UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Percrso di studio
Scienze e Tecnologie Informatiche

Elaborato finale

Titolo?

Sottotitolo (alcune volte lungo - opzionale)?

Supervisore
Montresor Alberto

Laureando
Leonardi Stefano

Anno accademico 2017/2018

Ringraziamenti

...thanks to...

Contents

1 Summary

This thesis goes back over the work that I was done during the internship inside the university, with the supervision of the PhD student Sheikh Nasrullah for the professor Montresor Alberto.

1.1 Introduction

The whole internship is based on the paper named "Community-enhanced Network Representation Learning for Network Analysis" alias CNRL, this paper proposes an innovative method for detecting the communities inside a graph, so my purpose is to take the code at the base of this paper, try to replicate the results shown and after that increase if it is possible the potentiality of this algorithm.
At the start, I had to need to understand the main problem and understand the dynamics at the base of it, therefore I read out some other papers like:

- "DeepWalk: Online Learning of Social Representations"
- "node2vec: Scalable Feature Learning for Networks"
- "How exactly does word2vec work?"

Before the explaining of the work done, we need to understand why the community detection is so important.

1.2 What are graphs and the communities

At the base of all, there are the graphs, a particular data structure formed of the nodes that can represent an entity, eventually with some attributes, and the edges that link two nodes, the edges are all directed or not. With this special data structure, we can represent a huge amount of situations. The purpose of the community detection is to find the communities inside the graph, that are groups of nodes that are similar.
What does this mean?
Two nodes are similar if have an equivalent role inside a group of nodes, they can are a hub of connection or an isolated point or more simply they have the same value for some attribute. To find this groups can permit to who look for it, to manage those nodes in a similar way. Like the possible application of the graphs, the same is for this technology, there are lots of situation and context where those algorithms can work very well.

1.3 How we evaluate the community

Up to now, I have explained what are graphs and communities, and from my words seem that when you choose some groups of nodes is may be easily understood if it is a good partition or not. And if you have two partitions to decide which is the best is not difficult. This is not really true. If we have a little graph with only one community we can look at it and decide if the partition is good or not, but this is not possible for a big graph.
So how we evaluate a partition?
First of all, we must explain what we intend with partitions, here the characteristics:

- A partition is a set of communities
- Not all the communities have the same size, or better the same number of nodes
- For the communities, there is not a minimum size neither a maximum size
- Is possible that some nodes do not have a community to which it belongs

- A node can belong to more than one community, this mean that I have an overlapping of communities

Said that, we can speak about the modularity.
The modularity is the method used for evaluating a partition, there are different implementations of it because the aim is the same but we can have different interest so we can weight different aspect, in our work we have used three different methods: mod_withMax, mod_overlap and modified modularity. The first two are unofficial names, we have chosen them for distinguishing it, the last one is the method used in the CNRL's paper. More explanation of those methods in the chapter on the modularity.

1.4 Our changes

The code of CNRL do x random walks (a sequence of l adjacent nodes) on the graph where $x = n * w$:

- n: is the number of nodes in the graph
- w: is the number of walks starting from each node
- l: is the maximum length of each walk

Given those walks, the algorithm calculates the partition. Note that CNRL use only the structure of the graph, the nodes and the edges, no other information is considered.
Our approach uses the code of CNRL but introduces the attributes of the nodes, this permit a better evaluation. This because two nodes that before are far, there was no one link between them, now are considered near if they share one or more attributes.

1.5 Results

To be continued...

2 Implementation

In this section, I am going to explain thanks to which method we tried to outperform the potentiality of the code of CNRL.

2.1 How the CNRL's code work

How the code of CNRL exactly work?
First of all, the data are loaded from files, this includes the graph in edgelist format. After that, the graph is prepared for node2vec algorithm this is a particular method for exploring the graph, in fact, it depends on two parameters, named p and q. The value of the two parameters allow a depth visit or something that are more similar to a breadth visit, or possibly a solution in the middle.
But why we walk on the graph?
The main idea is to start some walks (in the number of w), from each node of the graph (in the number of n). And each walk have a length not higher than the value of l, I said not higher because if we have a directed graph it is possible that we arrive in a sink, then the walk must be interrupted because it is not possible to walk more. Differently, if we have an undirected graph all the walks have a length of l because node2vec allows a path that goes back.
At the end of that, we have $w \cdot n$ walks with a maximum length of l
Given the walks it is possible to obtain the community of the graph, this process is done with two different approaches, the first one is one of the most famous, named word2vec, that is going to consider the nodes' ID find in the walks like words, and then consider the walks like sentences.
Differently in the code of CNRL is used an algorithm created on purpose, and we have never tried to understand how it works exactly.
After that the partition is created, it is possible to calculate the modularity, with the algorithm that I explained in his chapter.

2.2 Introduce of attribute

As you can understand, CNRL's code for calculating the community use only the structure of the graph, that means, how the nodes and the edges are interconnected.
We note that both dataset on which we work, named Cora and Citeseer (better explained in the chapter on the experiments), have for each node a set of attribute, this can be present 1 or not 0. All those pieces of information are not considered from the CNRL's code so we decide to introduce it. The only part where we can work is before the extraction of the community given the walks and after the loading of data from the input files, this means that we can change only the generation of the walks.
The idea is to create in addition to the firsts walks, some other walks that manage the nodes through the attributes. When we see at the graph we can say that two nodes are near if there is a link between them.
When can we say that two nodes through an attribute are similar? Of course, when both share the same attribute. We can imagine that the sharing of an attribute is like an undirected link or edge, in fact, there is not any direction with the sharing. Then we can build a new undirected graph that considers the attribute and we are going to go to walk on it.

2.3 Attribute graph

In all the previous words we have described the graph taken from the attribute. I try now to formalize this concept.
First of all, we must remember that we have the original edges that we must delete, in fact, we do not want to walk on it, or the new walks will be a copy of the first ones. Then from the original graph, we

preserve only the nodes, called from now N1.
Thanks to the reasoning done before, that if two nodes share the same attribute, they are near. We can imagine that an edge, goes or directly from a node of N1 to another. Or we can create a new set of nodes, named N2, that contain all the attributes, each attribute becomes a node.

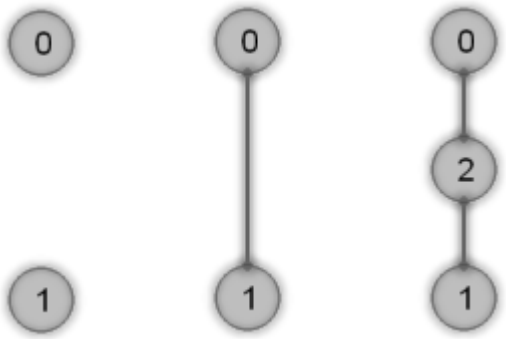

Figure 2.1: Two nodes that share an attribute, how we can link them, 1-no edge, 2-hypergraph, 3-bipartite graph

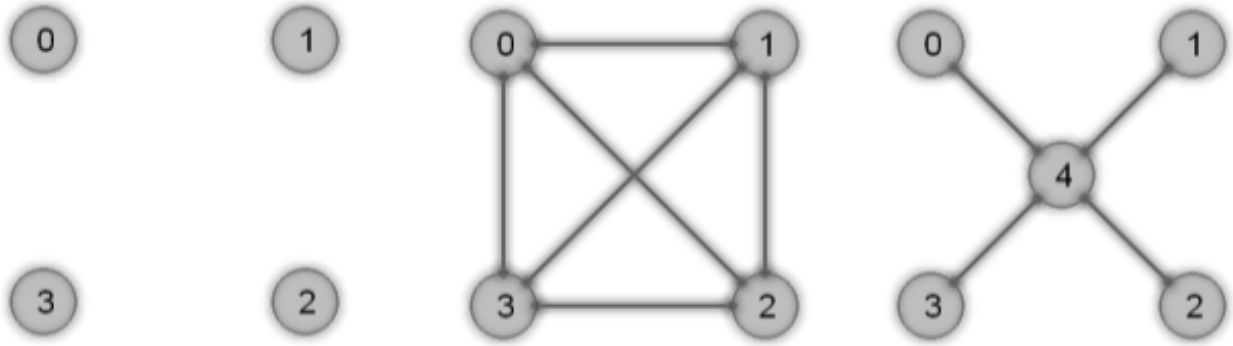

Figure 2.2: Four nodes that share an attribute, how we can link them, 1-no edge, 2-hypergraph, 3-bipartite graph

Then exist two methods to link some nodes through the attribute. In figure 2.1 and 2.2, there are two different examples. In both figures we have three scenes, the first one shows the nodes without any linkage, the second one shows the nodes connected directly, in the last one we have the nodes connected across another node that does the hub, this hub is the node taken from the attribute. Is easy to understand the difference, the second scene, named hypergraph, create a complete graph, this mean that I have $\binom{n}{2}$ edges, in figure 2.2 are $\binom{4}{2} = 6$, but no one node is added.
The third scene is a star graph that builds a bipartite graph, add a node for each attribute, but only n edges, for figure 2.2 this mean that we add one node and four edges.
Which one we prefer?
It depends on the situations, if there are lots of attributes and each of these is available for a little number of nodes, probably we are going to prefer the hypergraph. If there are some attribute that is available for a high number of nodes probably we prefer the bipartite graph.
In fact if for example we have an attribute available for 100 nodes this mean that with the bipartite graph we add 1 node and 100 edges, for the hypergraph we add zero nodes but $\binom{100}{2} = 4950$ edges.
For this reason, we have chosen for use the bipartite graph.
To note that when we walk on this new graph we save in the array of walks for both type of nodes, then after walking we must delete the ID of the nodes belonging to the attributes. Because the future steps of the algorithm must not know the existence of the bipartite graph.

2.3.1 Name of bipartite graph

I would like to explain where this name does come from. The graph is called bipartite because there are two set of nodes, the nodes that before I called N1 (original nodes) and N2 (nodes from attribute). Every edge of the graph goes from a node of N1 to a node of N2 and vice versa. There is no one edge that goes from N1 to N1 or from N2 to N2. This is perfect for our situation if we had two nodes of N1 linked this would mean that we have not deleted the original edges if we would have two nodes of N2 linked this would mean that we have done an error, two attributes can not be linked.

2.3.2 Name of hypergraph

I would like to explain where this name does come from. The hypergraph do not have simple edges but have hyperedges, the normal edges have two endpoints and both are nodes, so its possible to represent it with a line. Differently, the hyperedges do not have two endpoints but potentially an infinite number of endpoints, this allows that a hyperedge link more than two nodes directly. If you want to represent an hyperedges on a normal undirected graph this mean that, if the hyperedges link directly 5 nodes we must replace it with a complete graph on this 5 nodes so we use $\binom{5}{2} = 10$ normal edges.

Note that, this is exactly our situation, in fact, an attribute link the nodes to which it belongs directly, so we can imagine that for each attribute we are going to create a hyperedge on his nodes.

2.4 Build graph - example

In this section, I am going to show how a graph is transformed from the simple structure and the attributes to a hypergraph or to a bipartite graph.

ID	attributes ID	
1	11	
2	11	12
3	10	12
4	11	13
5	11	12
6	12	13
7	13	14

Table 2.1: For each node, there are the attributes ID that it has

att	node ID			
10	3			
11	1	2	4	5
12	2	3	5	6
13	4	6	7	
14	7			

Table 2.2: For each attribute, there are the nodes ID that it has

In figure 2.3 we can see the structure of the graph. In table 2.1 we can see that for each node is represented the ID of the attributes that it has. At the opposite in table 2.1 for each attribute, are represented the ID of the nodes that have it.
Then now it is possible to create the two graphs.
First, we look at the hypergraph shown in figure 2.4.

- we can see that some edges are no more present, two examples are $(1, 3)$ and $(5, 7)$, some other before was deleted and after they are replaced
- the nodes that share the same attribute build a complete graph, two example are $[1, 2, 4, 5]$ with the attribute 11 and $[4, 6, 7]$ with the attribute 13
- if only one node has a particular attribute this does not create an edge, this is the case of the two attributes 10 and 14

We look now a more complex situation, the bipartite graph shown in figure 2.5. It is not easy to see what happen.
The nodes from 1 to 7, on the left, are the original ones the nodes from 10 to 14 (higher equal than 10), on the right, are the new ones created from the attributes.

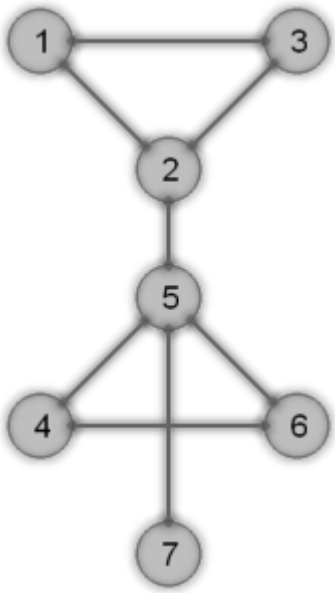

Figure 2.3: The structure representation of the graph

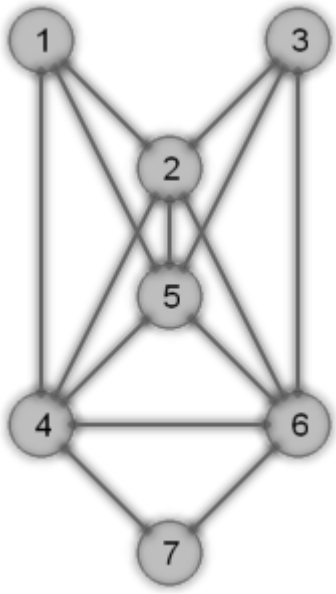

Figure 2.4: The representation of the hypergraph

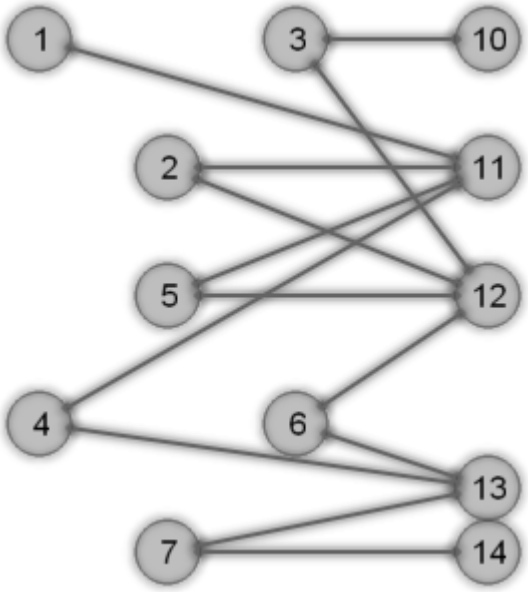

Figure 2.5: The representation of the bipartite graph

- the nodes on the left and the nodes on the right are separate are the two sets of which I spoke before N1 and N2
- all edges cross the middle of the graph, go from left to right
- all the edges present in the original graph are no more present, we have deleted all of them
- the nodes that share the same attribute build a star graph, two example are $[1, 2, 4, 5]$ with the attribute 11 and $[4, 6, 7]$ with the attribute 13
- if only one node has a particular attribute this do not do difference, in fact, the two attributes 10 and 14 are anyway linked to the left part

3 Evaluation methods

In this section, I am going to explain how we evaluate the partitions created by the community detection algorithms, as I introduce in the summary, this method, that we decided to use, is called modularity. Its symbol is Q.
The modularity is a value that is typically contained in the range of $[-1, 1]$, but this is not all times true because it depends on the implementations. The first persons who have worked on the modularity were Newman and Girvan, that like us, they had needed a method for choosing between two partitions of communities on the nodes.
They start from an assumption a node can not belong to more than one community, this assumption is not always true for our situation but it simplifies a lot the complex problem of calculating modularity. So for the first steps, we use the same assumption.
During the time of the internship we use three different methods, not ideate from our, now I am going to explain them from the simplest to the most complex. The formulas reported are for undirected graphs at the end of each subsection there is the differences with directed graphs.

3.1 Modularity with Maximum

We named it in this method because it permits at maximum one community for each node, perfectly in line with the assumption of Newman and Girvan.
The idea is that this method calculates a modularity value for each community in the graph and the total modularity is simply the summation of all the parts. We must specify that, in this method, the overlapping is not considered then the summation does not consider the same elements more than one times.
This method is based on the observation that we can identify a community on the base of the frequency of its edges, in fact, if a set of points have a huge amount of edges inside it probably this is a community if it is not, this is a bad situation that must penalize all the partition.
We chose it because is easy to calculate and because it penalizes a lot the wrong partitions.
Here the formula explained:

$$Q = \sum_{c}^{C_r} \left(\frac{l_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right) \tag{3.1}$$

Where the elements of the formula are:

- Q is the symbol for modularity
- C_r is the set of the communities
- c is the iterator on the communities
- L is the total number of edges in the graph
- l_c is the total number of edges inside, both nodes of each edge are inside the community c
- d_c is the degree of the community c, defined as the summation of the degree of the nodes belong to it

We can see that $\frac{l_c}{L}$ is the weight of the community compared to the others in the graph, and $\frac{d_c}{2L}$ is the density of edges, in other words, the percentage of edges that belong to this community compared to the total number.

To note that when the graph is directed change only a multiplicative factor indeed the "density of edges" change from $\frac{d_c}{2L}$ to $\frac{d_c}{L}$, and this is all.

3.2 Modularity with Overlap

This algorithm is taken from the paper named "Modularity measure of networks with overlapping communities".
As the name says, now the overlapping of the community it is allowed. Then now we do not have a summation but an average. The results are inside a range of $[-1, 1]$.
Here the formula explained:

$$Q = M^{ov} = \frac{1}{K} \sum_{r=1}^{K} \left[\frac{\sum_{i \in c_r} \left(\frac{\sum_{j \in c_r, i \neq j} (a_{ij}) - \sum_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right)}{n_{c_r}} \cdot \frac{n^e_{c_r}}{\binom{n_{c_r}}{2}} \right] \tag{3.2}$$

Where the elements of the formula are:

- K is the number of communities $|C_r|$
- c_r is the actual community
- n_{cr} number of nodes in the community

- $n^e_{c_r}$ number of edges in the community
- $\binom{n_{c_r}}{2}$ maximum number of edges in the community
- d_i degree of the node i
- s_i number of community to which the node i belong
- a_{ij} 1 if the edge from the node i to the node j exist, 0 otherwise

Now, we try to figure out the particular things that appear in this formula.
As before we calculate the density of the community (with the equation number 3.3), but not compared to the total number of edges of the graph, but compared to the hypothetical maximum number of edges inside the nodes considered:

$$\frac{n^e_{c_r}}{\binom{n_{c_r}}{2}} \tag{3.3}$$

In addition, we consider the relationship between the number of inward and outward edges, compared to the total number of edges of the community (in the equation 3.4). Then if the outward edges are more than the inward edges, the modularity is negative otherwise is positive.

$$\sum_{i \in c_r} \left(\frac{\sum\limits_{j \in c_r, i \neq j} (a_{ij}) - \sum\limits_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right) \tag{3.4}$$

Lastly, when we look at the inward and outward edges we consider the two parameters d_i and s_i, this is important because if a node has only one edge and it belongs only to one community it has a high weight, differently if it has a hundred of nodes and belongs to dozens of communities, the weight of one of its edges is very low.

We chose this method because it allows the overlapping of communities and this is necessary for our situation. In addition is it possible to figure out which community have a high value, then increase the final results and which one decrease it, cause the low value.
From each modularity value, we can understand two things of the graph:

- If the value is negative mean that the group of nodes is not densely connected inside it, but rather is connected to some external point. It is possible that there are more inwards edges compared to the outwards, but if the value is negative this means that the outwards edges are more relevant
- If the modularity in absolute value is very near at 0 this means that the graph is extremely sparse in fact the multiplication factor (shown in equation 3.3) collapse all the elements of the formula near zero

To note that when the graph are directed only the equation 3.3 change in fact the maximum number of edges is no more $\binom{n_{c_r}}{2}$ but became $2\binom{n_{c_r}}{2}$.

3.3 Modified modularity

This algorithm is taken from the paper named "Incorporating Implicit Link Preference Into Overlapping Community Detection".
Finally, I arrive to explain this method, this is the algorithm choose in the paper of CNRL, for this reason, we looked for it. It is very important because allow our to replicate the results shown in the paper of CNRL, then we can proceed to find a fixed point. Unfortunately, this is the slowest algorithm of the three that I show you in this section because it is the most complex.
Let's look at how it works:

$$Q = \frac{1}{M} \cdot \sum_{u,v \in V} \left(\left(A\left[u,v\right] - \frac{d_{in}\left(u\right) \cdot d_{out}\left(v\right)}{M} \right) \cdot \left| C_u \cap C_v \right| \right) \tag{3.5}$$

Where the elements of the formula are:

- M stands for m if the graph is undirected and for $2m$ if the graph is directed, where m is the number of edges of the graph
- u, v are the iterators on the graph's nodes named V
- $A[u, v]$ is the weight of the edge uv if the edge does not exist this value is 0, normally a node without weight is considered as 1. It is taken from the adjacency matrix
- $d_{in}(u)$ and $d_{out}(u)$ are the incoming/outgoing degree namely the number of incoming/outgoing edges of the node u. If the graph is undirected those values are equal
- C_u is the set of communities to which the node u belong
- $|C_u \cap C_v|$ is the number of communities that node u and v share

I try to explain this equation in words.[1]
Starting from the end we see the element $|C_u \cap C_v|$, this is important because allow ignoring the edges that link two nodes of different communities, so only the edges inside a community is considered. In addition, this part is a multiplicative factor, if an edge is inside lots of communities it has a high weight if it belongs only to one community is weight is lower.
But what is the base of this multiplier factor? We can see that this is a positive value if the edge exists (thanks to $A[u, v]$), in fact, more edges a community has more it is defined. From the weight of the edge we subtract a factor (defined from $\frac{d_{in}(u) \cdot d_{out}(v)}{M}$), this represents the relevance/likelihood of the edge, if it connects two nodes with a high degree, this edge is not too important therefore we decrease its weight slightly, at the opposite is very important if connect two nodes that have low values of degree.

[1] Non sono completamente sicuro delle meccaniche alla base della modularità modificata