

实验四

spark安装

- 首先从<http://spark.apache.org/downloads.html>下载了3.5.3的spark
- 解压spark文件

```
cd /usr/local #我是讲spark放在了文件下
sudo tar -xvzf spark-3.5.3-bin-hadoop3.tgz
sudo mv spark-3.3.0-bin-hadoop3 spark
```

- 设置环境变量

```
nano ~/.bashrc
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
source ~/.bashrc
```

- 配置Spark以确保其能与Hadoop集成

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

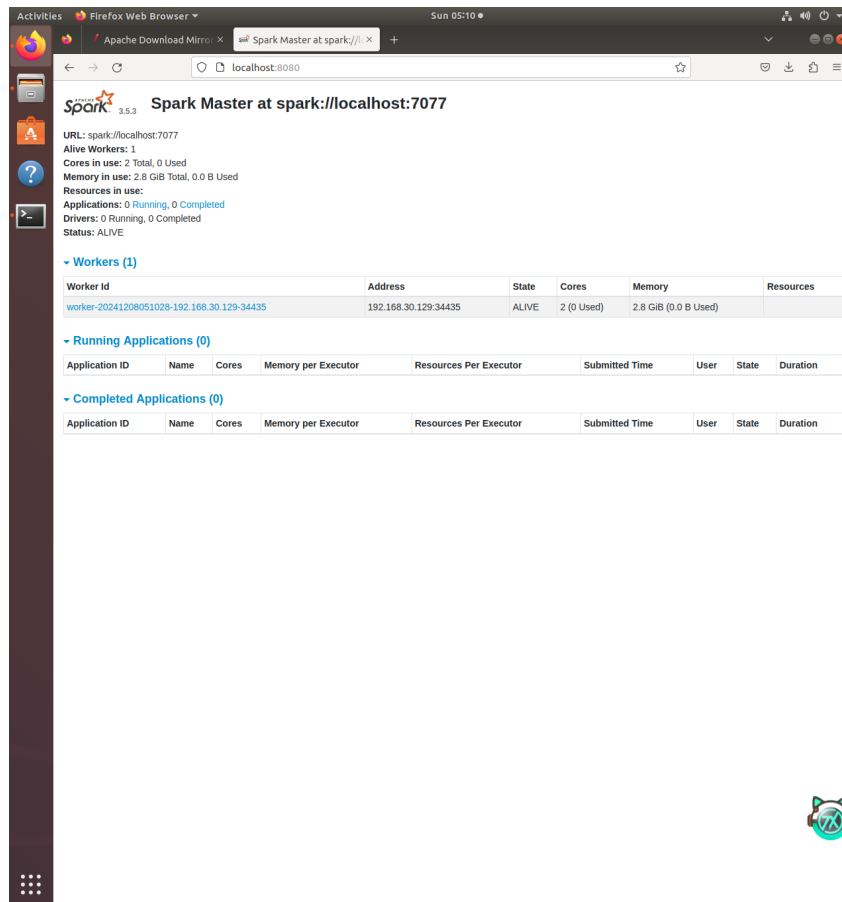
编辑 `spark-env.sh` , 添加Hadoop配置目录和Spark master主机地址:

```
sudo nano $SPARK_HOME/conf/spark-env.sh
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export SPARK_MASTER_HOST=localhost
```

- 启动Spark的Master和一个Worker:

```
start-master.sh
start-worker.sh spark://localhost:7077
```

- 在浏览器中访问 `http://localhost:8080` , 查看Spark Master的Web界面以确认Spark集群状态。



- 运行一个测试作业，如计算 π 的值，验证安装：

```
$SPARK_HOME/bin/run-example SparkPi 10
```

```
24/12/08 05:11:03 INFO Executor: Finished task 9.0 in stage 0.0 (TID 9). 1012 bytes result sent to driver
24/12/08 05:11:03 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 113 ms on 192.168.30.129 (executor driver) (9/10)
24/12/08 05:11:03 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 72 ms on 192.168.30.129 (executor driver) (10/10)
24/12/08 05:11:03 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
24/12/08 05:11:03 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.331 s
24/12/08 05:11:03 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/08 05:11:03 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
24/12/08 05:11:03 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.486839 s
Pi is roughly 3.141111141111141
24/12/08 05:11:03 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/12/08 05:11:03 INFO SparkUI: Stopped Spark web UI at http://192.168.30.129:4040
24/12/08 05:11:03 INFO BlockManagerInfo: Removed broadcast_0_piece0 on 192.168.30.129:41179 in memory (size: 2.3 KiB, free: 434.4 MiB)
24/12/08 05:11:03 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/12/08 05:11:03 INFO MemoryStore: MemoryStore cleared
24/12/08 05:11:03 INFO BlockManager: BlockManager stopped
24/12/08 05:11:03 INFO BlockManagerMaster: BlockManagerMaster stopped
24/12/08 05:11:03 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/12/08 05:11:03 INFO SparkContext: Successfully stopped SparkContext
24/12/08 05:11:03 INFO ShutdownHookManager: Shutdown hook called
24/12/08 05:11:03 INFO ShutdownHookManager: Deleting directory /tmp/spark-8698211a-5b0c-4487-b960-2d329d442c69
24/12/08 05:11:03 INFO ShutdownHookManager: Deleting directory /tmp/spark-60bf5e8b-f744-4270-abc1-52da99753a30
hadoop@ubuntu: /usr/local$
```

可以看到成功计算出结果，成功启动。

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。

```
#创建一个新目录来存放数据和代码
mkdir /usr/local/spark_work
mv ~/Downloads/user_balance_table.csv /usr/local/spark_work/
```

在 `/usr/local/spark_work` 目录中，创建一个新的Python脚本文件

```
cd /usr/local/spark_work
sudo nano /usr/local/spark_work/user_balance_analysis.py
```

使用 `spark-submit` 命令来运行Spark脚本:

```
spark-submit user_balance_analysis.py
```

代码:

```
from pyspark import SparkContext, SparkConf

def parse_line(line):
    fields = line.split(',')
    try:
        report_date = fields[1]
        total_purchase_amt = int(fields[4])
        total_redeem_amt = int(fields[8])
        return (report_date, total_purchase_amt, total_redeem_amt)
    except ValueError:
        return None

def main():
    conf = SparkConf().setAppName("User Balance Analysis")
    sc = SparkContext(conf=conf)
    lines =
sc.textFile("hdfs://localhost:9000/user/hadoop/user_balance_table.csv")
    parsed_rdd = lines.map(parse_line).filter(lambda x: x is not None)
    daily_totals = parsed_rdd.map(lambda x: (x[0], (x[1], x[2])))\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] +
b[1]))

    results = daily_totals.map(lambda x: f"{x[0]} {x[1][0]} {x[1][1]}")
    for result in results.collect():
        print(result)

    sc.stop()

if __name__ == "__main__":
    main()
```

- 使用 `parse_line` 函数解析每一行数据, 提取 `report_date` (报告日期), `total_purchase_amt` (总购买金额), 和 `total_redeem_amt` (总赎回金额)。如果转换过程中出现错误 (例如数据格式不正确), 该行数据将被过滤掉。
- 将解析后的数据映射成键值对形式, 键是 `report_date`, 值是一个包含 `total_purchase_amt` 和 `total_redeem_amt` 的元组。
- 使用 `reduceByKey` 方法对同一天的数据进行聚合, 计算每天的总购买和总赎回金额。

运行结果:

```
Activities Terminal Sun 05:49
hadoop@ubuntu: /usr/local/spark_work

24/12/08 05:48:26 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
24/12/08 05:48:26 INFO DAGScheduler: ResultStage 1 (collect at /usr/local/spark_work/user_balance_analysis.py:31) finished in 0.359 s
24/12/08 05:48:26 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/08 05:48:26 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
24/12/08 05:48:26 INFO DAGScheduler: Job 0 finished: collect at /usr/local/spark_work/user_balance_analysis.py:31, took 13.782243 s
24/12/08 05:48:26 INFO SparkContext: SparkContext is stopping with exitCode 0.
20140805 394780870 221706539
20140808 233903717 311648757
20140811 331550471 418603336
20140814 257702660 211939431
20140820 308378692 202452782
20140823 141412027 199377531
20140826 306945089 285478563
20140829 267554713 273756380
20140830 199708772 196374134
20140827 302194801 468164147
20140821 251763517 219963356
20140818 298499146 259169016
20140815 244551620 236516007
20140812 258493673 309754858
20140803 173825397 127112517
20140728 371762756 345986909
20140725 181641088 262874791
20140722 243084133 369043423
20140719 210318023 155464283
20140716 394890140 234775948
20140713 179759885 199459990
20140710 283095921 326009240
20140707 272182847 317612569
20140409 383347565 289330278
20140403 363877120 266605457
20140331 398884905 423852634
20140328 225966355 405443946
20140831 275090213 292943033
20140825 309574223 312413411
20140822 246316056 179349206
20140816 215059736 219214339
20140813 261506619 303975517
20140810 259534870 189909225
20140804 330640884 322907524
20140729 228093046 303480103
20140723 265461894 308353077
20140717 253011280 298279385
20140714 254797524 284753279
20140711 208671021 240050748
20140708 224240103 340453063
20140630 334054112 456547794
20140321 282351818 259655286
20140324 313180334 437825259
20140402 355347118 272612066
20140414 309853269 415986984
20140417 355792647 265341592
20140420 191259529 161057781
20140501 193045106 143362755
20140504 303087562 413222034
20140513 275241493 257918375
20140519 259077930 293791406
20140522 344636549 251108485
20140603 270887462 385622582
20140612 332365185 236467885
20140615 166080126 116623756
20140323 167456369 186443311
20140326 272935544 450254233
20140329 160250985 155006056
20140407 196936223 176966561
20140410 386567460 286914864
20140416 387847838 255914640
20140422 285248757 268810141
20140425 220927432 227764292
20140506 318002728 341108696
```

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五天记录的用户，统计2014年8月的活跃用户总数。

代码：

```
from pyspark import SparkContext, SparkConf

def parse_line(line):
    fields = line.split(',')
    try:
        user_id = fields[0]
        report_date = fields[1]
        if '201408' in report_date:
            return (user_id, report_date)
    except IndexError:
        return None
    return None

def main():
    conf = SparkConf().setAppName("Active User Analysis")
    sc = SparkContext(conf=conf)
```

```

lines =
sc.textFile("hdfs://localhost:9000/user/hadoop/user_balance_table.csv")

user_dates = lines.map(parse_line).filter(lambda x: x is not None)
user_unique_dates = user_dates.distinct().map(lambda x: (x[0], {x[1]}))
user_aggregated_dates = user_unique_dates.reduceByKey(lambda a, b:
a.union(b))
active_users = user_aggregated_dates.filter(lambda x: len(x[1]) >= 5)
active_user_count = active_users.count()
print(f"Active users total: {active_user_count}")

sc.stop()

if __name__ == "__main__":
    main()

```

- `parse_line(line)` 函数尝试解析文本，将其分割为字段，并检查日期字段是否包含"201408"。如果是，它返回一个元组，包含用户ID和报告日期。如果行不能正确解析或日期不匹配，函数返回 `None`。
- `filter(lambda x: x is not None)` 移除所有 `None` 值，这些通常是解析失败或日期不符的行。
- `distinct()` 去除重复的 (user_id, report_date) 对。
- `map(lambda x: (x[0], {x[1]}))` 转换为键值对，其中键是 `user_id`，值是包含一个日期的集合。

运行结果：

```

24/12/08 05:51:38 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/08 05:51:38 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
24/12/08 05:51:38 INFO TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
24/12/08 05:51:38 INFO DAGScheduler: Job 0 finished: count at /usr/local/spark_work/user_balance_analysis_2.py:36, took 17.420267 s
Active users total: 12767
24/12/08 05:51:38 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/12/08 05:51:38 INFO SparkUI: Stopped Spark web UI at http://192.168.30.129:4040
24/12/08 05:51:38 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/12/08 05:51:39 INFO MemoryStore: MemoryStore cleared

```

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。

代码：

```

from pyspark.sql import SparkSession

def main():
    spark = SparkSession.builder.appName("Average Balance by City").getOrCreate()

    user_profile_df =
spark.read.csv("hdfs://localhost:9000/user/hadoop/user_profile_table.csv",
header=True, inferSchema=True)
    user_balance_df =
spark.read.csv("hdfs://localhost:9000/user/hadoop/user_balance_table.csv",
header=True, inferSchema=True)

    #将DataFrame注册为临时SQL视图，允许我们像操作SQL数据库表一样操作这些DataFrame。
    user_profile_df.createOrReplaceTempView("user_profiles")

```

```

user_balance_df.createOrReplaceTempView("user_balances")

result = spark.sql("""
    SELECT p.city, AVG(b.tBalance) AS avg_balance
    FROM user_profiles p
    JOIN user_balances b ON p.user_id = b.user_id
    WHERE b.report_date = '20140301'
    GROUP BY p.city
    ORDER BY avg_balance DESC
""")

result.show()

spark.stop()

if __name__ == "__main__":
    main()

```

代码解释：

- 使用 JOIN 将 user_profiles 和 user_balances 表通过 user_id 字段联接。
- 通过 WHERE 语句筛选 report_date 为2014年3月1日的记录。
- 按 city 分组，并计算每个城市的平均余额 AVG(b.tBalance)。
- 结果按平均余额降序排列。

运行结果：

```

+-----+-----+
|   city|   avg_balance|
+-----+-----+
|6281949|2795923.837298216|
|6301949|2650775.0664451825|
|6081949|2643912.7566638007|
|6481949|2087617.2136986302|
|6411949|1929838.5617977527|
|6412149| 1896363.471625767|
|6581949|1526555.5551020408|
+-----+-----+

```

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。

代码：

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum
from pyspark.sql.window import window
from pyspark.sql.functions import rank

def main():
    spark = SparkSession.builder.appName("Top 3 Users by City").getOrCreate()

```

```

user_profile_df =
spark.read.csv("hdfs://localhost:9000/user/hadoop/user_profile_table.csv",
header=True, inferSchema=True)
user_balance_df =
spark.read.csv("hdfs://localhost:9000/user/hadoop/user_balance_table.csv",
header=True, inferSchema=True)

user_profile_df.createOrReplaceTempView("user_profiles")
user_balance_df.createOrReplaceTempView("user_balances")

# SQL查询计算每个用户在2014年8月的总流量
spark.sql("""
    SELECT p.city, b.user_id,
           (SUM(b.total_purchase_amt) + SUM(b.total_redeem_amt)) AS
total_traffic
    FROM user_profiles p
    JOIN user_balances b ON p.user_id = b.user_id
    WHERE b.report_date BETWEEN '20140801' AND '20140831'
    GROUP BY p.city, b.user_id
""").createOrReplaceTempView("city_user_traffic")

# 使用窗口函数按城市分组对用户总流量进行排序，并获取每个城市的前三名用户
windowSpec = Window.partitionBy("city").orderBy(col("total_traffic").desc())
top_users_by_city = spark.sql("""
    SELECT city, user_id, total_traffic, RANK() OVER (PARTITION BY city ORDER
BY total_traffic DESC) as rank
    FROM city_user_traffic
""").filter("rank <= 3")

# 显示结果
top_users_by_city.show()

spark.stop()

if __name__ == "__main__":
    main()

```

代码解释：

- 使用 `Window.partitionBy("city").orderBy(col("total_traffic").desc())` 定义了一个按城市分组，根据总流量降序排序的窗口。
- 在查询中，对每个城市的用户总流量使用了 `rank()` 函数进行排名，并通过过滤条件选择排名前三的记录。
- 首先联结用户档案表和余额表，过滤出2014年8月的数据，计算总流量。

运行结果：

```

+-----+-----+-----+-----+
| city|user_id|total_traffic|rank|
+-----+-----+-----+-----+
|6081949| 27235| 108475680| 1|
|6081949| 27746| 76065458| 2|
|6081949| 18945| 55304049| 3|
|6281949| 15118| 149311909| 1|
|6281949| 11397| 124293438| 2|
|6281949| 25814| 104428054| 3|
|6301949| 2429| 109171121| 1|
|6301949| 26825| 95374030| 2|
|6301949| 10932| 74016744| 3|
|6411949| 662| 75162566| 1|
|6411949| 21030| 49933641| 2|
|6411949| 16769| 49383506| 3|
|6412149| 22585| 200516731| 1|
|6412149| 14472| 138262790| 2|
|6412149| 25147| 70594902| 3|
|6481949| 12026| 51161825| 1|
|6481949| 670| 49626204| 2|
|6481949| 14877| 34488733| 3|
|6581949| 9494| 38854436| 1|
|6581949| 26876| 23449539| 2|
+-----+-----+-----+-----+
only showing top 20 rows

```

任务3: Spark ML编程

```

from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, functions as F
from pyspark.sql.types import IntegerType, DateType, StructType, StructField,
DoubleType
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from datetime import datetime, timedelta

def main():
    conf = SparkConf().setAppName("Finance Purchase and Redeem Prediction")
    sc = SparkContext(conf=conf)
    spark = SparkSession(sc)

    data_path = "hdfs://localhost:9000/user/hadoop/user_balance_table.csv"
    lines = sc.textFile(data_path)
    def parse_line(line):
        fields = line.split(',')
        try:
            report_date = int(fields[1])
            total_purchase_amt = float(fields[8])
            total_redeem_amt = float(fields[13])
            return (report_date, total_purchase_amt, total_redeem_amt)
        except:
            return None

    transactions = lines.map(parse_line).filter(lambda x: x is not None)

    schema = StructType([
        StructField("report_date", IntegerType(), True),
        StructField("total_purchase_amt", DoubleType(), True),
        StructField("total_redeem_amt", DoubleType(), True)
    ])
    df = spark.createDataFrame(transactions, schema)

    vectorAssembler = VectorAssembler(inputCols=["report_date"],
outputCol="features")
    df_vector = vectorAssembler.transform(df)

    train_data, test_data = df_vector.randomSplit([0.8, 0.2], seed=42)

```



```

lr_purchase = LinearRegression(featuresCol='features',
labelCol='total_purchase_amt')
lr_redeem = LinearRegression(featuresCol='features',
labelCol='total_redeem_amt')

model_purchase = lr_purchase.fit(train_data)
model_redeem = lr_redeem.fit(train_data)

date_range = [datetime(2014, 9, 1) + timedelta(days=x) for x in range(30)]
predict_df = spark.createDataFrame([(int(d.strftime('%Y%m%d')),) for d in
date_range], ["report_date"])
predict_features = vectorAssembler.transform(predict_df)

predictions_purchase = model_purchase.transform(predict_features)
predictions_redeem = model_redeem.transform(predict_features)

predictions = predictions_purchase.select("report_date",
F.col("prediction").alias("purchase")).join(
    predictions_redeem.select("report_date",
F.col("prediction").alias("redeem")), "report_date"
)
predictions.write.csv("hdfs://localhost:9000/user/hadoop/tc_comp_predict_table.csv", header=True)

# Stop the Spark context
sc.stop()

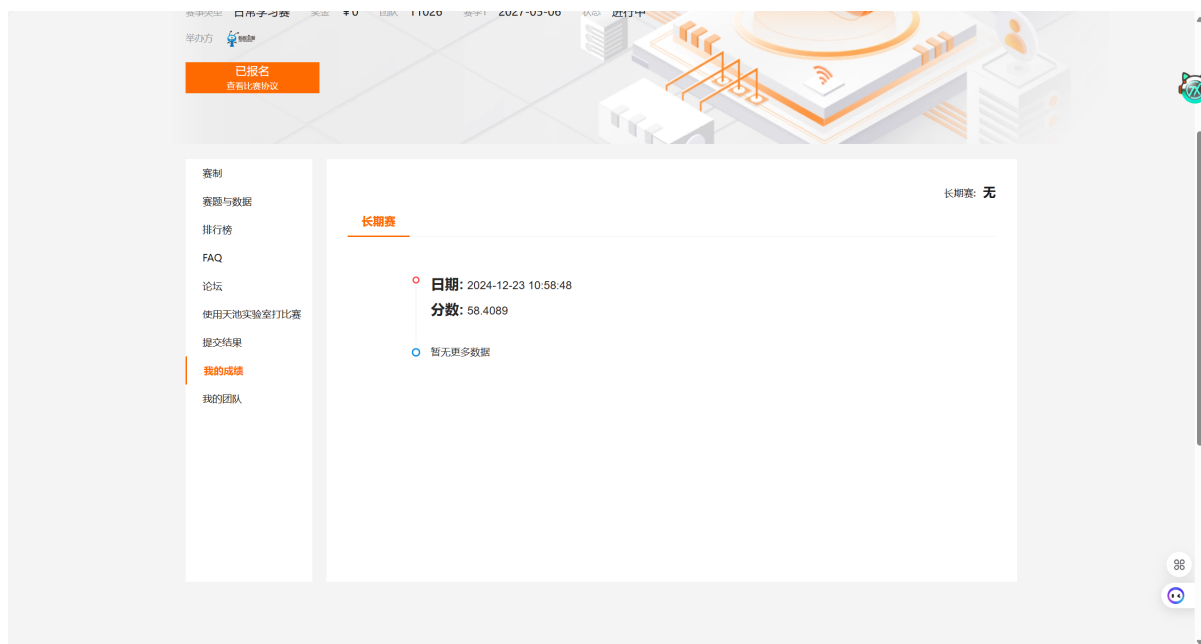
if __name__ == "__main__":
    main()

```

- 将数据随机分为训练集和测试集（80% 训练，20% 测试）。
- 分别为申购量和赎回量初始化线性回归模型，特征列为 `features`，标签列分别为 `total_purchase_amt` 和 `total_redeem_amt`。

结果：

	A	B	C	D	E	F	G	H
1	20140901	3.618E+09	345236642					
2	20140902	362454259	346054154					
3	20140903	363131475	346871463					
4	20140904	363808795	347689033					
5	20140905	364486088	348506460					
6	20140906	365163323	349323894					
7	20140907	365840662	350141343					
8	20140908	366517883	350958849					
9	20140909	367195234	351776305					
10	20140910	367872383	352593795					
11	20140911	368549592	353411141					
12	20140912	369227045	354228640					
13	20140913	369904187	355046133					
14	20140914	370581469	355863576					
15	20140915	371258667	356681103					
16	20140916	371935973	357498569					
17	20140917	372613375	358316052					
18	20140918	373290646	359133519					
19	20140919	373967757	359950803					
20	20140920	374645187	360768444					
21	20140921	375322469	361585770					
22	20140922	375999710	362403239					
23	20140923	376676842	363220788					
24	20140924	377354286	364038267					
25	20140925	378031487	364855696					
26	20140926	378708797	365673158					
27	20140927	379385931	366490590					
28	20140928	380063342	367307941					
29	20140929	380740660	368125421					
30	20140930	381417878	368943022					
31								
32								



效果比较一般，感觉有以下的问题：

1. 线性模型拟合效果较差

2. 在划分测试集和训练集可能比例不是很恰当

遇到的问题：

1. 读取文件

Spark 尝试从 HDFS 读取数据文件 `user_balance_table.csv`，但没有在指定的路径找到该文件：

```
org.apache.hadoop.mapred.InvalidInputException: Input path does not exist:  
hdfs://localhost:9000/usr/local/spark_work/user_balance_table.csv
```

所以需要将文件上传到HDFS,具体操作同实验一。