UCLouvain-EPL

Algorithms in Data Science

A Network-Based Study on Anna Karenina, a novel by Leo Tolstoy

Teachers:
Vincent Blondel
Jean-Charles Delvenne
Krings Gautier
Course assistant:
Alexey Medvedev

TEAM Members :
Nima Farnoodian - 68372000 - nima.farnoodian@student.uclouvain.be
Charles Rongione - 51841500 - charles.rongione@student.uclouvain.be
Breno Tiburcio - 74042000 - breno.tiburcio@student.uclouvain.be

Abstract

Anna Karenina is a very long novel by the Russian author Count Leo Tolstoy. Anna Karenina is admired by many novelists and bibliophiles as the world's most remarkable novel, and it provides a vast panorama of contemporary life in Russia and that of humanity in general. The author perfectly puts all his efforts into bringing astonishing characters into being and gripping the readers with his fanciful aristocratic love-story. The story mostly revolves around Anna Arkadyevna Karenin, Alexey's wife, who finds the Count Vronsky as her lover to consummate her passionate affairs. In this project, we accomplish a Network-Based study on the novel "Anna Karenina". We create a co-occurrence network such that the characters are linked if they are mentioned in the same paragraph at least once. Our resulting network consists of 45 actors and 289 links. Our network is in some respects analogous to the Barabasi-Albert network model with the same node number and average degree; they both represent a high average clustering coefficient, degree disassortativity, and inseparable community structure. Furthermore, the network seems to be randomly mixed according to the gender of the characters. Such an observation may spring from the fact that the story of the novel is a medley of love, secret love, and different-sex contact. Also, using greedy algorithm and independent cascade model, we found out that Prince Stepan Arkadyevitch Oblonsky and Prince Alexander Shtcherbatsky are two most influential actors in the network who can impact 18.48 actors on average.

1 Preprocessing: Characters and Network Creation

This project studies the network of characters in Leo Tolstoy's famous novel, Anna Karenina. Anna Karenina mainly contains the interactions—mostly love affairs, marriage, faith, and betrayal— among the characters in Russian royal families in the 1870s. Therefore, in our opinion, the novel could stand well for studying the characters and accomplishing our project. This novel is organized into eight parts, each of which is divided into several chapters. It has around 7705 paragraphs depending upon the translation and the publisher.

Moreover, of all characters, only 47 characters take significant roles in the story[1] — from now onwards, we only concentrate on the prominent characters. Also, of 7705 paragraphs, in only 1551 paragraphs, at least two character names appear. In these paragraphs, the average number of words is 87.58, and the average number of unique characters is 2.417.

In what follows, we describe the challenges we dealt with for processing the text and extracting the names of the characters efficiently and meaningfully and then explain the techniques that we applied to get around the challenges and also to create our network.

1.1 Challenges

Although at first glance, extracting the names from a large text may seem to be a simple process that can be handled by a naïve approach like regular expression operation, it would be inherently tricky since names never appear in a standard pattern in the text, specifically, if they are Russian archaic aristocratic names; Title, First Name, Middle Name or Father Name, Family Name, Nick Name, etc..

Maybe the right answer for the complexity of the names is to use built-in packages like spaCy and nameparser if we make use of Python; however, these packages only degrade the complexity, not solve it entirely. For instance, as we observed, spaCy sometimes makes a mistake to distinguish a full Russian name, and it indeed sometimes deems a full name as two separate names. So, in this case, it may output fake or false names, whereby our network would wrongly contain a node and its unrealistic links. Furthermore, since spaCy was trained mostly for English names, it could not recognize some names like "Stepan" in the text. Accordingly, such a character will likely never be added to the network while he is "Prince Stepan Arkadyevitch Oblonsky [2]," the brother of Anna, who is the main character of the story. [3]

The challenges are not only confined to the issues above. The Author also sometimes uses an alias or a short form of the name to mention a person. For example, the Author mentions "Prince Stepan Arkadyevitch

[1]Thanks to the internet, we could obtain a list of major characters, and therefore, we were enabled to perform text and names processing much more efficiently. A list of 45 characters is provided in https://www.bartleby.com/316/1009.html. We modified and extended this list based on our observations and the available information on the internet.

[2]To know the character, refer to https://www.litcharts.com/lit/anna-karenina/characters/prince-stepan-stiva-arkadyevich-oblonsky.

[3]Note: It is worth mentioning that, to extract the names, we first need to use spaCy (Tokenization, POS-Tag recognition, etc.), then we can apply "HumanName" method of nameparser to divide the names into their individual components.

Oblonsky" as Stepan Arkadyevitch 473 times and his alias "Stiva" 67 times in his book. Using excellent built-in packages without logic may lead to misrecognizing different people based on the observed names, while all names point out a unique person.

Lastly, we faced a challenge that we here call indirect mentioning. Indirect mentioning happens when someone's name is referred to as somebody else's brother, sister, or mother. Take, for example, the following paragraph of the novel:

Vronsky's mother, on hearing of his connection, was at first pleased at it, because nothing to her mind gave such a finishing touch to a brilliant young man as a liaison in the highest society; she was pleased, too, that Madame Karenina, who had so taken her fancy, and had talked so much of her son, was, after all, just like all other pretty and well-bred women,—at least according to the Countess Vronsky's ideas. But she had heard of late that her son had refused a position offered him of great importance to his career, simply in order to remain in the regiment, where he could be constantly seeing Madame Karenina. She learned that great personages were displeased with him on this account, and she changed her opinion. She was vexed, too, that from all she could learn of this connection it was not that brilliant, graceful, worldly liaison which she would have welcomed, but a sort of Wertherish, desperate passion, so she was told, which might well lead him into imprudence. She had not seen him since his abrupt departure from Moscow, and she sent her elder son to bid him come to see her.

In the example paragraph, we see three characters including Vronsky's mother, Madame Karenina, and the Countess Vronsky. The Countess Vronsky is indeed the Vronsky's mother. Therefore, the Countess Vronsky and Vronsky's mother should not be treated separately. In contrast, spaCy and nameparser recognize three names for three characters. As indirect mentioning may happen everywhere in the book, many wrongly found characters may be added to the resulting network, thereby a less meaningful network of the characters.

In conclusion, if we treat the problem naïvely, more specifically for this long Russian novel, we may end up with a less informative, less meaningful, and dense network whose representation only leads to fallacy and skepticism in our final analysis.

"Not everything that can be counted counts, and not everything that counts can be counted."

Albert Einstein, Physicist

1.2 Network Creation

In this section, we will briefly go through the details regarding the network creation and character extraction process. Unlike all text processing tasks that involve Sentence Tokenization, Word Tokenization, Stop-words removing, etc., extracting the character names do not need a clean and preprocessed text. Indeed, all names written in English alphabets (e.g., surname, nickname, first name, and so on) begin with a capital letter. Therefore, a simple regular expression (RE) operation that could specify strings matching "[A-Z][a-z]+" helps detect the words that can be candidates for extracting names. But, these are only words with a capital letter at the beginning. Many terms that are at the beginning of the sentences could also be deemed names because the first letter of the first term of a sentence is always capitalized in the standard English texts. In addition, a character name could consist of different components, so the RE operation separates a name, and outputs each component as a candidate name.

Although RE operation has drawbacks and may overestimate some names, it is inherently useful if it is bolstered with factual information, which in our case, is the factual character names. Take, for example, the following sentence:

When Countess Nordston ventured to hint that she had hoped for something better, Kitty was so angry and proved so conclusively that nothing in the world could be better than Levin, that Countess Nordston had to admit it, and in Kitty's presence never met Levin without a smile of ecstatic dmiration.

The above RE operation will output a list of words with the capital letter at the beginning:

['When','Countess','Nordston','Kitty','Levin','Countess','Nordston','Kitty','Levin']

In the above list, except 'When', all terms are name components. More interestingly, all words are ordered in the list as they appear in the sentence. So, for each word in position x, the words in the positions x-1 or x+1 may be the components of a real name that contains the word in position x. Using these observations,

we make use of a look-up technique to extract the real names even when the name is indirectly mentioned. It is worth saying that the aim here is to provide a better representation of the network, not an impeccable network; the errors are inevitable.

Thanks to the internet, we first gathered information and interpretations regarding the book. We could obtain a list of 47 significant characters plus their genders and their roles in the story—we put the full character names on the list. Second, we divide each character's names into components. Third, we created a dictionary, namely allnames, whose keys are the name components, and its values are the full names that include the components. We also stored the roles associated with the full names. For example, the values for the name Vronsky in the dictionary look like

'Vronsky': [('Count Alexey Kirillovitch Vronsky', 'a young officer'), ('Countess Vronsky', 'mother')]

Using this dictionary, we were enabled to learn that the name component "Vronsky" can point out two different characters with two different roles. So, if we observe "Vronsky" in the text, we look-up our dictionary to see to whom it concerns. If the name component only points out a character, the problem is simple, and then the algorithm recognizes a character and gives its full name as the output. However, the problem arises when the name component belongs to two or more characters. In this situation, the algorithm checks if which of its neighbors (in position x-1 and x+1 given that the word is in position x) is a substring of the full names stored as its values. If it finds a match (for example, 'Countess' is a substring of Countess Vronsky), it will give the full name that it matches as the output. Although this approach may mitigate the error and the complexity of the problem, there is still a problem with indirect mentioning. Nevertheless, it is simply solvable using the provided dictionary. In the dictionary, not only are the full names stored but also the roles that the characters take are stored as well. So, if no match in the full names can be found, we can examine the role of the characters. For instance, suppose we see Vrosnky's mother in the sentence. As "mother" appears after Vronsky, the algorithm can examine if the next word is a substring of the role of the name, and if it does, it recognizes that the name points out Countess Vronsky. Using this simple idea, we can extract the characters even if they are indirectly mentioned; however, it is worth noting that this solution can be useful if the names and their roles are properly gathered. For example, if the algorithm observes somebody's sister while the role of his/her is not already mentioned, it cannot detect the relation due to the lack of evidence. Moreover, our approach cannot detect the character names if they appear as his/her sister, mother, brother, and so on.

We used this approach to extract the character names in the paragraph. After detecting the names, we constructed an undirected unweighted network in which two characters are linked, provided they appear at least in one paragraph. The resulting network consists of 44 nodes/actors and 289 links/edges. Figure 1 illustrates the resulting network. In this Figure, the larger nodes names point out higher degree nodes.

Figure 1: The resulting network with 44 nodes and 289 links

2 Network Properties

In this section, we will review the network characteristics such as Degree Distribution, Correlation, Clustering, Density, Homophily, Community Structure, and so on.

2.1 Degree

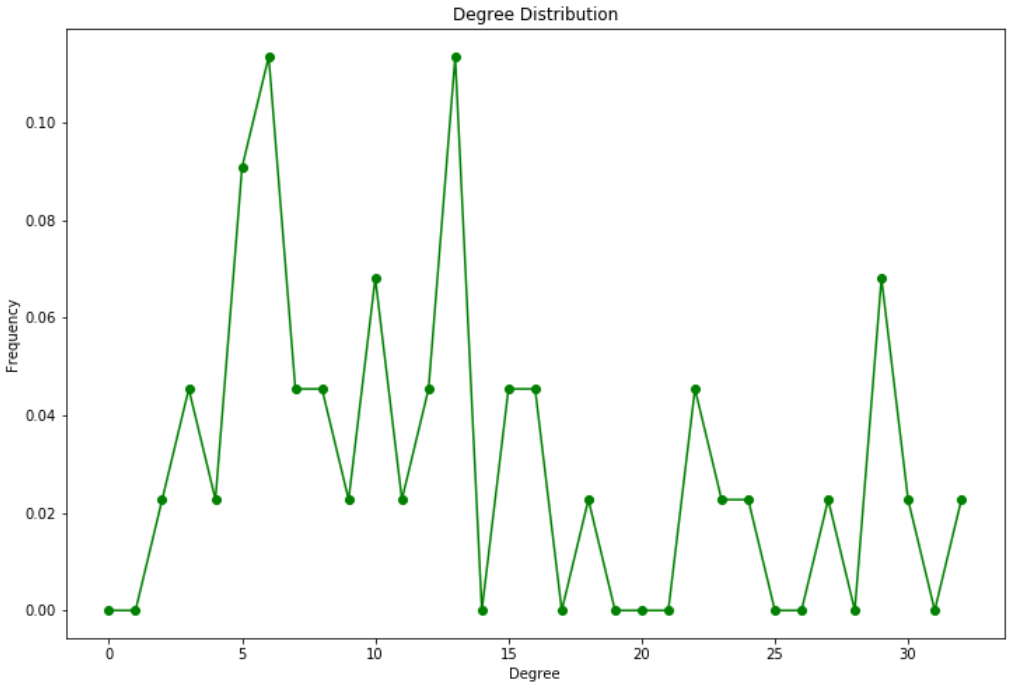

Figure 2: Degree Distribution of The Anna Karenina Network

The network includes 44 actors and 289 undirected links. Its average degree and its density are 13.136 and 0.305, respectively. An eyeball look at the degree distribution in Figure 2 reveals that the distribution does not exhibit an excellent right-skewed distribution, as observed in the many real-world networks. However, degrees 6 and 13 have the highest frequency, constituting around 27 percent of the actors.

The six high-degree actors in the network are respectively "Konstantin Dmitrievitch Levin", "Princess Ekaterina Alexandrovna Shtcherbatsky", "Anna Arkadyevna Karenin (Anna)", "" Prince Stepan Arkadyevitch Oblonsky", "Count Alexey Kirillovitch Vronsky", and "Princess Darya Alexandrovna". All these names are major characters so that the absence of each of them can change the whole story. For instance, Konstantin Dmitrievitch Levin , aka, Kostya [4] is a co-protagonist, as central as Anna herself, Prince Stepan Arkadyevitch Oblonsky is Anna's brother, Count Alexey Kirillovitch Vronsky is Anna's lover, and Princess Darya Alexandrovna is Anna's sister-in-law. As it's clear, all these characters have a tied relationship with the main character, so it is expected that the Author more frequently mentioned their names or aliases in the paragraphs.

The degree assortativity for our network is -0.248. This negative value indicates that the network is degree disassortative, meaning that high-degree actors in the network associate preferentially with low-degree actors.

[4]To know the character, please go to https://www.litcharts.com/lit/anna-karenina/characters/konstantin-kostya-dmitrich-levin

It may come from the fact that the names of the significant characters and marginal characters almost always appear together in the paragraphs.

2.2 Clustering Coefficient

The network appears to be highly clustered, meaning that the connections among the actors are dense. The average clustering coefficient for the network is 0.726, which is too high. We observed that the marginal characters are mostly linked to major characters who constitute the core of the network. And, since these major characters form the whole story, they are frequently mentioned together in the same paragraphs. Therefore, the chance that the characters linked to a marginal character are also linked together is high, thereby high average clustering coefficient. Below, we provide a list of six characters with the highest clustering coefficient. It is worth noting that all these characters are marginal and possess a high clustering coefficient of more than 0.95 and a low degree, on average, 4.33.

Mlle. Linon, Kritsky, Marya Philimonovna, Madame Sviazhskaya, Captain Kamerovsky, Varya

2.3 Homophily and Assortativity based on Gender

In this section, we will examin assortativiy based on the gender of the characters. As Figure 3 shows, almost all characters prefer to be linked to the characters with different genders. To have a quantative analysis for assortativity, we use Mark Newman's Modularity to compute the assortitiveness in the network. The modularity calculated based on Gender is 0.011. A cursory look at the Modularity in the network and the depicted network unveils that no strong gender-assortativity exits between actors. Generally, the story of the book "Anna Karenina" is fully composed of Love, Affair, Extramarital Relationship, and infidelity. Therefore, the network represents a sexual contact among actors. In other words, in a paragraph, the chance to see the names with differnt gender is more than to see same-sex names. However, based on the modularity, the network seems to be randomly mixed if the gender of the actors is taken into account.

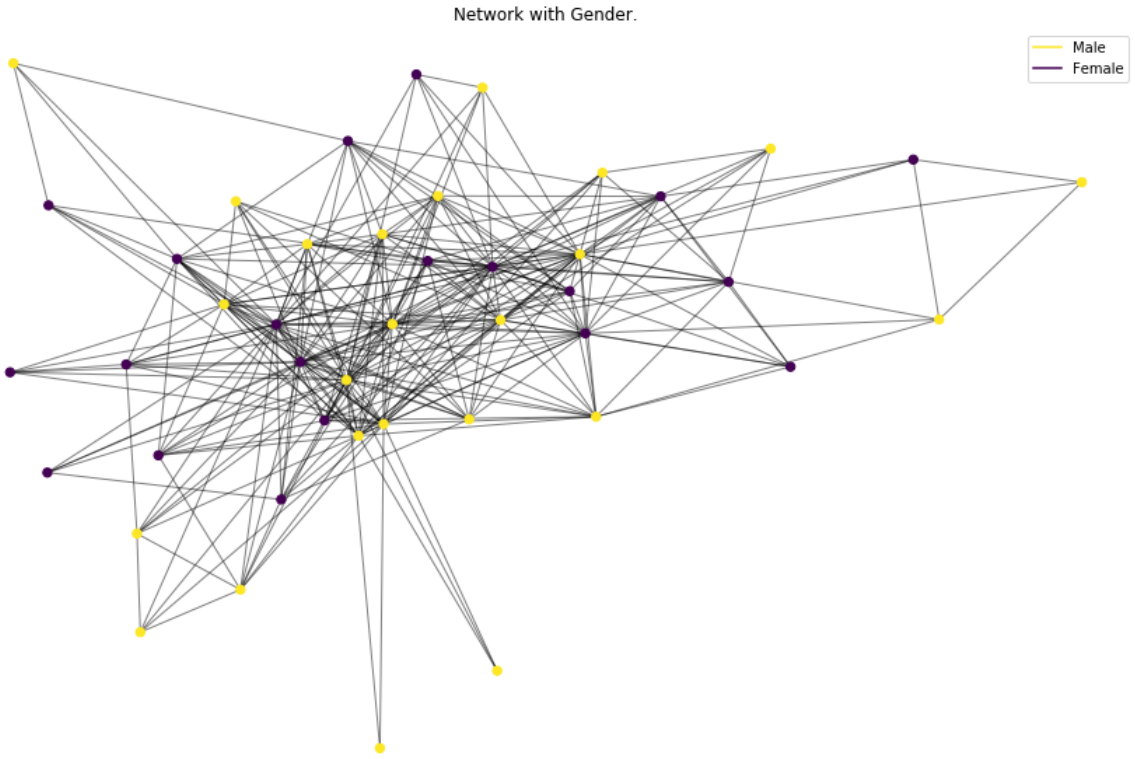

Figure 3: Network with actors' gender

2.4 Community Structure

Nowadays, the real-world networks including Social Networks, Economy Networks, etc. are understood to represent unique characteristics such as sparseness, high clustering coefficient, community structure, and so on. Among all these characteristics, community structure has recently drawn many researchers' attention due to its great applications. The revelation of this hidden structure can help better understand the social patterns existing in the networks like how the nodes are organized in the network, how they can impact each other, etc.

Generally speaking, the existence of the community structure in a network may come from the fact that the nodes have a strong tendency to be linked to other nodes having something in common. For instance, in the society, the people who have similar attitudes, the same language, related jobs, or even who live in the same city probably would rather have closer relationships with each other than with those who share nothing.

Technically, although there has been no exact definition for the communities, a community is typically referred to as a set of nodes that represent a dense connection between each other and a sparse connection to the rest of the Network. To find a community or set of communities, a criterion or scoring function is often used to measure the quality of the detected communities, or at least, to provide a quantitative measure that could explain the separability of the Network. Modularity, proposed by Mark Newman, is a well-known criterion for this purpose. One of the state-of-the-art and fast algorithm that utilizes Modularity is Louvain community detection. We use this algorithm, as suggested, to detect the possible community structure in our Network. This algorithm tries to find a set of communities that can maximize the Modularity; the higher Modularity, the better network separation. The outcome of this algorithm on our Network is a community structure with four communities, sometime three communities, and a Modularity score of 0.191. The low Modularity for our Network indicates the inseparability of the Network. By inseparability, we mean that the Louvain Algorithm could not find the communities that are well separated from the rest of the network. Figure 4 depicts the network with its communities detected by the Louvain Algorithm.

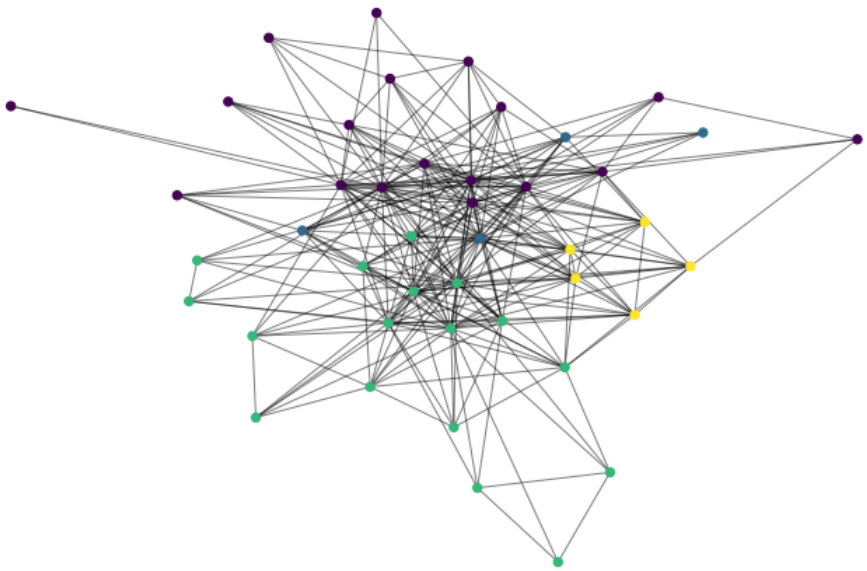

Figure 4: Detected Communities in our network. Each color represents a Community.

2.5 Summary

In this section, we provide a full summary of the characteristics of our Network "Anna Karenina". Table 1 shows the summary.

	Value	Description
Avg. Deg	13.136	-
Density	0.305	Relatively High
Degree Assortativity	-0.247	The appearance of the significant and marginal characters together
Avg. Clustering	0.726	The connections among the actors are dense.
Diameter	3	Low
Avg. Path Length	1.779	Low
Gender-Assortativity	0.0011	The network is randomly mixed based on the gender of the actors
Q for Community set	0.191	Four communities observed using the Louvain Alg.

Table 1: Network Characteristics

3 K-Core Decomposition

3.1 Explanation

A k-core of a graph is a sub-graph in which every node has a degree of at least k, which could be any integer. For example, the 0-core of any graph is always the graph itself. If there is a non-empty k-core, the graph has degeneracy at least k, and the degeneracy of the graph is the largest k, we could decompose the graph. It is useful for identifying the clusters of a graph.

3.2 Implementation

The difficulty was to implement it without using the built-in function of NetworkX. We created a for loop that goes through all nodes, see if a node has a degree lower than K and if so, remove it. But if you do that without ordering the nodes, it will not be consistent because a degree of a node can become lower than K after the algorithm has checked, and it will not be removed. That's why we had to implement a sorting function that sorts the nodes by their degrees and then apply the algorithm.

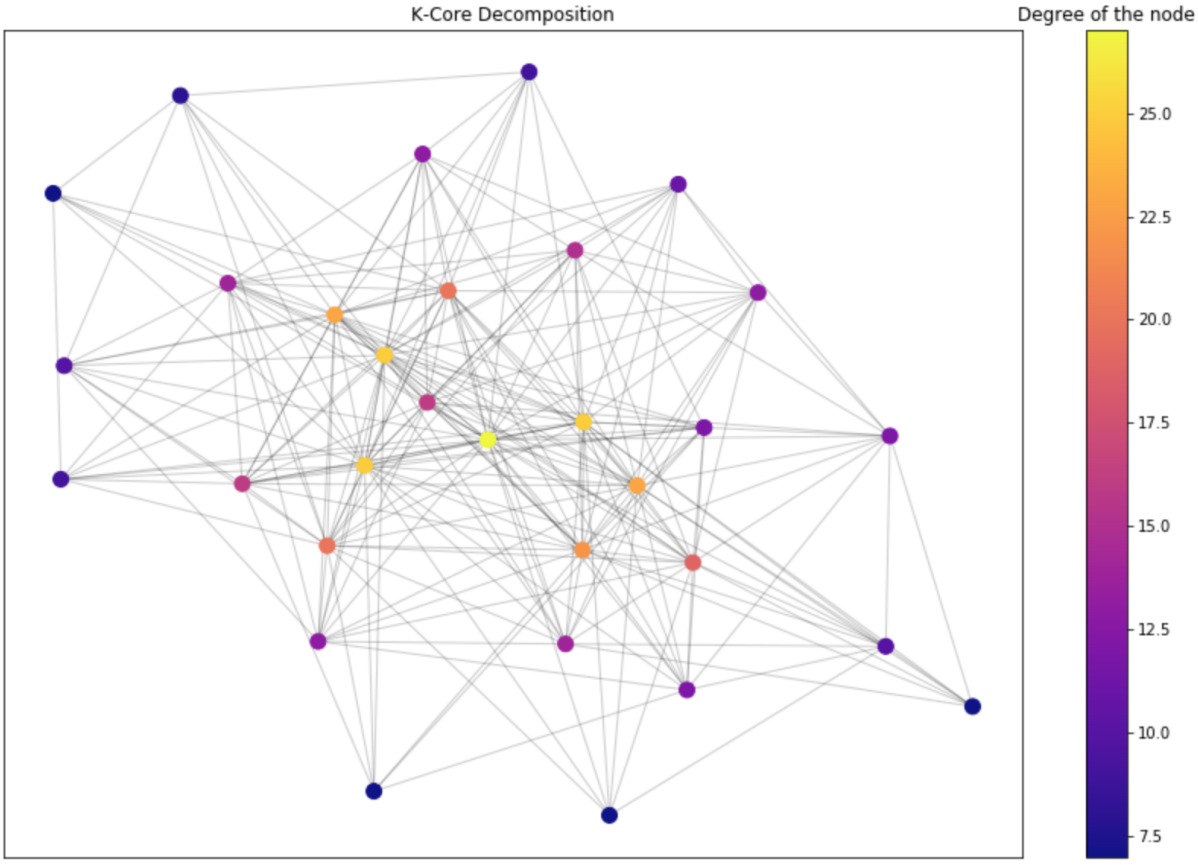

Figure 5: 7-core decomp

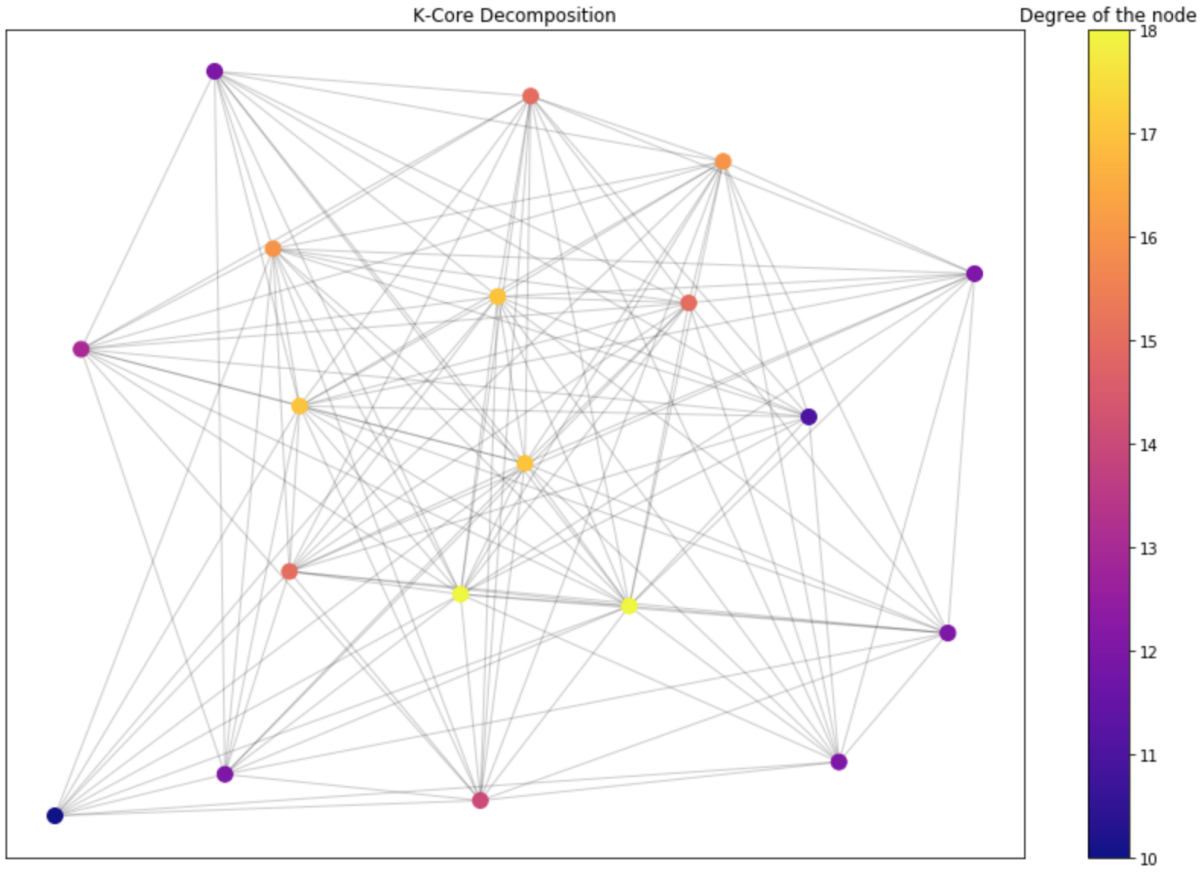

Figure 6: Max core decomp

Our Graph has a high mean degree and that's the reason why either we have many nodes either we have none

This is our max-core subgraph, it still has 18 nodes. There were no unlinked subgraphs for any of the k-core subgraphs, which is likely to indicate that there are no clusters

4 Disease/Rumor Spreading

4.1 Influence Maximization: Greedy Algorithm

In the influence maximization problem, we try to find k nodes that have the greatest influences in the network. (These nodes are sometimes called spreaders that maximize information spread.) Although many algorithms for this problem have been proposed up to now, among all these algorithms, only a Greedy algorithm is well known to provide the best solution. This algorithm finds the node with the most significant spread, adds it to the seed set, and then finds the node with the next biggest marginal spread over and above the spread of the original and so on until k seed nodes are detected. The authors demonstrate that the algorithm is theoretically guaranteed to choose a seed set whose spread will be at least 63 percent of the spread of the optimal seed set. In this algorithm, to simulate the influence spread, the "Independent Cascade" model (ICM) is used as a spreading function— though there are many other spread functions like the Linear threshold.

We utilized this greedy algorithm in order to find 5 percent of the actors (for our network, k=2) who are the most influential. This algorithm has special parameters such as the propagation probability denoted by P and the number of Monte-Carlo simulations denoted by M, which should be tuned accordingly. To find the two most influential actors, we set P=0.1 and M=100. Based on the results of the algorithm, the most influential actors happened to be "Prince Stepan Arkadyevitch Oblonsky", the brother of Anna, and "Prince

Alexander Shtcherbatsky", Anna's sister-in-law's husband. In total, the biggest influence that these two characters could achieve is 18.48 people on average. Hypothetically, if we consider the influence as infection, therefore, if these two characters get infected, around 19 people will get infected.

4.2 Examing three strategies for diffusion: K-most influential actors versus Large-degree actors and Randomly chosen actors.

In this section, we will examine three strategies for diffusion in our network; the first approach is the two most influential actors found by greedy algorithms, the second is two large degree actors, and finally is the randomly chosen actors. To compare these strategies, we used SIR model as well as ICM. It is worth noting that in the SIR model, we only iterated the diffusion/infection for each strategy once, whereas we iterated it 100 times for each strategy in the ICM; therefore, the results of the ICM is just an average. The selected actors for each strategy are as follows:

- Two most influential actors: Prince Stepan Arkadyevitch Oblonsky and Prince Alexander Shtcherbatsky.
- Large degree actors: Konstantin Dmitrievitch Levin and Princess Ekaterina Alexandrovna Shtcherbatsky.
- Randomly chosen actors: Anna Arkadyevna Karenin and Golenishtchev

4.2.1 SIR Model

A cascade model is a SIR (Susceptible, Infected, Recovered) type model often used in information diffusion or in disease spreading modeling. The principle is that at each step of time, if a node is "infected" (this term can also mean "victim of a rumor") it will make an attempt to infect his neighbors. This attempt will succeed with a probability of p. If a node has been infected for a predefined amount of steps of time, it becomes a Recovered node and is now immune and can't be infected anymore.

In order to implement the cascade model we made an algorithm that allow us to select one or multiple nodes from which the disease will start. For each step of time, the algorithm will select the infected nodes. For each one of them, it goes through its neighbors and then for each neighbors draw a random number between 0 and 1. If the number is higher than the configurable infection probability, it becomes infected. After each infection process, a recovery process exists. The algorithm goes through the list of infected nodes and check if they have been infected for more than a configurable amount of steps of time, it sets a "Recovered" attribute and the node becomes Immune, whereby it can no longer get infected. For our experiment, we chose an infection probability of 0.2 and set the recovery time equal to two.

Figures 7-9 illustrate the process of diffusion of each strategy in the network only for four time steps. As seen in the figures 7-9, Greens are recovered actors and blacks are the infected ones, we understand that due to the high mean degree and high connectivity of our network, a rumor or an infection will spread really quickly through it. Even if the diffusion process begins with random actors, almost 14 actors are still infected at time t=4, and considering the recovery, 19 actors got infected in total. It is worth mentioning that, you might observe that the k-most strategy could impact the less actors compared to the large degree strategy. Although it might be surprising, it should be insisted that the plots only show four time steps and since it is a random process, the infection will be more or less in the next time steps. Therefore, we made use of ICM to provide better results for comparing the different strategies.

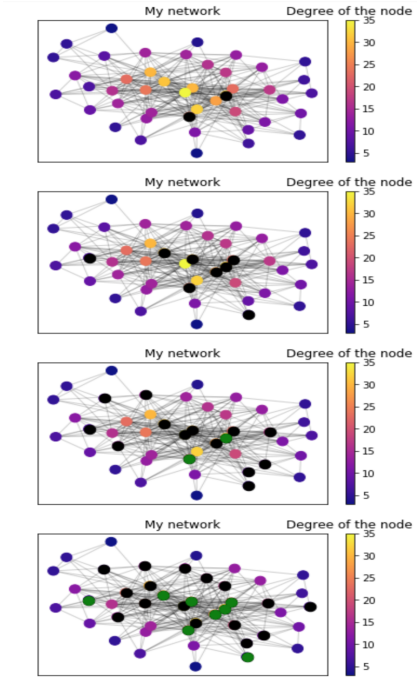

Figure 7: SIR Model for Top-k Influential actors.

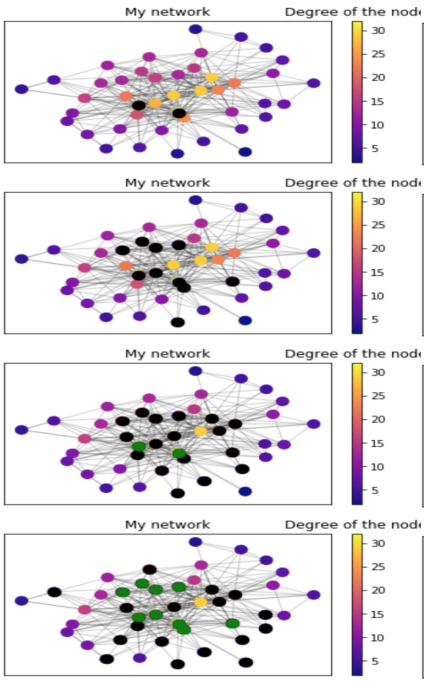

Figure 8: SIR for Large-Degree node Strategy

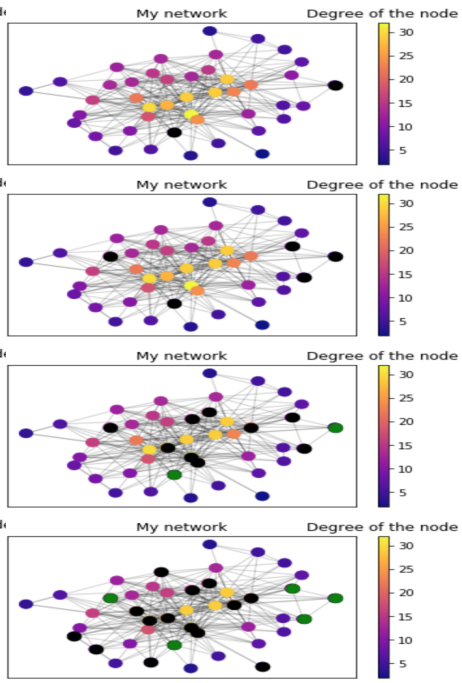

Figure 9: SIR Model for random actors

4.2.2 Independent Cascade Model

In this section, we will show the diffusion rate by each strategy based on ICM score. ICM calculates the expected spread of a given seed set, in our case the chosen actors, by taking the average over a large number (M) of Monte Carlo simulations. For our ICM experiment, we set the propagation probability equal to 0.1 and M=100.

As it can been seen in Figures 10-12, almost all strategies could reach the same infection/diffusion rate. However, it is observed that the infection rate in the k-most strategy increases sharply until time=3; it could almost infect 40 percent of the actors until this time, while two other strategies could reach around 37 percent. Precisely speaking, k-most influential actor technique could infect 18.48 actors at the end of the process while it is 17.99 and 17.42, respectively for other techniques.

With these observations, we can only declare that the k-most influential actor technique could outperform well, as expected, although the difference is negligible. It may originate from the fact that the network is relatively connected and also mixed. Moreover, the average clustering coefficient is too high in the network, which may cause an actor, who did not get infected by a neighbor x, will get infected by another neighbor y who indeed got infected by the neighbor x. In the section 2.4, we have seen that the network is inseparable in terms of community structure. So, the reason that these strategies have a similar infection rate on our network is that the infection can simply spread over the network because the infection is less likely to get stuck in a community.

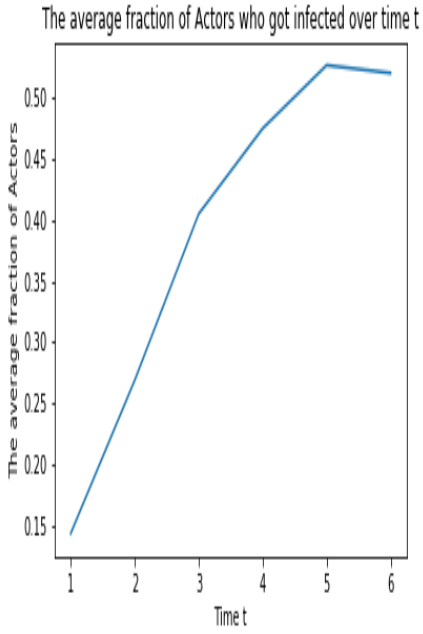

Figure 10: ICM Model for Top-k Influential actors.

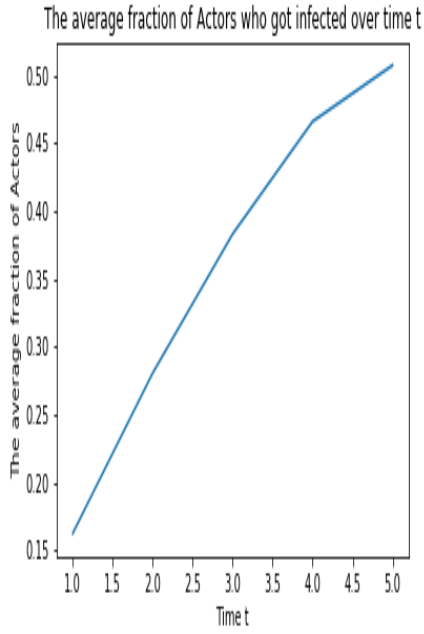

Figure 11: ICM for Large-Degree actor Strategy

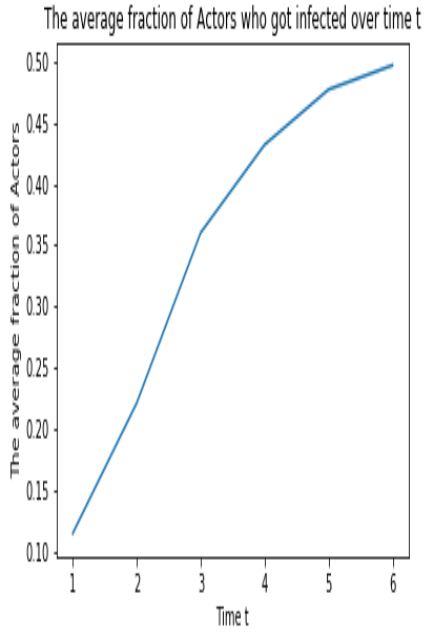

Figure 12: ICM Model for random actors

5 Preferential Attachement (Barabási-Albert)

5.1 Explanation

Preferential Attachment is network models commonly utilized to evaluate networks growth and the characteristics intrinsic to this process. By definition, the probability of a new node to attach to a particular node is proportional to this specific node degree.

In order words, as higher as the node's degree is, higher are the chances to him be connected to a newborn node. Hence, older nodes are more likely to achieve higher degrees than the following nodes. As long as the network total degrees increases, the probability of newborn node establish further connections reduces.

The preferential Attachment is widely known for its capacity to represent how cumulative advantage performs in specific scenarios, for instance, wealth distribution. Now, it is up to us analyze and comment on how it matches the network we extracted from Leo Tolstoy romance.

At the following subsections, we will describe the implementation of Barabási-Albert model, and its results.

5.2 Implementation

We utilized Barabási_Albert_Graph function (from Networkx 2.1) to generate our network. This function is base on two variables: the final node number of our graph (N); and the number of edges that each newborn node will connect to the pre-existing network (M).

At our model, we opted for a fixed number of edges of eight to attain an average network degree of 13.090 (0.03% inferior to the Anne Karenina Network). It means the newborn nodes will connect to eight pre-existing nodes.

The number of edges doesn't necessarily have to be fixed, being possible to vary along with each node addition. It is important to mention that, although the number of edges influences the degree distribution

equation, the Power Law distribution (commonly utilized to explain behavioural growths) is unrelated to that.

5.3 Results

In comparison to the book network, our Barabási-Albert model presented the Clustering Coefficient significantly lower: 0.387. Almost half of the Anne Karenina Network despite the same number of nodes and an approximate average of edges.

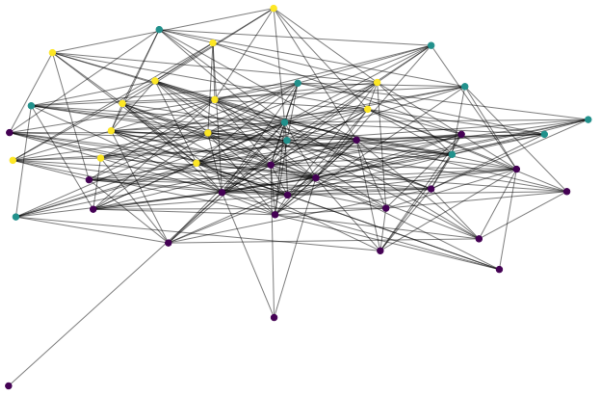

Figure 13: Barabási-Albert Network with Communities.

By that, we can assume that the preferential attachment network generated fewer triangles among the nodes than the Anne Karenina Network. The assumption is that the novel's plot demands a higher interconnectivity among its characters. An expected result if we take to account the Preferential Attachment models particularity towards cumulative advantage.

In regards to the Assortativity Coefficient, the Barabási-Albert model presented a value of -0.144 while the book's network -0.248. The negative value means that high-degree nodes associate preferentially with low-degree nodes in both networks affirming a dissortativity characteristic in both networks.

In terms of communities, it was possible to identify four communities in the Preferential Attachment model using the Louvain Algorithm. One community more than the network extracted from the book. The positive modularity values of 0.17 indicates that the number of connection is superior to the expectation reaffirm its dense connectivity, although yet inferior to Anna Karenina network. The algorithm divided the book network into four communities with a modularity value of 0.191.

Plotting the degree distribution, we could visualize better how a network with the same numbers of 44 nodes and approximate edge average can differ structurally.

To achieve the closest degree average, we set a fixed number of edges, per new node to eight. Therefore, each new node has an initial degree of eight. As a consequence, the frequency distribution peaks between eight and 13 degrees.

Curiously, there is one node with a degrees lower than eight (less than the initial degree of a new node) which, undoubtedly, it refers to a pre-existing node. What leads us to conclude that, in this case, the Preferential Attachment did not embrace the oldest nodes with which the network started. Thereby, the upmost degrees are likely to be the nodes added firstly.

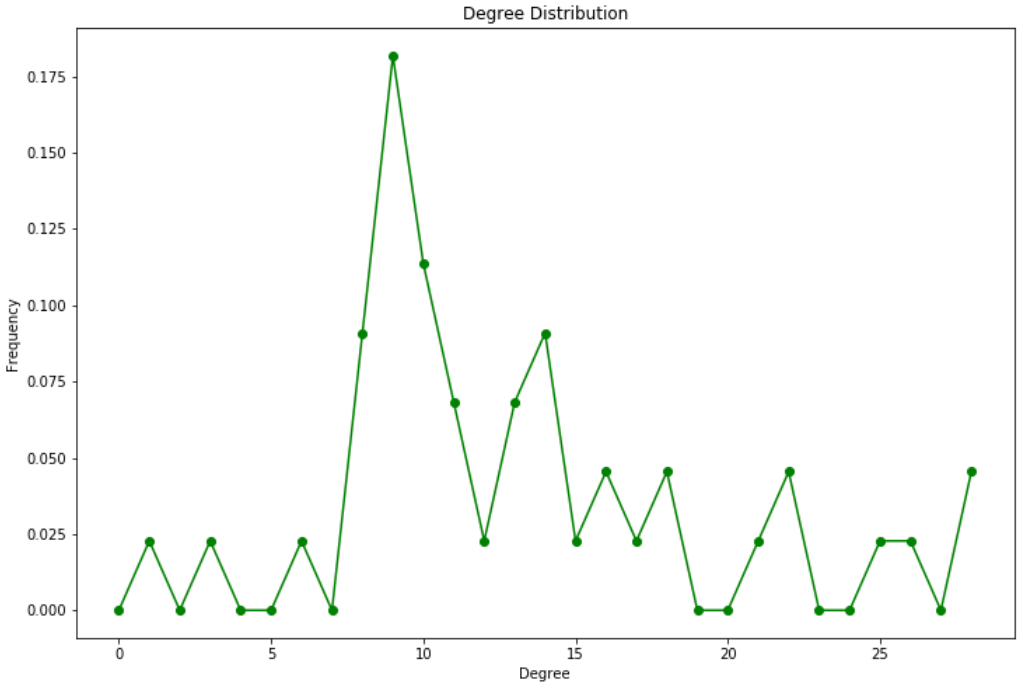

Figure 14: Barabási-Albert Degree Distribution.

The relatively small number of nodes express a degree distribution that timidly resembles the Power Law. That manner, as long as the network grows, it will approximate the "Fat Tail" curve.