Recitation 4 Group 7 Report

Weijia Sun(ws368)
Tianyi Zhang(tz168)
Jiahao Tang(jt846)

Task 1

What does the code in PageRank.scala do?

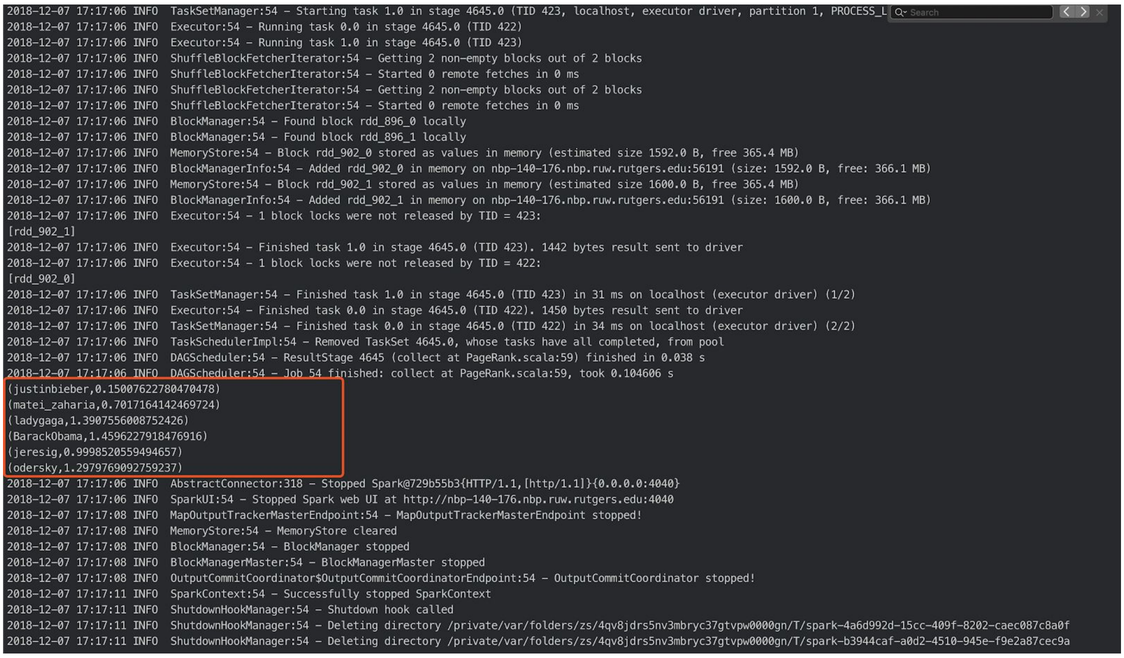
The code is used to rank the web page for the follower. The result of the code shows the relationship among the followers.

What is the convergence criteria for PageRank in the example?

```
val ranks = graph.pageRank(0.0001).vertices
```

This line is the convergence criteria of the PageRank in this example. When the value is smaller than 0.0001. The algorithm converges and stop.

ScreenShot



```
2018-12-07 17:17:06 INFO  TaskSetManager:54 - Starting task 1.0 in stage 4645.0 (TID 423, localhost, executor driver, partition 1, PROCESS_L
2018-12-07 17:17:06 INFO  Executor:54 - Running task 0.0 in stage 4645.0 (TID 422)
2018-12-07 17:17:06 INFO  Executor:54 - Running task 1.0 in stage 4645.0 (TID 423)
2018-12-07 17:17:06 INFO  ShuffleBlockFetcherIterator:54 - Getting 2 non-empty blocks out of 2 blocks
2018-12-07 17:17:06 INFO  ShuffleBlockFetcherIterator:54 - Started 0 remote fetches in 0 ms
2018-12-07 17:17:06 INFO  ShuffleBlockFetcherIterator:54 - Getting 2 non-empty blocks out of 2 blocks
2018-12-07 17:17:06 INFO  ShuffleBlockFetcherIterator:54 - Started 0 remote fetches in 0 ms
2018-12-07 17:17:06 INFO  BlockManager:54 - Found block rdd_896_0 locally
2018-12-07 17:17:06 INFO  BlockManager:54 - Found block rdd_896_1 locally
2018-12-07 17:17:06 INFO  MemoryStore:54 - Block rdd_902_0 stored as values in memory (estimated size 1592.0 B, free 365.4 MB)
2018-12-07 17:17:06 INFO  BlockManagerInfo:54 - Added rdd_902_0 in memory on nbp-140-176.nbp.ruw.rutgers.edu:56191 (size: 1592.0 B, free: 366.1 MB)
2018-12-07 17:17:06 INFO  MemoryStore:54 - Block rdd_902_1 stored as values in memory (estimated size 1600.0 B, free 365.4 MB)
2018-12-07 17:17:06 INFO  BlockManagerInfo:54 - Added rdd_902_1 in memory on nbp-140-176.nbp.ruw.rutgers.edu:56191 (size: 1600.0 B, free: 366.1 MB)
2018-12-07 17:17:06 INFO  Executor:54 - 1 block locks were not released by TID = 423:
[rdd_902_1]
2018-12-07 17:17:06 INFO  Executor:54 - Finished task 1.0 in stage 4645.0 (TID 423). 1442 bytes result sent to driver
2018-12-07 17:17:06 INFO  Executor:54 - 1 block locks were not released by TID = 422:
[rdd_902_0]
2018-12-07 17:17:06 INFO  TaskSetManager:54 - Finished task 1.0 in stage 4645.0 (TID 423) in 31 ms on localhost (executor driver) (1/2)
2018-12-07 17:17:06 INFO  Executor:54 - Finished task 0.0 in stage 4645.0 (TID 422). 1450 bytes result sent to driver
2018-12-07 17:17:06 INFO  TaskSetManager:54 - Finished task 0.0 in stage 4645.0 (TID 422) in 34 ms on localhost (executor driver) (2/2)
2018-12-07 17:17:06 INFO  TaskSchedulerImpl:54 - Removed TaskSet 4645.0, whose tasks have all completed, from pool
2018-12-07 17:17:06 INFO  DAGScheduler:54 - ResultStage 4645 (collect at PageRank.scala:59) finished in 0.038 s
2018-12-07 17:17:06 INFO  DAGScheduler:54 - Job 54 finished: collect at PageRank.scala:59, took 0.104606 s
(justinbieber,0.15007622780470478)
(matei_zaharia,0.7017164142469724)
(ladygaga,1.3907556008752426)
(BarackObama,1.4596227918476916)
(jeresig,0.9998520559494657)
(odersky,1.2979769092759237)
2018-12-07 17:17:06 INFO  AbstractConnector:318 - Stopped Spark@729b55b3{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-12-07 17:17:06 INFO  SparkUI:54 - Stopped Spark web UI at http://nbp-140-176.nbp.ruw.rutgers.edu:4040
2018-12-07 17:17:08 INFO  MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!
2018-12-07 17:17:08 INFO  MemoryStore:54 - MemoryStore cleared
2018-12-07 17:17:08 INFO  BlockManager:54 - BlockManager stopped
2018-12-07 17:17:08 INFO  BlockManagerMaster:54 - BlockManagerMaster stopped
2018-12-07 17:17:08 INFO  OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator stopped!
2018-12-07 17:17:11 INFO  SparkContext:54 - Successfully stopped SparkContext
2018-12-07 17:17:11 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-12-07 17:17:11 INFO  ShutdownHookManager:54 - Deleting directory /private/var/folders/zs/4qv8jdrs5nv3mbryc37gtvpw0000gn/T/spark-4a6d992d-15cc-409f-8202-caec087c8a0f
2018-12-07 17:17:11 INFO  ShutdownHookManager:54 - Deleting directory /private/var/folders/zs/4qv8jdrs5nv3mbryc37gtvpw0000gn/T/spark-b3944caf-a0d2-4510-945e-f9e2a87cec9a
```

Task 2

Part 1

In [20]:
```
sc.stop()
import pyspark
from pyspark.sql import SparkSession
sc = pyspark.SparkContext(appName="sparkSQL")
ss = SparkSession(sc)
```

In [21]:
```
data = "file:////Users/weijiasun/CloudComputing18/CloudComputingRec4/Task2_problems/kddcup.data
raw = sc.textFile(data).cache()
```

DataFrame ¶

A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs

We want to convert our raw data into a table. But first we have to parse it and assign desired rows and headers, something like csv format.

In [22]:
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
from pyspark.sql import Row
```

In [23]:
```
csv_data = raw.map(lambda l: l.split(","))
row_data = csv_data.map(lambda p: Row(
    duration=int(p[0]),
    protocol_type=p[1],
    service=p[2],
    flag=p[3],
    src_bytes=int(p[4]),
    dst_bytes=int(p[5])
    )
)
```

Once we have our RDD of Row we can infer and get a schema. We can operate on this schema with SQL queries.

In [24]:
```
kdd_df = sqlContext.createDataFrame(row_data)
kdd_df.registerTempTable("KDDdata")
```

In [25]:
```
# Select tcp network interactions with more than 2 second duration and no transfer from destina
tcp_interactions = sqlContext.sql("SELECT duration, dst_bytes FROM KDDdata WHERE protocol_type
tcp_interactions.show(10)
```

```
+--------+---------+
|duration|dst_bytes|
+--------+---------+
|    5057|        0|
|    5059|        0|
|    5051|        0|
|    5056|        0|
|    5051|        0|
|    5039|        0|
|    5062|        0|
|    5041|        0|
|    5056|        0|
|    5064|        0|
+--------+---------+
only showing top 10 rows
```

In [26]:
```
# Complete the query to filter data with duration > 2000, dst_bytes = 0.
# Then group the filtered elements by protocol_type and show the total count in each group.
# Refer - https://spark.apache.org/docs/latest/sql-programming-guide.html#dataframegroupby-reta

kdd_df.select("protocol_type", "duration", "dst_bytes").filter(kdd_df.duration>2000)#.more quer
```

Out[26]:
```
DataFrame[protocol_type: string, duration: bigint, dst_bytes: bigint]
```

In [27]:
```
def transform_label(label):
    '''
    Create a function to parse input label
    such that if input label is not normal
    then it is an attack
    '''



row_labeled_data = csv_data.map(lambda p: Row(
    duration=int(p[0]),
    protocol_type=p[1],
    service=p[2],
    flag=p[3],
    src_bytes=int(p[4]),
    dst_bytes=int(p[5]),
    label=transform_label(p[41])
    )
)
kdd_labeled = sqlContext.createDataFrame(row_labeled_data, samplingRatio=0.5)

'''
Write a query to select label,
group it and then count total elements
in that group
'''
# query
```

Out[27]:
```
'\nWrite a query to select label, \ngroup it and then count total elements\nin that group\n'
```

We can use other dataframes for filtering our data efficiently.

In [28]:
```
kdd_labeled.select("label", "protocol_type", "dst_bytes").groupBy("label", "protocol_type", kdd
```

```
+-----+-------------+---------------+------+
|label|protocol_type|(dst_bytes = 0)| count|
+-----+-------------+---------------+------+
| null|          tcp|          false| 70169|
| null|          udp|          false| 15594|
| null|          tcp|           true|119896|
| null|          udp|           true|  4760|
| null|         icmp|           true|283602|
+-----+-------------+---------------+------+
```

It can be inferred that we have large number of tcp attacks with zero data transfer = 110583 as compared to normal tcp = 9313.

This type of analysis is known as [exploratory data analysis]

In [ ]:

Part 2

In [1]:
```
import pyspark
from pyspark.sql import SparkSession
sc = pyspark.SparkContext(appName="sparkSQL")
ss = SparkSession(sc)
```

In [4]:
```
data = "file:////Users/weijiasun/Downloads/Rec4_Sol/Task2/recitation4/problems/kddcup.data_10_perc
raw = sc.textFile(data).cache()
```

We will create a local dense vector for our KDD dataset.

In [5]:
```
raw.take(1)
```

Out[5]:
```
[u'0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,
0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.']
```

In [6]:
```
import numpy as np

def parse_kdd(line):
    split = line.split(",")
    # we will keep just numeric and logical values
    # discard any string values
    symbolic_indexes = [1,2,3,41]
    clean_split = [item for i,item in enumerate(split) if i not in symbolic_indexes]
    return np.array([float(x) for x in clean_split])

vector_data = raw.map(parse_kdd)
```

In [7]:
```
from pyspark.mllib.stat import Statistics
from math import sqrt

# Compute column summary statistics.
summary = Statistics.colStats(vector_data)

print ("Duration Statistics:")
print (" Mean: {}".format(round(summary.mean()[0],3)))
print (" St. deviation: {}".format(round(sqrt(summary.variance()[0]),3)))
print (" Max value: {}".format(round(summary.max()[0],3)))
print (" Min value: {}".format(round(summary.min()[0],3)))
print (" Total value count: {}".format(summary.count()))
print (" Number of non-zero values: {}".format(summary.numNonzeros()[0]))
```

```
Duration Statistics:
 Mean: 47.979
 St. deviation: 707.746
 Max value: 58329.0
 Min value: 0.0
 Total value count: 494021
 Number of non-zero values: 12350.0
```

We are interested in preparing a classification system for attack/no attack or different attack types. This requires us to use label along with summary statistics and analyse data properly.

In [8]:
```
# Create a function to return a tuple with label as its zeroth index
# and corresponding summary statistic as its first index.
def parse_kdd_label(line):
    split = line.split(",")
    # we will keep just numeric and logical values
    # discard any string values
    label = split[41]
    symbolic_indexes = [1,2,3,41]
    clean_split = [item for i,item in enumerate(split) if i not in symbolic_indexes]
    return (label, np.array([float(x) for x in clean_split]))
```

In [9]:
```
def summary_by_label(raw_data, label):
    label_vector_data = raw_data.map(parse_kdd_label).filter(lambda x: x[0]==label)
    return Statistics.colStats(label_vector_data.values())
```

In [10]:
```
label_list = ["back.","buffer_overflow.","ftp_write.","guess_passwd.",
              "imap.","ipsweep.","land.","loadmodule.","multihop.",
              "neptune.","nmap.","normal.","perl.","phf.","pod.","portsweep.",
              "rootkit.","satan.","smurf.","spy.","teardrop.","warezclient.",
              "warezmaster."]
```

In [11]:
```
label_summary_dict = {label:summary_by_label(raw,label) for label in label_list}
# Create a dictionary of key = label_list elements, value = corresponding summary statistics
```

n our case, the interesting part of the summary statistics comes from being able to get them through cyber attacks or "tags" types in our dataset. By doing so, we will be able to better characterize our dataset dependent variables based on the range of independent values of the values.

If we want to do something like this, we can filter the RDD containing the label as a key and a vector as a value. To do this, we only need to adjust our parse_interaction function to return a tuple containing two elements.

In [11]:
```python
label_summary_dict = {label:summary_by_label(raw,label) for label in label_list}
# Create a dictionary of key = label_list elements, value = corresponding summary statistics
```

In [13]:
```python
label = 'buffer_overflow.'
print ("Duration Statistics for label : '" + label + "'")
print (" Mean: {}".format(round(label_summary_dict[label].mean()[0],3)))
print (" St. deviation: {}".format(round(sqrt(label_summary_dict[label].variance()[0]),3)))
print (" Max value: {}".format(round(label_summary_dict[label].max()[0],3)))
print (" Min value: {}".format(round(label_summary_dict[label].min()[0],3)))
print (" Total value count: {}".format(label_summary_dict[label].count()))
print (" Number of non-zero values: {}".format(label_summary_dict[label].numNonzeros()[0]))
print()
```

```
Duration Statistics for label : 'buffer_overflow.'
 Mean: 91.7
 St. deviation: 97.515
 Max value: 321.0
 Min value: 0.0
 Total value count: 30
 Number of non-zero values: 22.0
```

In [ ]: