目录

一、实验内容.....1
二、环境配置.....1
1. 安装 java.....1
2. 安装 hadoop.....1
3. 安装 scala.....2
4. 安装 spark.....3
5. 与 Anaconda 搭配使用.....3
三、实验实现.....4
1. 数据集介绍.....4
2. 初始化 spark.....4
3. 数据预处理.....5
4. 创建 Graph.....7
5. 寻找 Top10 大 V.....7
6. 寻找关系密切人群.....8
附件.....10

一、实验内容

1. 寻找社交网络数据集，数量不少于一万条。
2. 使用 pyspark 编写程序实现如下任务：

（1）找出 top10 的大 V；

（2）应用社区发现算法，找出关系密切的人群；

（3）讲解并分析所使用的社区发现算法程序。

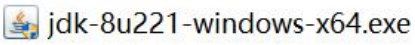
二、环境配置

Windows10，java 1.8，hadoop 3.2.0，spark 3.1.2，scala 2.12.14.

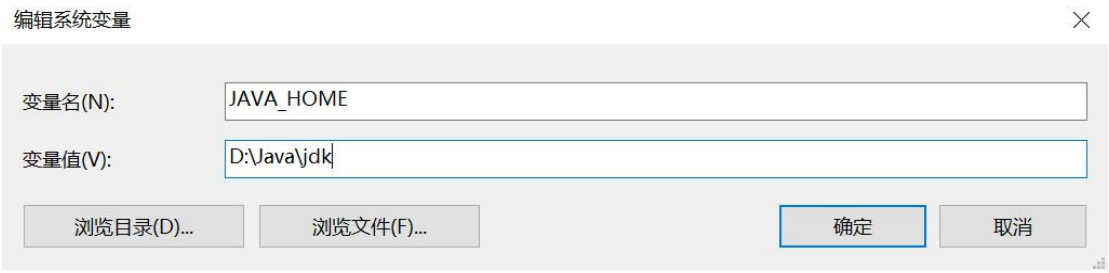
1. 安装 java

从官网下载安装包，安装并配置环境变量：

（1）安装包：

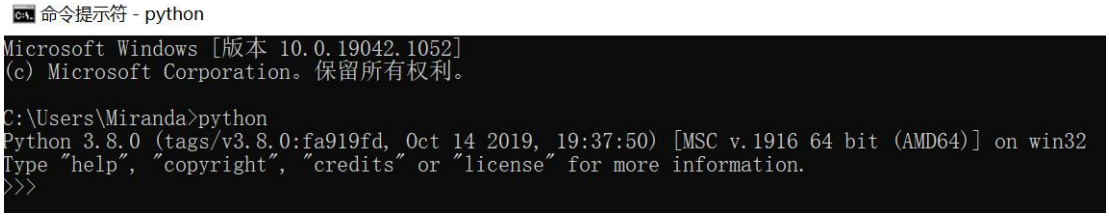


（2）设置系统变量：



（3）在系统变量 Path 中添加路径：

（4）在 cmd 命令框中输入 python，测试是否安装成功：



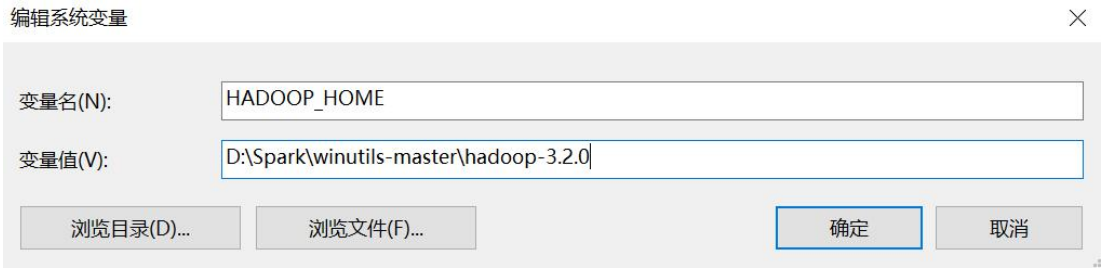
2. 安装 hadoop

从官网下载安装包，直接解压并配置环境变量：

（1）安装包：

由于是在 windows 上运行，还需要额外下载对应版本的 winutils.exe，放入 hadoop 的 bin 文件夹中辅助 hadoop 工作。

（2）配置系统变量：



（3）在系统变量 Path 中添加路径：

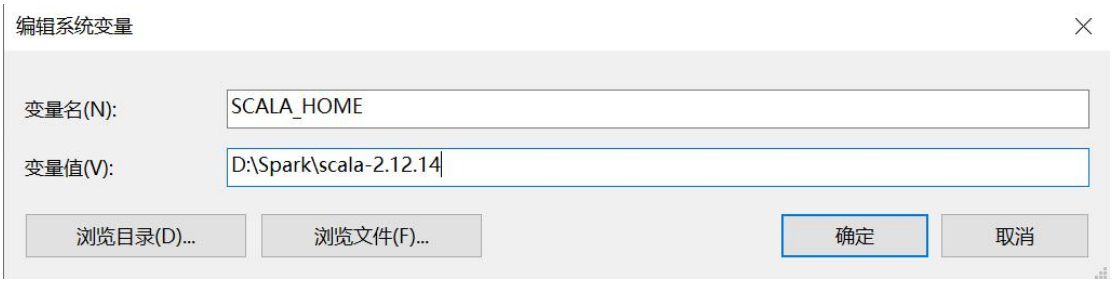
D:\Spark\winutils-master\hadoop-3.2.0\bin

3. 安装 scala

从官网下载安装包，直接解压并配置环境变量：

（1）安装包：

（2）配置系统变量：



（3）在系统变量 Path 中添加路径：

D:\Spark\scala-2.12.14\bin

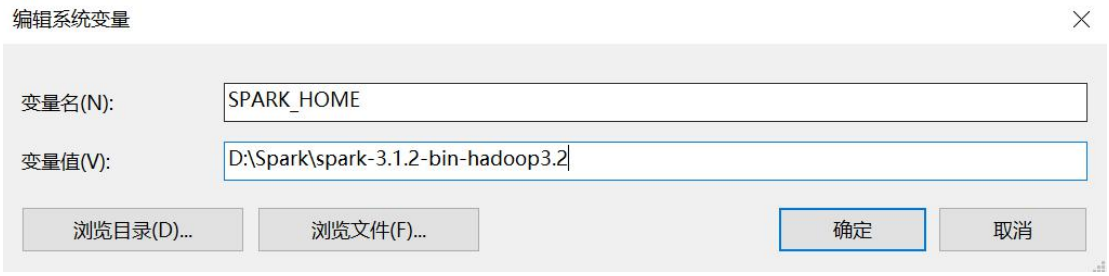
（4）在 cmd 命令框中输入 scala，测试是否安装成功：

4. 安装 spark

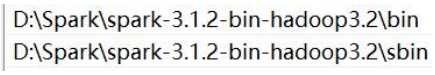
从官网下载安装包，直接解压并配置环境变量：

（1）安装包：

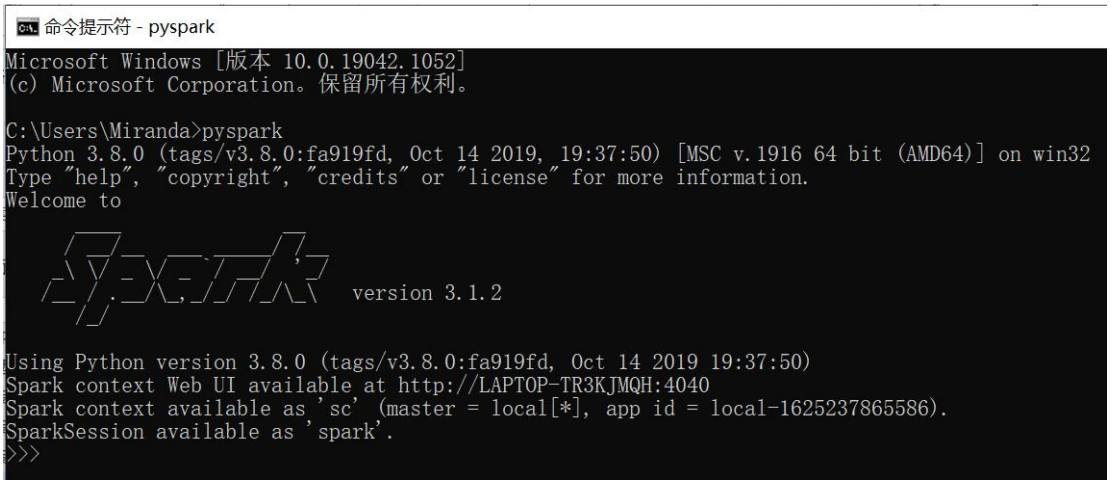
（2）配置系统变量：



（3）在系统变量 Path 中添加路径：



（4）在 cmd 命令框中输入 pyspark，测试是否安装成功：



5. 与 Anaconda 搭配使用

将 spark 安装包中 python 文件夹下 pyspark 文件夹复制到 anaconda 某个环境的 Lib\site-websites\文件夹下，就可以利用 jupyter notebook，利用搭建的 spark 环境，来编写代码。

三、实验实现

1. 数据集介绍

实验使用的社交网络数据集是公开社交网络数据集 twitter-combined，部分样本如图所示：

```
1  214328887 34428380
2  17116707 28465635
3  380580781 18996905
4  221036078 153460275
5  107830991 17868918
6  151338729 222261763
7  19705747 34428380
```

该数据集为脱敏数据集，用户姓名用编号代替，每一个编号为一个用户，每一行的数据之间用空格分隔，每一行代表一个关注三元组。每一行的第一个用户为关注行为的发起人，第二个用户为关注行为的接收人，即第一个用户关注了第二个用户。例如图中第 6 行，即代表用户 151338729 关注了用户 222261763。

该数据集共有 2420766 条关注数据，在实现算法时需要耗费大量时间，于是截取前 11604 条数据组成实验实际使用的数据集 twitter-sub。

2. 初始化 spark

在开始编写 python 代码前，需要再安装两个工具包：

```
C:\Users\Miranda>conda install py4j
```

```
C:\Users\Miranda>conda install findspark
```

Py4j 是用于 Java 与 Python 进行交互使用的包；使用 python 编写 spark 程序，需要用到 pyspark 第三方包去转为 jvm 中调用核心，而 findspark 可以提供简便的初始化 spark 环境，后续直接使用 pyspark 即可，这需要在代码的最前面加上两行代码：

```
import findspark
findspark.init()
```

然后 import 代码需要的包：

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark import SparkContext, SparkConf
from graphframes import GraphFrame
import time
```

接着程序初始化用 SparkSession 初始化 Spark，SparkSession 是 spark2.0 引入的概念，主要用在 sparkSQL 中，也可以用在其他场合，可以代替 SparkContext，SparkSession 其实是封装了 SQLContext 和 HiveContext。

```
spark = SparkSession.builder.appName("MySpark").getOrCreate()
sc = spark.sparkContext
```

3. 数据预处理

在数据预处理部分，首先加载数据，然后遍历每一条数据，统计共有多少位用户，再为每一位用户编号，接着将每一条关系（每一个样本）转换成对应的 id 表示，并补全关系，分别生成两个 DataFrame 格式的数据，用于后面构建 graph：

```
#提取所有实体并去重，然后转换成dataFrame
lines = sc.textFile('twitter_sub.txt')
entity = lines.flatMap(lambda x:x.split(' ')).distinct()
entity_zip = entity.zipWithIndex()
entity_row = entity_zip .map(lambda e: Row(id = e[1], name = e[0]))
entity_table = spark.createDataFrame(entity_row)
entity_table.show()
```

```
+---+---------+
| id|     name|
+---+---------+
|  0| 17116707|
|  1| 28465635|
|  2|380580781|
|  3| 18996905|
|  4|153460275|
|  5|222261763|
|  6| 88323281|
|  7| 19933035|
|  8| 17434613|
|  9|364971269|
| 10|100581193|
| 11|279787626|
| 12| 69592091|
| 13|187773078|
| 14|262802533|
| 15|280935165|
| 16|285312927|
| 17|254839786|
| 18|204317520|
| 19| 21548772|
+---+---------+
only showing top 20 rows
```

```
#实体数量
entity.count()
```

```
638
```

```
#提取所有关系，然后转换成dataFrame（实体id，实体id，关系）
entity_dict = dict(entity_zip .map(lambda e: (e[0], e[1])).collect())
relation_row = lines.map(lambda x:x.split(' ')).map(lambda r: Row(src = entity_dict[r[0]], dst = entity_dict[r[1]],\
                                                                  relationship = 'concerned '))
relation_table = spark.createDataFrame(relation_row)
relation_table.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|339|340|  concerned |
|  0|  1|  concerned |
|  2|  3|  concerned |
|341|  4|  concerned |
|342|343|  concerned |
|344|  5|  concerned |
|345|340|  concerned |
|  5|  6|  concerned |
|  7|346|  concerned |
|347|  8|  concerned |
|346|348|  concerned |
|  9|348|  concerned |
| 10| 11|  concerned |
|349| 12|  concerned |
|344| 13|  concerned |
|350| 14|  concerned |
|351|  6|  concerned |
| 15|352|  concerned |
|  5|353|  concerned |
| 16|344|  concerned |
+---+---+------------+
only showing top 20 rows
```

```
#关系数量
relation_row.count()
```

```
11604
```

两个 dataframe 数据分别保存了每个用户编号对应的 id，以及用 id 表示的关系，前者的表头必须是 id，name，后者的表头必须是 src，dst，relationship，这样才能利用 graphframes 创建 Graph。

为了便于查看，可以将这两个 dataframe 数据转存为 csv 文件:

```
#将提取出来的实体表和关系表写入csv文件夹，以供查看
entity_table.toPandas().to_csv('entity2id.csv',encoding='utf_8_sig',index=False)
relation_table.toPandas().to_csv('relation.csv',encoding='utf_8_sig',index=False)
```

用户及其对应 id 保存为 entity2id.csv 文件，格式如图所示：

	id	name
2	0	17116707
3	1	28465635
4	2	380580781
5	3	18996905
6	4	153460275
7	5	222261763
8	6	88323281
9	7	19933035
10	8	17434613
11	9	364971269
12	10	100581193
13	11	279787626
14	12	69592091

每一条关系（每一个样本）对应的 id 表示，保存为 relation.csv 文件，格式如图所示：

1	src	dst	relationship
2	40781	40782	concerned
3	0	1	concerned
4	2	3	concerned
5	40783	4	concerned
6	40784	40785	concerned
7	40786	5	concerned
8	40787	40782	concerned
9	5	6	concerned

4. 创建 Graph

将已经创建好的 dataframe 数据 entity_table 和 relation_table 作为参数，创建 Graph：

```
#构建Graph
g = GraphFrame(entity_table, relation_table)
```

在使用 GraphFrame 之前，不仅需要安装 graphframes，还需要到官网下载相应的 jar 包放到 Spark 安装包中 jars 文件夹下：

5. 寻找 Top10 大 V

对图中用户的入度进行计算，选择 top10 进行展示。结果如图所示。

```
#入度取前10展示
g.inDegrees.sort("inDegree",ascending=False).show(10)
```

```
+---+--------+
| id|inDegree|
+---+--------+
| 35|     174|
| 47|     167|
| 41|     160|
|340|     159|
|  3|     117|
|364|     112|
|353|     111|
|343|     110|
|388|     106|
|359|     101|
+---+--------+
only showing top 10 rows
```

从图中可以发现，入度最高的用户的 id 从高到低的前 10 位分别为：35、47、41、340、3、364、353、343、388、359。如果单从被关注的数量来看，这 10 位就是 Top10 大 V。但是如果从每个用户的权重值进行探究，可以使用 PageRank 算法来实现，结果展示了 PageRank 值从大到小排序的前 10 位：

```
#使用PageRank算法选出重要用户
tic = time.time()
result = g.pageRank(resetProbability=0.15, maxIter=5)
result.vertices.sort('pagerank',ascending=False).show(10)
toc = time.time()
print("PageRank算法用时:" + str(toc-tic))
```

```
+---+--------+------------------+
| id|    name|          pagerank|
+---+--------+------------------+
| 35|40981798| 9.571762777234309|
| 47|43003845| 8.536144105795914|
|230| 7861312| 7.479196505734391|
| 41|22462180| 7.092325211932122|
|340|34428380|7.0908748071041465|
|353|27633075| 5.410156272908095|
|514|25970331| 5.396491631422866|
|364|31331740| 5.349122654587942|
| 72|133055665| 5.105792570054932|
|359| 8088112|  5.01979464154914|
+---+--------+------------------+
only showing top 10 rows

PageRank算法用时:7.063082933425903
```

从图中可以发现，PageRank 值最高的前 10 个用户的 id 分别为：35、47、230、41、340、353、514、364、72、359。因此，Top10 大 V 分别为：40981798、43003845、7861312、22462180 、34428380、27633075、25970331、31331740、133055665、8088112。

6. 寻找关系密切人群

对创建的 Graph 应用标签传播算法（Label Propagation Algorithm）LPA 进行社区发现：

```
#使用LPA算法来做社区发现（CommunityDetection）
tic = time.time()
results = g.labelPropagation(maxIter=5)
results.orderBy('label').show(10)
toc = time.time()
print("LPA算法用时:" + str(toc-tic))
```

```
+---+--------+-----+
| id|    name|label|
+---+--------+-----+
|348|153226312|   23|
|415| 55033682|   23|
|385| 57490887|   23|
| 19| 21548772|   23|
|112|102765423|   23|
|367|430268163|   23|
| 65| 86221475|   23|
|418|186212304|   23|
|442|273149543|   23|
| 54|220068522|   23|
+---+--------+-----+
only showing top 10 rows

LPA算法用时:2.697854518890381
```

将结果按照标签排序展示前 10 位用户，其标签都是 23。然后查看整个社区分类的情况：

```
#查看社区分类情况
results.groupBy('label').count().collect()
```

```
[Row(label=487, count=31),
 Row(label=227, count=2),
 Row(label=621, count=1),
 Row(label=246, count=2),
 Row(label=307, count=2),
 Row(label=328, count=2),
 Row(label=185, count=2),
 Row(label=550, count=193),
 Row(label=89, count=1),
 Row(label=254, count=1),
 Row(label=451, count=67),
 Row(label=559, count=1),
 Row(label=478, count=13),
 Row(label=83, count=15),
 Row(label=514, count=78),
 Row(label=286, count=1),
 Row(label=313, count=1),
 Row(label=312, count=1),
 Row(label=23, count=207),
 Row(label=633, count=1),
 Row(label=282, count=1),
 Row(label=469, count=15)]
```

对 label 标签进行统计，共有 22 个标签。可以看出，LPA 算法将 638 个用户不重叠地划分到了 22 个社区之中，每个社区对应的用户数量展示在第二列。

将整个结果保存到 results.csv 文件中便于查看：

```
#将结果写入csv文件夹，以供查看
results.toPandas().to_csv('results.csv',encoding='utf_8_sig',index=False)
```

1	id	name	label
2	26	206923844	23
3	29	451250774	23
4	474	20757640	451
5	65	86221475	23
6	191	37008705	514
7	418	186212304	23
8	541	25592034	514
9	558	58714972	550

附件

1. 完整代码及生成的相关文件：
   https://github.com/MIrandaStock/Pyspark-CommunityDetection
2. 相关的安装包和 jar 包：
   链接：https://pan.baidu.com/s/190zKj8Jr_UVIXU-bNVIc0Q
   提取码：ryuk