

Spark安装

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
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。

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。
- 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。

任务3: Spark ML编程

- 3-1 首先统计2013 - 2014年8月的数据，计算出每天的总申购和赎回总额
- 3-2 然后使用线性回归拟合数据，预测九月的结果

可能的改进：

Spark安装

- 从网站下载安装包
- 配置环境变量

```
export HADOOP_HOME=/home/czz/hadoop_installs/hadoop-3.4.0
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH

# Hive environment variables
export HIVE_HOME=/home/czz/Hive/hive
export PATH=$HIVE_HOME/bin:$PATH

export SPARK_HOME=/home/czz/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

export HADOOP_CONF_DIR=/home/czz/hadoop_installs/hadoop-3.4.0/etc/hadoop
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
```

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：

- 加载数据
- 按照日期进行聚合，计算每一天的总资金流入和流出
- 按日期排序

代码：

```
from pyspark import SparkContext

# 初始化SparkContext
```

```

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

# 加载数据
lines =
sc.textFile("file:///home/czz/projects/python_project/user_balance_table.csv")
header = lines.first() # 获取第一行
data = lines.filter(lambda line: line != header) # 过滤标题行

# 解析CSV数据，假设数据字段用逗号分隔
def parse_line(line):
    fields = line.split(",")
    report_date = fields[1] # 假设 report_date 在第二列
    total_purchase_amt = float(fields[4]) # 总购买金额
    total_redeem_amt = float(fields[9]) # 总赎回金额
    return (report_date, total_purchase_amt, total_redeem_amt)

# 解析数据并映射为 (日期, 资金流入, 资金流出) 元组
parsed_data = data.map(parse_line)

# 按照日期进行聚合，计算每一天的总资金流入和流出
daily_flow = parsed_data.map(lambda x: (x[0], (x[1], x[2]))) \
    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

# 按日期排序
sorted_daily_flow = daily_flow.sortByKey()

# 格式化输出结果
result = sorted_daily_flow.map(lambda x: f"{x[0]} {x[1][0]:.2f} {x[1][1]:.2f}")

# 输出到文件
output_path = "file:///home/czz/projects/python_project/output1_1"
result.saveAsTextFile(output_path)
# result.collect() # 触发计算

print(f"结果已保存到 {output_path}")

import os

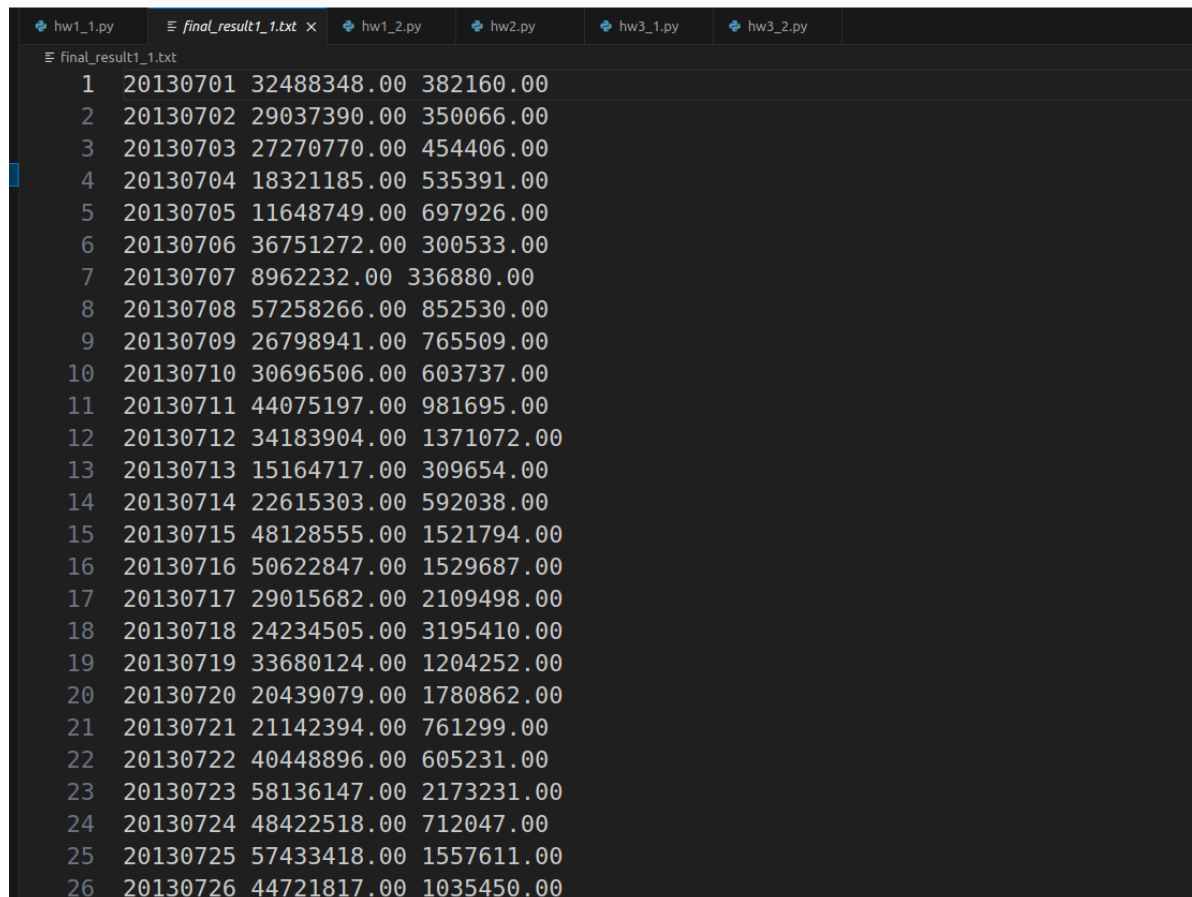
output_dir = "/home/czz/projects/python_project/output1_1"
final_output_file = "/home/czz/projects/python_project/final_result1_1.txt"

# 合并所有 part 文件
with open(final_output_file, "w") as outfile:
    for filename in sorted(os.listdir(output_dir)):
        if filename.startswith("part-"):
            with open(os.path.join(output_dir, filename), "r") as infile:
                outfile.write(infile.read())

print(f"结果已整合到 {final_output_file}")

```

输出的结果：



1	20130701	32488348.00	382160.00
2	20130702	29037390.00	350066.00
3	20130703	27270770.00	454406.00
4	20130704	18321185.00	535391.00
5	20130705	11648749.00	697926.00
6	20130706	36751272.00	300533.00
7	20130707	8962232.00	336880.00
8	20130708	57258266.00	852530.00
9	20130709	26798941.00	765509.00
10	20130710	30696506.00	603737.00
11	20130711	44075197.00	981695.00
12	20130712	34183904.00	1371072.00
13	20130713	15164717.00	309654.00
14	20130714	22615303.00	592038.00
15	20130715	48128555.00	1521794.00
16	20130716	50622847.00	1529687.00
17	20130717	29015682.00	2109498.00
18	20130718	24234505.00	3195410.00
19	20130719	33680124.00	1204252.00
20	20130720	20439079.00	1780862.00
21	20130721	21142394.00	761299.00
22	20130722	40448896.00	605231.00
23	20130723	58136147.00	2173231.00
24	20130724	48422518.00	712047.00
25	20130725	57433418.00	1557611.00
26	20130726	44721817.00	1035450.00

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。

统计2014年8月在 `user_balance_table` 中有至少 5天记录 的用户数。

思路：

- 筛选出 2014年8月 的数据（通过 `WHERE report_date` 过滤）。
- 按 `user_id` 分组，统计每个用户在该月份的记录天数。
- 过滤出记录天数 ≥ 5 的用户。
- 统计符合条件的用户总数。

代码：

```
from pyspark import SparkContext

# 初始化 SparkContext
sc = SparkContext("local", "User Balance Analysis")

# 加载数据（文件路径替换为实际路径）
data_path = "file:///home/czz/projects/python_project/user_balance_table.csv"
rdd = sc.textFile(data_path)

# 解析数据，忽略表头
header = rdd.first()
data_rdd = rdd.filter(lambda line: line != header).map(lambda line:
line.split(','))
```

```

# 提取需要的字段: report_date, total_purchase_amt, total_redeem_amt
# 假设字段顺序对应: report_date 在第 1 列, total_purchase_amt 在第 5 列,
total_redeem_amt 在第 9 列
# 转换为 (date, (流入量, 流出量)) 格式

## 任务 2: 活跃用户分析
# 提取 user_id 和日期, 构建 (user_id, date) 对
user_date_rdd = data_rdd.map(lambda x: (x[0], x[1]))

# 过滤指定月份的数据 (2014 年 8 月)
active_user_rdd = user_date_rdd.filter(lambda x: x[1].startswith("201408"))

# 按 user_id 分组, 并统计每个用户的活跃天数
user_active_days_rdd = active_user_rdd.distinct().groupByKey().mapValues(len)

# 筛选出活跃天数 >= 5 的用户
active_users_count = user_active_days_rdd.filter(lambda x: x[1] >= 5).count()

print("\n2014 年 8 月的活跃用户总数: ")
print(active_users_count)

# 停止 SparkContext
sc.stop()

```

结果: (由于较简单, 没有保存相关的输出文件)

```

2014 年 8 月的活跃用户总数 :
12767
24/12/17 17:38:00 INFO SparkUI:

```

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。
- 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。

- 使用 user_balance_table 和 user_profile_table , 通过 user_id 关联用户的城市。
- 筛选 2014年8月 数据。
- 计算每个用户的总流量, 并按城市分组。
- 使用窗口函数对用户按总流量排序, 筛选排名前3的用户

代码:

```

from pyspark.sql import SparkSession

# 初始化 SparkSession
spark = SparkSession.builder.appName("User Analysis").getOrCreate()

# 加载 user_balance_table
user_balance_table = spark.read.option("header", "true").option("inferSchema",
"true").csv("file:///home/czz/projects/python_project/user_balance_table.csv")
user_balance_table.createOrReplaceTempView("user_balance_table")

# 加载 user_profile_table
user_profile_table = spark.read.option("header", "true").option("inferSchema",
"true").csv("file:///home/czz/projects/python_project/user_profile_table.csv")
user_profile_table.createOrReplaceTempView("user_profile_table")

query1 = """
SELECT
    p.City AS city_id,
    ROUND(AVG(b.tBalance), 2) 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
"""

result1 = spark.sql(query1)
result1.show()

query2 = """
WITH city_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_table b
    JOIN
        user_profile_table p
    ON
        b.user_id = p.user_id
    WHERE
        b.report_date LIKE '201408%' -- 2014 年 8 月的数据
    GROUP BY
        p.City, b.user_id
),
city_top3_flow AS (
    SELECT
        city_id,
        user_id,

```

```

        total_flow,
        ROW_NUMBER() OVER (PARTITION BY city_id ORDER BY total_flow DESC) AS rank
    FROM
        city_user_flow
)
SELECT
    city_id,
    user_id,
    total_flow
FROM
    city_top3_flow
WHERE
    rank <= 3
"""
result2 = spark.sql(query2)
result2.show()

result1.write.csv("file:///home/czz/projects/python_project/result2.1",
header=True)
result2.write.csv("file:///home/czz/projects/python_project/result2.2",
header=True)

```

结果:

每个城市在2014年3月1日的用户平均余额:

```

24/12/17 18:17:24 INFO DAGScheduler: Job 6 finished: showString at NativeMet
24/12/17 18:17:24 INFO CodeGenerator: Code generated in 5.884762 ms
24/12/17 18:17:24 INFO CodeGenerator: Code generated in 17.63608 ms
+-----+-----+
|city_id|avg_balance|
+-----+-----+
|6281949| 2795923.84|
|6301949| 2650775.07|
|6081949| 2643912.76|
|6481949| 2087617.21|
|6411949| 1929838.56|
|6412149| 1896363.47|
|6581949| 1526555.56|
+-----+-----+

```

每个城市总流量前3高的用户:

```

2017-12-21 2017-12-20 info code generator code generator
+-----+-----+-----+
|city_id|user_id|total_flow|
+-----+-----+-----+
|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|
+-----+-----+-----+
only showing top 20 rows

```

任务3：Spark ML编程

3-1 首先统计2013 - 2014年8月的数据，计算出每天的总申购和赎回总额

代码：

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum as spark_sum

# 创建 SparkSession
spark = SparkSession.builder \
    .appName("Daily Purchase and Redeem Analysis") \
    .getOrCreate()

# 读取数据，假设文件在 CSV 格式
data_path = "file:///home/czz/info/Ali/user_balance_table.csv"
user_balance_df = spark.read.csv(data_path, header=True, inferSchema=True)

# 按天汇总申购和赎回总额
result_df = user_balance_df.groupBy("report_date").agg(
    spark_sum("total_purchase_amt").alias("total_purchase"),
    spark_sum("total_redeem_amt").alias("total_redeem")
)

# 排序输出

```

```

result_df = result_df.orderBy("report_date")

# 展示结果
result_df.show()

# 如果需要保存结果到文件
result_df.write.csv("file:///home/czz/projects/python_project/result3.1",
header=True)

# 停止 SparkSession
spark.stop()

```

结果：

```

1 report_date,total_purchase,total_redeem
2 20130701,32489348,5525022
3 20130702,29037390,2554548
4 20130703,27270770,5953867
5 20130704,18321185,6410729
6 20130705,11648749,2763587
7 20130706,36751272,1616635
8 20130707,8962232,3982735
9 20130708,57258266,8347729
10 20130709,26798941,3473059
11 20130710,30696506,2597169
12 20130711,44075197,3508800
13 20130712,34183904,8492573
14 20130713,15164717,3482829
15 20130714,22615303,2784107
16 20130715,48128555,13107943
17 20130716,50622847,11864981
18 20130717,29015682,10911513
19 20130718,24234505,11765356
20 20130719,33680124,9244769
21 20130720,20439079,4601143
22 20130721,21142394,2681331
23 20130722,40448896,19144267
24 20130723,58136147,24404051
25 20130724,48422518,36258592
26 20130725,57433418,38212836

```

3-2 然后使用线性回归拟合数据，预测九月的结果

- 先对日期进行处理，使其能够作为线性回归的变量
- 读取训练集，进行训练
- 预测结果

代码：

```

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.sql.functions import unix_timestamp, col,to_date,date_format

# 创建SparkSession
spark = SparkSession.builder.appName("PurchaseRedeemPrediction").getOrCreate()

# 加载数据
data =
spark.read.csv("file:///home/czz/projects/python_project/result3.1/total_balance.
csv", header=True, inferSchema=True)

# 将整数类型的 report_date 转换为字符串类型，并格式化为日期格式

```



```

data = data.withColumn("report_date_str",
to_date(col("report_date").cast("string"), "yyyyMMdd"))

# 转换日期格式为数值：日期转换为自某一基准日期以来的天数
data = data.withColumn("date_num", unix_timestamp("report_date_str", "yyyy-MM-dd").cast("long"))

# 显示数据结构
data.show(5)

# 构建特征向量
assembler = VectorAssembler(inputCols=["date_num"], outputCol="features")
assembled_data = assembler.transform(data)

# 显示处理后的数据
assembled_data.select("report_date", "features", "total_purchase",
"total_redeem").show(5)

# 训练回归模型：预测申购总额
lr_purchase = LinearRegression(featuresCol="features", labelCol="total_purchase")
lr_model_purchase = lr_purchase.fit(assembled_data)

# 训练回归模型：预测赎回总额
lr_redeem = LinearRegression(featuresCol="features", labelCol="total_redeem")
lr_model_redeem = lr_redeem.fit(assembled_data)

import pandas as pd

# 生成2014年9月1日至9月30日的日期
dates_september = pd.date_range('2014-09-01', '2014-09-30').strftime('%Y-%m-%d').tolist()

# 创建DataFrame并转换日期为数字格式
df_september = spark.createDataFrame([(date,) for date in dates_september],
["date"])
df_september = df_september.withColumn("date_num", unix_timestamp("date", "yyyy-MM-dd").cast("long"))

# 生成特征向量
df_september_assembled = assembler.transform(df_september)

# 使用模型进行预测
forecast_purchase = lr_model_purchase.transform(df_september_assembled)
forecast_redeem = lr_model_redeem.transform(df_september_assembled)

# 显示预测结果
# forecast_purchase.select("date", "prediction").show()
# forecast_redeem.select("date", "prediction").show()

# 合并预测结果，生成期望的输出格式
forecast_purchase_redeem = forecast_purchase.join(forecast_redeem, on="date",
how="inner") \
.select(date_format("date", "yyyyMMdd").alias("date"),

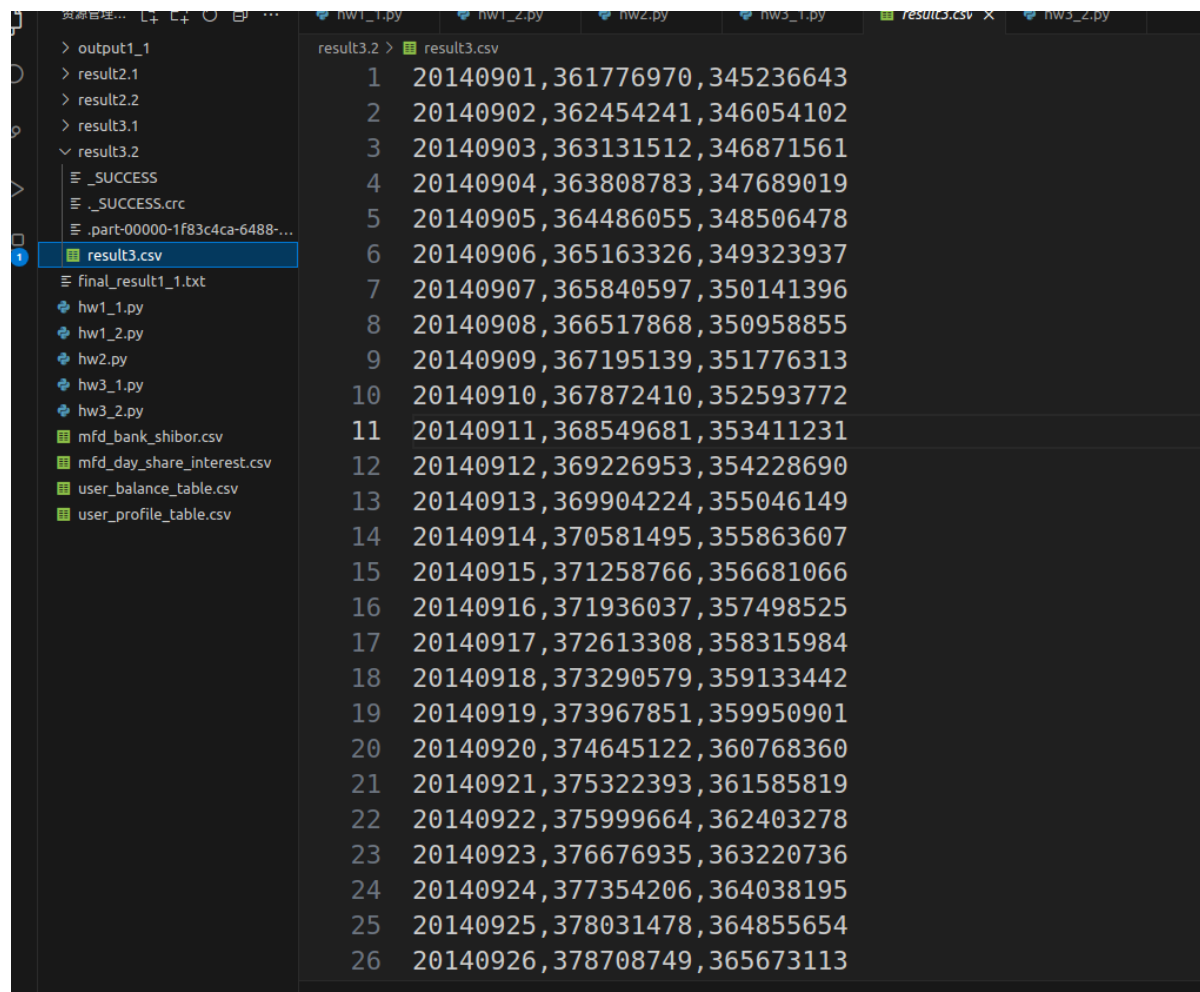
```

```
forecast_purchase["prediction"].cast("int").alias("total_purchase"),  
forecast_redeem["prediction"].cast("int").alias("total_redeem"))
```

保存为CSV文件，不带header

```
forecast_purchase_redeem.select("date", "total_purchase", "total_redeem") \  
  .write.option("header",  
  "false").csv("file:///home/czz/projects/python_project/result3.2")
```

结果:



1	20140901,361776970,345236643
2	20140902,362454241,346054102
3	20140903,363131512,346871561
4	20140904,363808783,347689019
5	20140905,364486055,348506478
6	20140906,365163326,349323937
7	20140907,365840597,350141396
8	20140908,366517868,350958855
9	20140909,367195139,351776313
10	20140910,367872410,352593772
11	20140911,368549681,353411231
12	20140912,369226953,354228690
13	20140913,369904224,355046149
14	20140914,370581495,355863607
15	20140915,371258766,356681066
16	20140916,371936037,357498525
17	20140917,372613308,358315984
18	20140918,373290579,359133442
19	20140919,373967851,359950901
20	20140920,374645122,360768360
21	20140921,375322393,361585819
22	20140922,375999664,362403278
23	20140923,376676935,363220736
24	20140924,377354206,364038195
25	20140925,378031478,364855654
26	20140926,378708749,365673113

所得分数:



可能的改进：

1. 使用多元线性回归，使用给定的其他特征元素进行多元线性回归，可能预测更加准确
2. 改用其他的模型，如神经网络模型，将多个特征和总额组成训练集，去训练一个神经网络。然后去获取2014年9月的相关特征，进而去预测总额