UNIVERISTÀ DI ROMA "LA SAPIENZA"

INTERNSHIP @ ALMAWAVE S.R.L.

students:

Alessandro Gallo, Wesam Rukun, Simone Tilia

supervised by:

Ing. Cristina Giannone

March 27, 2017

1 Initial part

First of all we have imported all the necessary libraries in order to complete our Social Network Analysis task. In particular, we have installed the missing ones (iGraph, Pycairo, cairocffi) using the "pip install" command in the open source platform "Anaconda".

```
import os
import findspark
findspark.init()
import pyspark
```

Since we have used the "IPython Notebook" (also known as Jupyter notebook), we had to import the spark context and add manually GraphFrames package

```
sc = pyspark.SparkContext()
sc.addPyFile(os.path.expanduser(
    './graphframes-0.3.0-spark2.0-s_2.11.jar'))
from graphframes import *
```

Considering that we are using Spark 2.0, we had to create a new Spark Session

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Tweets") \
    .getOrCreate()

spark.conf.set("spark.driver.cores", 2)
spark.conf.set("spark.executor.memory", "4g")

from scipy.stats import itemfreq
from __builtin__ import *
import sys
import math
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import igraph as ig
import cairo
import time
import random

from pyspark.sql.functions import split, explode, size, col, asc, desc
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *

matplotlib.style.use('ggplot')
```

We have used "inline" command from matplotlib in order to plot directly in the notebook

```
%matplotlib inline
```

2 Definition of the functions

2.1 adjustFile

It converts the starting JSON file in a format that can be readable from pySpark and allows to make smaller cut files used for tests

```
def adjustFile(input_name, output_name, n):
    out_file = open(output_name,"w")

    i = 0

    with open(input_name,"r") as f:
        for line in f:
            if i >= n:
                break

            if line.startswith(","):
                out_file.write(line.strip(","))
                i += 1
            elif line.startswith("{"):
                out_file.write(line)
                i += 1


    out_file.close()
```

2.2 plotInDegreesDistrubution

It takes in input a GraphFrames graph and plot the in-degree distribution of the nodes

```
def plotInDegreesDistrubution(graph):
    # inDegree table on pandas

    indegrees_df = graph.inDegrees.toPandas()
    indegrees_df['id'].replace(
        regex=True,inplace=True,to_replace='\D',value=r'')

    # Plot Part

    fig = plt.figure(figsize=(20,10))
    plot = fig.add_subplot(111)

    degrees = {}

    for d in np.array(indegrees_df["inDegree"]):
        if d in degrees:
            degrees[d] += 1
        else:
            degrees[d] = 1


    plot.loglog(degrees.keys(),degrees.values(),'b-',
                basex=2,basey=2,marker='o',linestyle='None'
               )

    plt.xlim(2**-0.5, 2**(math.log(max(degrees.keys()),2)+1))
    plt.ylim(2**-0.5, 2**(math.log(max(degrees.values()),2)+1))

    plt.ylabel("# of nodes")
    plt.xlabel("degree")
    plt.title("In-Degree distribution")
    plt.show()
```

2.3 plotHashtagsGraph

It plots the input directed graph. The nodes colors, red and blue, mark respectively the users and the hashtags. The image will be saved on the same folder of this notebook and its size is auto-generated based on the number of the nodes

```
def plotHashtagsGraph(graph, users, hashtags, name):
  graph_to_plot = ig.Graph(directed=True)

  users_df = users.toPandas()
  for index,vertex in users_df.iterrows():
    if vertex["id"] is not None:
      graph_to_plot.add_vertex(vertex["id"],
                  label = vertex["id"].encode('utf-8'),
                  color="red")

  hashtags_df = hashtags.toPandas()
  for index,vertex in hashtags_df.iterrows():
    if vertex["id"] is not None:
      graph_to_plot.add_vertex(vertex["id"],
                  label = vertex["id"].encode('utf-8'),
                  color="blue")

  edges_df = graph.edges.toPandas()

  for index,edge in edges_df.iterrows():
    #print edge
    graph_to_plot.add_edge(edge["src"],edge["dst"])

  layout = graph_to_plot.layout("kk")

  number_of_vertex = len(users_df) +len(hashtags_df)
  vertex_size = number_of_vertex / 10
  image_size = (number_of_vertex*10, number_of_vertex*10)

  ig.plot(graph_to_plot, vertex_size = vertex_size,
      layout = layout, bbox = image_size, target="./" + name)
```

2.4 plotGraph

It simply plots the input graph. The image will be saved on the same folder of this notebook and its size is auto-generated based on the number of the nodes

```
def plotGraph(graph, name):
  graph_to_plot = ig.Graph(directed=False)

  nodes_df = graph.vertices.toPandas()

  for index,vertex in nodes_df.iterrows():
    if vertex["id"] is not None:
      graph_to_plot.add_vertex(vertex["id"],
                   label = vertex["id"].encode('utf-8'))

  edges_df = graph.edges.toPandas()

  for index,edge in edges_df.iterrows():
    graph_to_plot.add_edge(edge["src"],edge["dst"])

  number_of_vertex = len(nodes_df)
  vertex_size = number_of_vertex / 10
  image_size = (number_of_vertex*10, number_of_vertex*10)

  layout = graph_to_plot.layout("kk")
  ig.plot(graph_to_plot, vertex_size = vertex_size,
      layout = layout, bbox = image_size, target="./" + name)
```

2.5 plotGraphCommunities

It plots the input graph where the nodes colors mark the assigned community. The image will be saved on the same folder of this notebook and its size is auto-generated based on the number of the nodes.
Note that each community has a unique color.

```
def plotGraphCommunities(graph, nodes, edges, communities, name):
  graph_to_plot = ig.Graph(directed=False)

  nodes = nodes.select(col("id").alias("node_id"))

  nodes_and_communities = nodes.join(
    communities, nodes.node_id == communities.id)

  nodes_and_communities = nodes_and_communities.select(
    nodes_and_communities.id, nodes_and_communities.label)

  nodes_and_communities_df = nodes_and_communities.toPandas()

  colors = ig.known_colors.items()
  communities_colors = {}

  for index,vertex in nodes_and_communities_df.iterrows():
    if vertex["id"] is not None:
      if vertex["label"] not in communities_colors:
        communities_colors[vertex["label"]] =
          colors.pop(random.randint(0, len(colors)-1))[0]

      graph_to_plot.add_vertex(vertex["id"],
                  label = vertex["id"].encode('utf-8'),
                  color = communities_colors[vertex["label"]])

  edges_df = edges.toPandas()

  for index,edge in edges_df.iterrows():
    graph_to_plot.add_edge(edge["src"],edge["dst"])

  number_of_vertex = len(nodes_and_communities_df)
  vertex_size = number_of_vertex / 10
  image_size = (number_of_vertex*10, number_of_vertex*10)
  #vertex_size = 20
  #image_size = (200,200)

  layout = graph_to_plot.layout("kk")
  ig.plot(graph_to_plot, vertex_size = vertex_size,
      layout = layout, bbox = image_size, target="./" + name)
```

3 Social Network Analysis

3.1 Import a sample of the tweets

We have generated a new file called "xaa-new-cut" with the first 20000 tweets from the first dataset and we have put them into a spark data frame and a RDD sql table

```
adjustFile("xaa-new","xaa-new-cut", 20000)
df = spark.read.json("./xaa-new-cut")
df.createOrReplaceTempView("tweets")
```

3.2 Hashtags Graph

We have taken all the screen names and the hashtags in the previously imported dataframe so that only rows with at least a hashtag remained and then we have exploded the hashtag list. These rows represent the edges. This graph has an important role in the analysis because it will be used to generate the graph on which we will do the analysis.

```
edges = spark.sql(
   "SELECT DISTINCT \
     _source.user.screen_name as screen_name, \
     _source.entities.hashtags.text as hashtags \
   FROM tweets \
   WHERE _source.entities.hashtags IS NOT NULL")

 edges = edges.where(size(col("hashtags")) > 0)
 edges = edges.select(edges.screen_name.alias("src"),
           explode("hashtags").alias("dst"))
```

We have created the users, the hashtags and the nodes dataframes and finally we built the graph

```
# Users and hashtag generation
users = edges.select(edges["src"].alias("id")).distinct()
hashtags = edges.select(edges["dst"].alias("id")).distinct()

# Build the graph
nodes = hashtags.unionAll(users)
hashtags_graph = GraphFrame(nodes, edges)

# Show
edges.select(edges.src.alias("user"), edges.dst.alias("hashtag")).show()

plotInDegreesDistrubution(hashtags_graph)

# Statistics
print "number of tweet", df.count()
print "number of users", users.count()
print "number of hastags", hashtags.count()
print "number of edges", edges.count()
```

3.3 Selection of the useful hashtags

Based on in-degree distribution of the previoulsy generated graph, we remove the hashtags that are too popular or too rare in order to keep the "real" communities building an "induced" set of nodes and edges.

```
count = users.count()

# Remove all the hashtags tweeted by less than 0.2% of the total users
min_degree = max(count/500, 16)
# Remove all the hashtags tweeted by more than 10% of the total users
max_degree = count/10

print "Min Hashtag Degree (0.2% of the users)", min_degree
print "Max Hashtag Degree (10% of the users)", max_degree

# Filter application
new_nodes = hashtags_graph.inDegrees.filter(
  "inDegree > " + str(min_degree) + \
  " AND inDegree < " + str(max_degree)).select("id")

new_nodes.createOrReplaceTempView("new_nodes")
edges.createOrReplaceTempView("edges")

# Retain only the edges that goes to the new hashtags
new_edges = spark.sql(
  "SELECT edges.src, edges.dst \
  FROM edges INNER JOIN new_nodes \
  ON new_nodes.id = edges.dst").distinct()

print "number of new hashtags", new_nodes.count()
print "number of new edges", new_edges.count()
```

We have created the new users, hashtags and nodes dataframes and finally we built the final hashtag graph

```
# Users and hashtag generation
users = new_edges.select(new_edges["src"].alias("id")).distinct()
hashtags = new_edges.select(new_edges["dst"].alias("id")).distinct()

# Build the graph
nodes = hashtags.unionAll(users)
hashtags_n = GraphFrame(nodes, new_edges)

plotInDegreesDistrubution(hashtags_n)

plotHashtagsGraph(hashtags_n, users, hashtags, "hashtags-graph.png")
```

3.4 Users Graph

We have generated all the new edges using the hashtags graph. In particular, we have taken all the possible combinations of the users that shared the same hashtag. We have also removed all the loop edges (from a user to himself)

```
new_edges = hashtags_n.find("(u1)-[e1]->(ht); (u2)-[e2]->(ht)")
new_edges = new_edges.select(new_edges.u1, new_edges.u2)
new_edges = new_edges.where(col("u1") != col("u2"))
```

Since the output columns are lists that contain only one element, we have defined a UserDefinedFunction that extracts the users from them.

```
firstElement = UserDefinedFunction(lambda x : x[0], StringType())
new_edges = new_edges.withColumn("src", firstElement("u1"))
new_edges = new_edges.withColumn("dst", firstElement("u2"))
```

We have created the new edges and nodes dataframes and finally we built the user graph

```
new_edges = new_edges.select(new_edges.src, new_edges.dst).distinct()
users = new_edges.select(new_edges["src"].alias("id"))
users = users.unionAll(
    new_edges.select(new_edges["dst"].alias("id"))).distinct()

users_graph = GraphFrame(users, new_edges)

print "number of users", users.count()
print "number of edges", users_graph.edges.count()

plotInDegreesDistrubution(users_graph)

plotGraph(users_graph, users, new_edges, "users-graph.png")
```

3.5 PageRank Algorithm

We have applied the PageRank algorithm to the user graph in order to understand which are the most important nodes in it

```
results = users_graph.pageRank(resetProbability=0.15, tol=0.01)

pagerank_results = results.vertices.orderBy(results.vertices.pagerank.desc())

pagerank_results.toPandas().to_csv("./pagerank.csv")

pagerank_results.show(20)
pagerank_results = pagerank_results.select("id")
pagerank_results = spark.createDataFrame(pagerank_results.head(20))
```

3.6 Label Propagation Algorithm

We have applied the Label Propagation algorithm to the user graph in order to understand how many community are there and in which of them the users are.

```
lp_results = users_graph.labelPropagation(maxIter=6)

users_communities = lp_results.distinct()
users_communities.show()
print "Number of communities: ", users_communities.select(
  col("label")).distinct().count()

plotGraphCommunities(users_graph,
           users_communities,
           "communities-graph.png")
```