhost:7077
spark.driver.memory         2g
spark.executor.memory       4g
spark.eventLog.enabled      false
```

配置 `slave` 文件

```
vim $SPARK_HOME/conf/slaves

localhost
```

启动Spark集群

启动 Spark Master:

```
$SPARK_HOME/sbin/start-master.sh
```

可以访问网页 <http://localhost:8080> 来检查 Spark Master 的状态。

启动 Spark Worker:

启动 Spark Worker 节点并连接到 Master 节点:

```
$SPARK_HOME/sbin/start-worker.sh spark://localhost:7077
```

可以访问网页 <http://localhost:8080> 来查看 Worker 是否成功注册到 Master。

Spark Master at spark://localhost:7077

URL: spark://localhost:7077

Alive Workers: 1

Cores in use: 4 Total, 0 Used

Memory in use: 6.7 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241222233638-127.0.0.1-42361	127.0.0.1:42361	ALIVE	4 (0 Used)	6.7 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

终止Spark集群

终止Worker：

```
$SPARK_HOME/sbin/stop-worker.sh
```

终止 Master：

```
$SPARK_HOME/sbin/stop-master.sh
```

二、实验内容

（一）任务一：Spark RDD编程

1、查询特定日期的资金流入和流出情况

（1）实验要求

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。

格式： <日期> <资金流入量><资金流出量>

示例： 20130701 32488348 5525022

（2）实验设计

资金流入量= total_purchase_amt

资金流出= total_redeem_amt

代码写在 1_1.py 文件里面， 需要使用到 pyspark 里面的库。

将需要用到的 user_balance_table.csv 放进和py文件同一个文件里， 就可以写文件相对路径了。

得到的结果放进 output_task1.txt 文件里

```
from pyspark import SparkContext
from datetime import datetime

sc = SparkContext("local", "UserBalanceAnalysis")

# 加载数据
data = sc.textFile("user_balance_table.csv")
```

```

# 去掉表头
header = data.first()
rows = data.filter(lambda line: line != header)

# 解析数据
def parse_line(line):
    fields = line.split(",")
    report_date = fields[1] # 日期
    total_purchase_amt = float(fields[4]) # 资金流入
    total_redeem_amt = float(fields[8]) # 资金流出
    return (report_date, (total_purchase_amt, total_redeem_amt))

# 计算每天的资金流入和流出
daily_flow = rows.map(parse_line) \
    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1])) \
    .sortByKey()

# 格式化输出
result_task1 = daily_flow.map(lambda x: f"{x[0]} {int(x[1][0])} {int(x[1][1])}")
result_task1.saveAsTextFile("output_task1.txt")

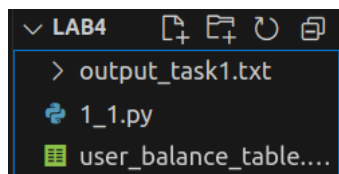
```

(3) 程序运行及结果

```

hadoop@shotaroilc-virtual-machine:~/Documents/lab4$ /usr/bin/python3 /home/hadoop/Documents/lab4/1_1.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/23 13:12:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
hadoop@shotaroilc-virtual-machine:~/Documents/lab4$ 

```



output_task1.txt里面有分段处理五个文件，因为user_balance_table.csv的数据量太大，进行分段处理，输出结果是五个文件的拼接：part-00000~part-00004就是输出文件。

```
▼ output_task1.txt
≡ _SUCCESS
≡ ._SUCCESS.crc
≡ .part-00000.crc
≡ .part-00001.crc
≡ .part-00002.crc
≡ .part-00003.crc
≡ .part-00004.crc
≡ part-00000
≡ part-00001
≡ part-00002
≡ part-00003
≡ part-00004
```

整理之后存入 `result_task1_1.txt` 文件里面

任意取一段输出文件里的结果作为代表展示：

```
45 20130814 30541167 15031683
46 20130815 42421222 52659327
47 20130816 41683536 33739844
48 20130817 17670519 4674983
49 20130818 59703092 15218300
50 20130819 109184209 28339260
51 20130820 53675509 30131225
52 20130821 38184446 29983342
```

2、活跃用户分析：

(1) 实验要求

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。

输出格式：<活跃用户总数>

(2) 实验设计

本实验主要是找出日期列为202408开头的日期，所以以这个条件进行筛选，就可以得到最后的结果。

```
# 解析数据，提取用户和日期
def parse_user_date(line):
    fields = line.split(",")
    user_id = fields[0] # 用户 ID
    report_date = fields[1] # 日期
```

```

        return (user_id, report_date)

# 过滤出 2014 年 8 月的数据
def is_august_2014(report_date):
    date_obj = datetime.strptime(report_date, "%Y%m%d")
    return date_obj.year == 2014 and date_obj.month == 8

# 统计每个用户在 2014 年 8 月的活跃天数
active_users = rows.map(parse_user_date) \
    .filter(lambda x: is_august_2014(x[1])) \
    .distinct() \
    .map(lambda x: (x[0], 1)) \
    .reduceByKey(lambda x, y: x + y)

# 筛选出活跃用户（至少 5 天记录）
active_user_count = active_users.filter(lambda x: x[1] >= 5).count()

# 输出结果
with open("output_task2.txt", "w") as f:
    f.write(f"{active_user_count}")

```

(3) 程序运行及结果

因为user_balance_table.csv的数据量太大，进行分阶段处理，输出结果是output_task2.txt文件。

```

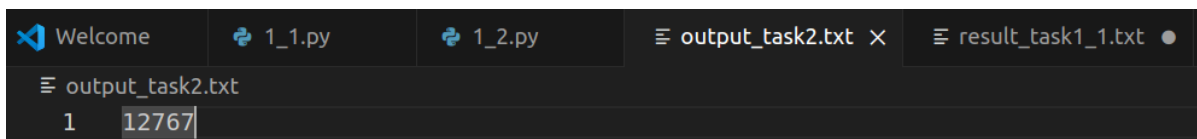
/usr/bin/python3 /home/hadoop/Documents/lab4/1_2.py
hadoop@shotaroilc-virtual-machine:~/Documents/lab4$ /usr/bin/python3 /home/hadoop/Documents/lab4/1_2.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/23 13:31:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
[Stage 1:>                                     (0 + 1) / 5]

```

```

• hadoop@shotaroilc-virtual-machine:~/Documents/lab4$ /usr/bin/python3 /home/hadoop/Documents/lab4/1_2.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/23 13:31:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
○ hadoop@shotaroilc-virtual-machine:~/Documents/lab4$

```



(二) 任务二：Spark SQL编程

1、按城市统计2014年3月1日的平均余额

(1) 实验要求

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。

输出格式： <城市ID> <平均余额>

示例： 1001234 150000

6081949 140000

(2) 实验设计

1. 筛选 `report_date` 为 20140301 的数据。
2. 按 `category1`（假设为城市ID）分组，计算每个城市的平均 `tBalance`。
3. 按平均余额降序排列。

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("City Average Balance") \
    .config("spark.eventLog.enabled", "false") \
    .getOrCreate()

# 加载数据
user_balance_path = "/home/hadoop/Documents/lab4/user_balance_table.csv"
user_profile_path = "/home/hadoop/Documents/lab4/user_profile_table.csv"

# 读取CSV文件，自动识别列名（header="true"）
user_balance_df = spark.read.option("header", "true").csv(user_balance_path)
user_profile_df = spark.read.option("header", "true").csv(user_profile_path)

# 注册DataFrame为临时视图
user_balance_df.createOrReplaceTempView("user_balance_table")
user_profile_df.createOrReplaceTempView("user_profile_table")

# 执行Spark SQL查询
result_df = spark.sql("""
    SELECT
        p.city,
        AVG(CAST(b.tBalance AS DOUBLE)) AS avg_balance
    FROM
        user_balance_table b
    JOIN
        user_profile_table p
    ON
        b.user_id = p.user_id
    WHERE
        b.report_date = '20140301'
    GROUP BY
        p.city
    ORDER BY
        avg_balance DESC
""")

result_df.show()
spark.stop()
```

(3) 程序运行及结果

```
hadoop@shotaroilc-virtual-machine:~/Documents/lab4$ /bin/python3 /home/hadoop/Documents/lab4/2_1.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/23 14:46:36 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
+-----+-----+
| city|      avg_balance|
+-----+-----+
|6281949| 2795923.837298216|
|6301949|2650775.0664451825|
|6081949|2643912.7566638007|
|6481949|2087617.2136986302|
|6411949|1929838.5617977527|
|6412149| 1896363.471625767|
|6581949|1526555.5551020408|
+-----+-----+

hadoop@shotaroilc-virtual-machine:~/Documents/lab4$
```

2、统计每个城市总流量前3高的用户

(1) 实验要求

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。

输出格式：<城市ID> <用户ID> <总流量>

(2) 实验设计

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("CityTopUsers") \
    .config("spark.eventLog.enabled", "false") \
    .getOrCreate()

user_balance_path = "/home/hadoop/Documents/lab4/user_balance_table.csv"
user_profile_path = "/home/hadoop/Documents/lab4/user_profile_table.csv"

# 读取数据到 DataFrame
user_balance_df = spark.read.option("header", "true").csv(user_balance_path)
user_profile_df = spark.read.option("header", "true").csv(user_profile_path)

# 注册 DataFrame 为临时视图
user_balance_df.createOrReplaceTempView("user_balance")
user_profile_df.createOrReplaceTempView("user_profile")

# 执行 SQL 查询
result = spark.sql("""

WITH user_flow AS (
    SELECT
        p.city AS city_id,
        b.user_id,
        SUM(b.total_purchase_amt + b.total_redeem_amt) AS total_flow
    FROM
        user_balance b
    JOIN
        user_profile p
    ON
```



```

        b.user_id = p.user_id
    WHERE
        b.report_date LIKE '201408%'
    GROUP BY
        p.City, b.user_id
),
ranked_users AS (
    SELECT
        city_id,
        user_id,
        total_flow,
        RANK() OVER (PARTITION BY city_id ORDER BY total_flow DESC) AS rank
    FROM
        user_flow
)
SELECT
    city_id,
    user_id,
    total_flow
FROM
    ranked_users
WHERE
    rank <= 3
ORDER BY
    city_id,
    rank;
""")

result.show()
spark.stop()

```

(3) 程序运行及结果

```

hadoop@shotaroilc-virtual-machine:~/Documents/Lab4$ /bin/python3 /home/hadoop/Documents/lab4/2_2.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/23 15:02:41 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
+-----+-----+-----+
|city_id|user_id| total_flow|
+-----+-----+-----+
|6081949| 27235| 1.0847568E8|
|6081949| 27746| 7.6065458E7|
|6081949| 18945| 5.5304049E7|
|6281949| 15118| 1.49311909E8|
|6281949| 11397| 1.24293438E8|
|6281949| 25814| 1.04428054E8|
|6301949| 2429| 1.09171121E8|
|6301949| 26825| 9.537403E7|
|6301949| 10932| 7.4016744E7|
|6411949| 662| 7.5162566E7|
|6411949| 21030| 4.9933641E7|
|6411949| 16769| 4.9383506E7|
|6412149| 22585| 2.00516731E8|
|6412149| 14472| 1.3826279E8|
|6412149| 25147| 7.0594902E7|
|6481949| 12026| 5.1161825E7|
|6481949| 670| 4.9626204E7|
|6481949| 14877| 3.4488733E7|
|6581949| 9494| 3.8854436E7|
|6581949| 26876| 2.3449539E7|
+-----+-----+-----+
only showing top 20 rows
hadoop@shotaroilc-virtual-machine:~/Documents/Lab4$

```

(三) 任务三：Spark ML编程

要解决这个问题，我们将采用以下步骤，并在每个步骤中使用特定的方法：

1. 数据预处理

使用Spark的 `read.csv` 方法加载四个CSV文件，检查缺失值、异常值，并进行处理，将日期字符串转换为日期类型，并从中提取年、月、日等特征，将用户信息、用户行为数据、收益率表和银行间拆放利率表进行合并。

2. 构建训练集和测试集

- **数据划分**：将数据集分为训练集和测试集，通常比例为70%训练集和30%测试集。

3. 模型选择与训练

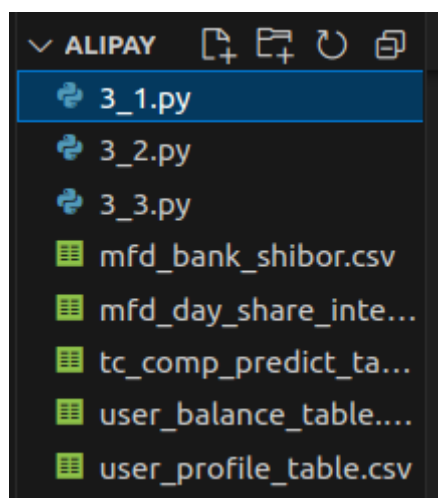
- **选择模型**：考虑到问题的复杂性和非线性特征，我们选择随机森林作为主要模型。随机森林能够处理大量特征且对于非线性关系有很好的捕捉能力。
- **训练模型**：使用Spark MLlib中的 `RandomForestRegressor` 对训练数据进行拟合。

使用优化后的模型对2014年9月每天的申购与赎回总额进行预测。

将预测结果存储到 `tc_comp_predict_table.csv` 文件中，使用Spark的 `write.csv` 方法。

通过这些步骤，我们可以构建一个机器学习模型来预测2014年9月每天的申购与赎回总额，并将结果存储到CSV文件中。这个过程涉及到数据预处理、模型选择、训练、评估、优化和预测等多个方面。

```
预测结果已保存到 tc_comp_predict_table.csv
hadoop@shotaroilc-virtual-machine:~/Documents/alipay$
```



4. 遇到的问题

如果使用的数据集变小，那么会使预测的效果不如使用数据集大的情况，比如我也尝试了只考虑其中几个月的数据，发现预测结果不如之前的情况：

○ **日期:** 2024-12-24 16:52:45

分数: 91.1171

○ **日期:** 2024-12-24 16:51:11

分数: 99.8775

○ **日期:** 2024-12-23 18:26:26

分数: 99.8701

○ **日期:** 2024-12-23 18:25:42

分数:

ERROR Bad input file

○ 暂无更多数据

① Note

上传csv之前，需要把表第一行的列名删去，不然会显示是错误的格式。