Data Mining

Homework 4

Due: 22/12/2017, 23:59.

Instructions—Read carefully!

You must hand in the homeworks electronically and before the due date and time.
Handing in: You must hand in the homeworks by the due date and time by an email to Mara (`sorella@dis.uniroma1.it`) that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework #`
where `#` is the homework number. In the text you must also mention the your name and student id (matricola). After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact Mara.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups (but the writeup should be yours, as per the collaboration policy), and to ask Mara (first) or Aris in case of questions.**

Problem 1.

Let $G = (V,E)$ be an unweighted, undirected graph, with $|V| = n$ and $|E| = m$. Let $\Gamma(v)$ denote the set of neighbors of a node v, (i.e., $\Gamma(v) = \{w \in V : (v,w) \in E\}$), and d_v the degree of v, $d_v = |\Gamma(v)|$.

Recall that the clustering coefficient is a measure of the extent to ones friends are connected with each other. Note that graphs with high clustering coefficient will contain a lot of triancles: subgraphs of nodes u,v,w such that all thre edges (u,v), (v,w), and (w,u) exist.

Often, instead of the average clustering coefficient in a graph we count the total number of triangles to measure the extent to which friends of friends tend to be friends. In this problem, we will see how we can count the number of triangles in a graph.

We will start from the sequential case, and then move to a distributed setting.

Assume to have a data structure that can answer edge existence queries in constant time, and onsider the following algorithms:

Brute Force enumerates over all triples of distinct vertices, and keeps track of how many of these triples are triangles.

WedgeEnumerator enumerates over all existing two-hop paths, or *wedges*:

```
$T \leftarrow 0$
for $v \in V$ do
    for $u \in \Gamma(v)$ do
        for $w \in \Gamma(v)$ do
            if $(u,w) \in E$ then
                $T \leftarrow T + \frac{1}{X}$
return $T$
```

Algorithm 2: `WedgeEnumerator(G(V,E))`

1. What value should X be for Alg. 2 to be correct (i.e., report the correct number of triangles)?

2. Give a worst-case bound on the running time in terms of n for a general graph, and for constant-degree graphs for both algorithms.

We would like a better triangle-counting algorithm. Recall that social networks are characterized by a *skewed* degree distribution—this means that the average degree might be constant, but there is a *heavy tail* of vertices with very high degree. Consider a star graph of n nodes: there are only $n - 1$ edges, and 0 triangles, yet the running time of Alg. 2 is $\Theta(n^2)$ because of all the work performed at the center vertex.

We would prefer to only count each triangle once, and it might seem like this would only save us a factor of X, yet we get much bigger savings if we are clever about the choice of the vertex responsible for counting a given triangle. The key idea, motivated by the star example, is that only the *lowest-degree* vertex of a triangle should be responsible for counting it. This allows to obtain a running time of $O(m^{3/2})$ (**prove it for extra credit**).

3. Report the pseudocode of an algorithm `WedgeEnumerator++` that implements this optimization.

These algorithms assume that G fits into memory of a single machine. This is not the case for a real-world social network. Therefore we will consider a distributed version of `WedgeEnumerator++` that uses the MapReduce paradigm (Alg 3). It is composed of 2 rounds, the first collects nodes neighbors and the second finds wedges for which a further edge exists in the graph, to close the corresponding triangle. A total order function $\prec$ (based on degree) is used to implement the "smaller degree" optimization.

Note that in Round 2, in addition to taking the output of Round 1, the algorithm also takes as input the original edge list, and differentiates between the two types of input by using a special character '\$' that is assumed not to appear anywhere else in the input.

```
Data: $G = (V,E)$ total order $\prec$ over nodes
Map 1: input $\langle (u,v); \varnothing \rangle$
    if $u \prec v$ then
        emit $\langle u; v \rangle$
    else
        emit $\langle v; u \rangle$
Reduce 1: input $< v; \mathcal{S} \subseteq \Gamma(v) >$
    for $(u, w) : u, w \in \mathcal{S}$ do
        emit $\langle (u,w); v \rangle$
Map 2:
    if input of type $\langle (u,w); v \rangle$ then
        emit $\langle (u, w); v \rangle$
    if input of type $\langle (u,v); \varnothing \rangle$ then
        emit $\langle (u, v); \$ \rangle$
Reduce 2: $(u, w); \mathcal{S} \subseteq V \cup \{\$\}$
    if $\$ \in S$ then
        for $v \in \mathcal{S} \cap V$ do
            emit $\langle v; 1 \rangle$
```

Algorithm 3: MR-WedgeEnumerator++

4. In MapReduce, the memory used by a single mapper or a reducer should be sublinear in the total size of the input. Prove that (even if there is skew in the input) the input to any reduce instance in the first round has $O(\sqrt{m})$ edges. **Hint: can you partition the set of nodes in "high" and "low" degree nodes?)**

5. Download a dataset of your choice browsing the WADAM dataset repository (for example, one from SNAP). It should be an undirected social network graph, where user relationshipss are represented as a list of edges, with $|E| > 0.5M$. **Note: (please!) include only the link and not the dataset in your delivery.**

6. Provide an implementation of the `MR-WedgeEnumerator++` algorithm, using Spark, and run it on your dataset to report: (1) the total number of triangles and (2) the clustering coefficient of the top-10 nodes by degree. Think of an easy way to get and use degree information that does not require storing extra information in each worker process. For the second round, you can exploit for example the `union` operation over Spark `RDDs`.

Note: you can, again, exploit the **Databricks** platform for this assignment, yet since the resource grant is limited (6GB RAM per cluster) you will have to experiment choosing an appropriate dataset to avoid *Out-Of Memory* errors. Check out the DBFS manual for help with uploading the dataset on Databricks distributed file system.

Problem 2. In this assignment we will begin practicing with Tensorflow and Neural Networks. We'll start with installing and setting up our TensorFlow environment with Jupiter Notebooks support.

1. Set up Tensor Flow on your machine. You have many installation options. The simplest (and sugggested) option is to use Docker; see the corresponding *Installing with Docker* paragraph for your OS. Note: if you are on Windows, follow this guide for installing Docker.

2. Pull the tensorflow docker image `gcr.io/tensorflow/tensorflow`:

   ```
   docker pull gcr.io/tensorflow/tensorflow
   ```

3. Run it with Jupiter support:

   ```
   docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow
   ```

Download the `tf_first_steps_lin_reg.ipynb` and `intro_to_neural_nets.ipynb` notebooks, and put them in the Jupyter notebooks tree root using the *upload* button.

- Run the `tf_first_steps_lin_reg.ipynb` notebook that will demonstrate the use of a simple linear regression model to predict house values. Go through it and complete the missing tasks.
- Then in the `intro_to_neural_nets.ipynb` notebook we will see how we can perform the same task by training a neural network using Tensor Flow.