Individual node's contribution to the mesoscale of complex networks: A modified implementation.

Wilmer Uruchi Dominik Stefanicka

June 18, 2020

Abstract

In the following work we explore the research done in [6] about measuring the contribution of individual nodes to the mesoscale of complex networks in the context of a *Twitter* discussion. In the original work they use undirected networks, we apply the necessary changes to adapt the approach to directed networks. Then, we calculate the features mentioned in the original paper for each node and build a Graph where the nodes are the participants of the discussion and the edges the messages they sent to other participants. Finally, we study how we can use Machine Learning and Deep Learning methods to characterize the Graph result as a network that shows conflict or a network with no conflict.

1 Introduction

Representing a real system as a network allows for the abstraction of some key concepts and the study of the components of the system using the properties of network models, and the mathematical tools that they provide. In this work we take the model proposed in [6] that was originally aimed at understanding the biological function of observed anatomical structures, and we apply it to analyze *Twitter* users interaction for a given period of time and topic.

We propose some changes to the original model because our subject of study (a twitter network) has a directed nature; conversely, those studied in the original paper are more on the undirected spectrum. Consequently, some changes are implemented in the mathematical formulation of the metrics studied in the model. Furthermore, a new set of experiments is added to test these changes.

The idea for this work started as an study by *Universitat de Barcelona* physicists who tried to identify some key roles in the communication structure of the users of *Twitter* during specific social events, i.e. strikes, political unrest, or even just a plain day. That work was based in metrics that were proposed in previous studies of similar nature.

Along the process of experimentation of the original work, the researchers found that it could be useful to change some of the formulas to calculate the features of each node (user) in the network. This work is an attempt at analyzing the interaction on networks using those improved formulas, and also adding the new dimension of dealing with directed networks.

Furthermore, as another step in the process, after the feature for each node have been calculated and the Graph has been defined and constructed. We plan to apply Machine Learning and Deep Learning methods in a semi supervised model to find an underlying structure or intuition that tells us if the graph (network) presents conflict or not. This would be mostly an exploratory study of current methods.

The code used in this project has been uploaded to a GitHub repository referenced in [3].

2 Characterization of a node's role in a modular network

In this section we go very quickly through the mathematical basis of the model. Further details can be found in the original paper [6]. We will put special emphasis into the changes we applied for our implementation.

2.1 Community Finding

For the purposes of community finding, we make use of *Infomap* [1], a network clustering algorithm based on the map equation [2]. *Infomap* is well suited for directed networks, although its options allow for experimentation with a wide array of network types.

Community finding will always be performed with the original directions of the edges in the dataset. **The community finding process, also known as clustering, is the first step in our workflow.**

The **purpose** of this first step is to identify communities inside the network sample and also identify to which community each participant (node) belongs to. This value will be used for the subsequent calculations.

2.2 Hubness Index

Hubs are typically defined as those nodes with many more connection than others, in this section we formalize the calculation of this feature.

The main idea of this calculation is to compare the degree of a node in a community, to the degree this node would have in a random generated graph with the same density as the sample.

We define hubness of a node in a network of size N and density ρ as:

$$h_i = \frac{k_i - \langle k \rangle_R}{\sigma_R} = \frac{k_i - (N-1)\rho}{\sqrt{(N-1)\rho(1-\rho)}}$$

Where $\langle k \rangle_R$ is the mean out-degree of the equivalent random graph against which the out-degree k_i is compared. The hubness is negative $k_i < \langle k \rangle_R$ for nodes less connected than expected from randomness. In general, **the greater the value of the hubness index, the more confidently we can state that the referenced node is a hub**.

We characterize nodes by two hubness values: **local hubness** h^l and **global hubness** h^g, where h^g is the hubness in relation to the network and h^l is the hubness related to the community to which the node belongs. For h^l, $N^{'}$ and $\rho^{'}$ take their values from the community.

It is important to mention that both **local hubness** and **global hubness** can take negative values.

2.3 Participation and dispersion indices

The **participation vector** P_i is a vector whose elements P_{im} represent the probability that node i belongs to community C_m, where $m = 1, 2, ..., M$. The probability is defined by: $P_{im} = \frac{k_{im}}{N_m}$, where N_m is the size of the community, k_{im} **assumes the value of the out-degree**. Participation vectors are normalized such that $\sum_{m=1}^{M} P_{im} = 1$. As an example, if $M = 4$, and node i belongs only to community 1, then $P_i = (1, 0, 0, 0)$. If node were connected to all communities, then $P_i = (1/4, 1/4, 1/4, 1/4)$.

Then, we define the **participation index** $p_i = 0$ if the node devotes all its nodes to a single community, and $p_i = 1$ if it is equally connected among all communities.

$$p_i = 1 - \frac{\sigma(P_i)}{\sigma_{max}(M)} = 1 - \frac{M}{\sqrt{M-1}}\sigma(P_i)$$

The larger the p_i the more difficult is to classify it into a single community.

Then we define the dispersion index d_i in a similar way than p_i but only considering non-zero entries.

$$d_i = 1 - \frac{\sigma(P_i^{'})}{\sigma_{max}(M^{'})} = 1 - \frac{M^{'}}{\sqrt{M^{'}-1}}\sigma(P_i^{'})$$

The **dispersion index** is a measure of how difficult is to classify a node. It is determined by the formula:

$$p^{+}(d, M) = 1 - \sqrt{\frac{M}{M-1}\left[\left(\frac{2-d}{2}\right)^2 + \left(1 - \frac{2-d}{2}\right)^2 - \frac{1}{M}\right]}$$

However, not all combinations of dispersion and participation index are possible. We will comment about it on Section **??**.

2.4 Data

The primary data consists of three datasets:

- Marta_Rovira: A dataset that shows some community structure.
- Vaga8Nov: A dataset that shows strong community structure. There are three communities that include the majority of nodes.
- nochebuena: A dataset with weak community structure. Serves a null model.

We also mined more datasets on our own directly from Twitter API. We used hashtags: #BLM, #COVID19, #primavera, #quarantine and #santjordi.

2.5 Application of formulas to Twitter datasets

The formulas described in the paper have been implemented using C++ in the following way:

1. The program reads an ".edge" file and transforms the input into a Graph object defined for this implementation.
2. The Graph edge is processed and a link-list formatted file is created.
3. The link-list file is used as input for Infomap.
4. Infomap is called as a command (This implies that an installation of Infomap already exists), the link-list file is sent as input, as options we define "-N 5 –directed" so the algorithm runs 5 iterations and assumes the input describes a directed network.
5. Infomap runs independently and produces a ".tree" output with the details of the results of the community partition.
6. The program reads the ".tree" file and starts to calculate the individual metrics mentioned in the previous sections, starting with the **participation vector** and ending the global and local hubness.
7. The result is 2 ".graphml" that store all the calculated metrics as node attributes.
8. Then, these ".graphml" files can be used in any platform that implements a library that can parse them. For example, *python* using the *networkx* library.

All the code and results can be found in [3]. It will be continually updated with further implementation as the research project continues.

2.6 Pre-processing Analysis

In this section, when we refer to some graphic or result as out-degree if the corresponding metric has been calculated using the out-degree of the node; then, in-degree refers to results using the in-degree of the node.

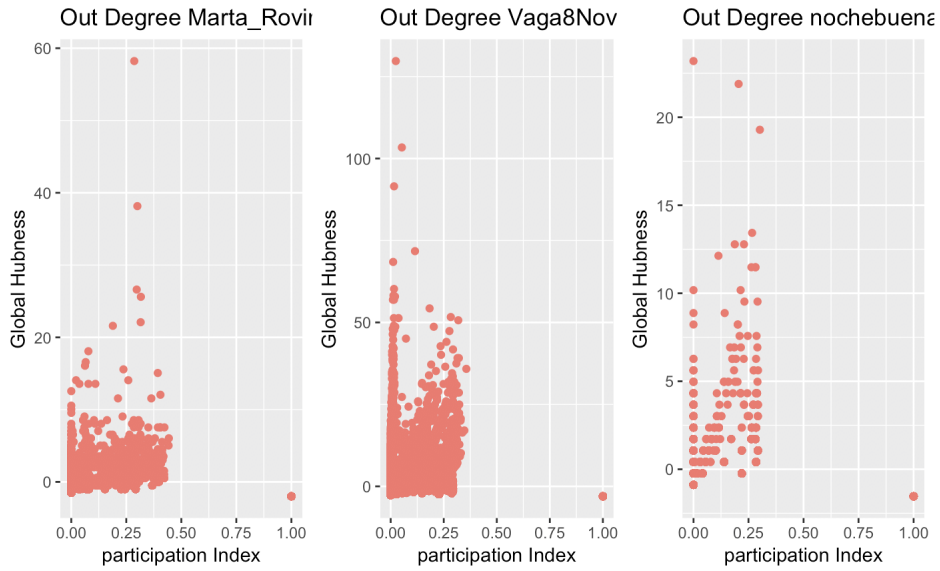

Figure 1: Participation Index compared to Global Hubness for Out-Degree

Global Hubness and **Participation Index** are two of the most useful metrics calculated. Global Hubness h^g tells which nodes have a higher connection in the whole network. Participation Index tells us how distributed is the degree of a node among communities, $p_i = 0$ if the edges of a node are located in a single community.

We observe in Figure 1 that *Marta_Rovira* presents a high concentration of nodes with low h^g and low p_i. These metrics give some intuition into what happened during the timespan this data was retrieved, i.e. participants tended to concentrate their interactions with members of their communities. Higher values for h^g represent those participants that referenced through retweets or comments to a great number of participants from other communities, in *Marta_Rovira* these are few.

Vaga8Nov shows a similar distribution than *Marta_Rovira*, but in this case p_i is even lower, suggesting a higher quantity of participants or nodes that did not choose to interact with other nodes that did not share their opinions.

The dataset *nochebuena* shows a similar trend, however we know that this dataset does not present strong community structure, so we can attribute the low p_i to the fact that there are not big enough communities that would make this index grow in a way that will

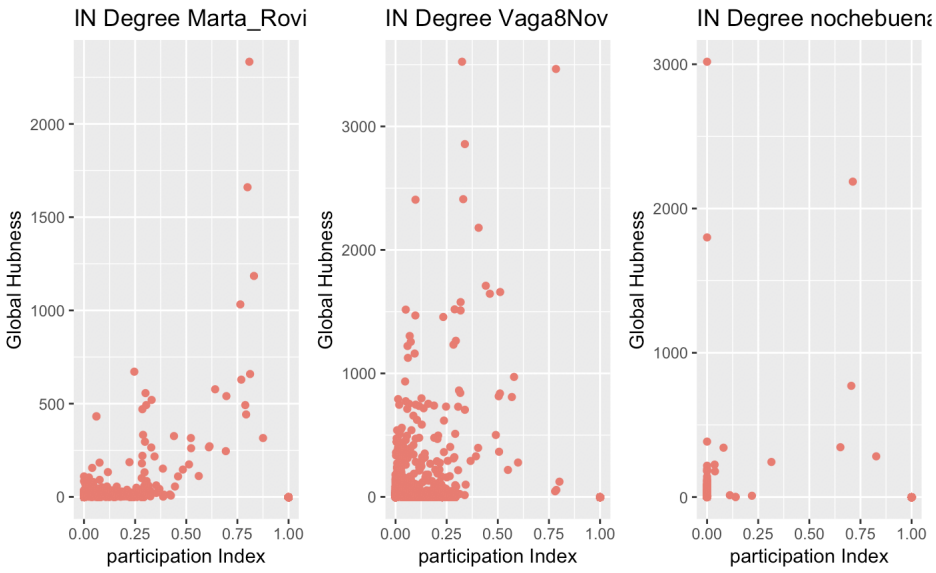

Figure 2: Participation Index compared to Global Hubness for In-Degree

make it comparable to the results from the other datasets.

The three plots show a cluster of points with $p_i = 1$. This is due to the fact that the definition of the calculation of this metric does not include the case where the out-degree is 0, same for the in-degree. So, to avoid a division by 0 scenario, that code just returned 0 in those cases. Perhaps it is best to give NA as a result for the p_i in these cases, but we will leave it at 0 for now.

We see in Figure 2 the scale corresponding to h^g increases significantly, suggesting that there are nodes with high in-degree. Moreover, p_i signals that the in-degree of the nodes seems to come from a more varied source than the out-degree. We can say that people tend to reference more people outside their community, and as a consequence, they receive less references from people in their own community. We can summarise this interaction in one sentence as: *People that talk with their friends about other people.* *Vaga8Nov* seems to follow the trend described by *Marta_Rovira*.

On the other hand, *nochebuena* shows results corresponding to a network with weak community structure, low p_i and low h^g. The node with high h^g in Figure 2 for *nochebuena* corresponds to a user that represented an event.

Analyzing Figure 3 we see that the results for *Marta_Rovira* look almost identical to that of *Vaga8Nov*. Comparing Figures 2 and 3 for *Vaga8Nov* we see that the scale for local hubness is almost double that for global hubness, suggesting that there are nodes with high in-degree inside and outside their communities. Intuitively, these nodes should be the same. The results for *Marta_Rovira* and *nochebuena* suggest a similar trend.

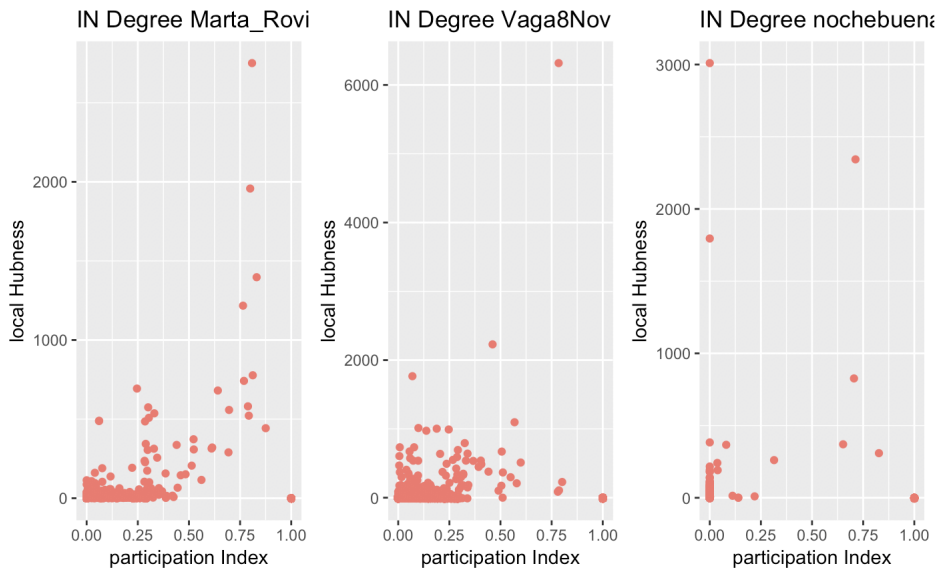

Figure 3: Participation Index compared to Local Hubness for In-Degree

2.7 Machine Learning data insight using Clustering

We want a little insight into the data. What are the possibilities to create the clusters of the data and how could be the networks possibly classified. We will try **DBSCAN** method [12] for clustering by using **Scikit-learn** [5] library. The library implements this clustering algorithm that we will test in this part of the project.

We start by taking the **.graphml** files obtained in Section 2.5 and reading it into the **DataFrame** object. Every user in the **DataFrame** is represented by 4 relevant features:

- Dispersion Index
- Participation Index
- Local Hubness
- Global Hubness

With the data in the structure we proceed to clustering. DBSCAN method is based on density of the data and create clusters of different shapes. It has 2 hyperparameters - Epsilon and **Min_pts**. For each dataset we try several pair of hyperparameters for DBSCAN.

Before using the data in clustering method we need to transform the matrix of features into something less dimensional. For this transformation we use **Principal Component Analysis** (PCA) that combine the attributes into a simpler dimensional variant.

Then we use this transformed data in DBSCAN method. For each dataset and pair of hyperparameters we gain results as a labels for each entry in the dataset. Since it is quite hard to visualize high-dimensional labeled data, we use some clustering quality criteria for evaluation.

After we get the results we save them and count result scores for evaluation the evaluation. Since we do not have golden data with the right classification, we consider 2 internal score indexes during the clusters evaluation [11] that mirror quality of possible clusters:

- **Davies-Bouldin index** (DBI) - the lower the better
- **Silhouette coefficient** - from interval (-1, 1). The higher the better

These indexes are just for help. They do not guarantee if the clustering is right. They reflect if the clusters are compact and separated from each other.

Then we also try to visualize the results by training T-distributed Stochastic Neighbor Embedding (TSNE) method. It is a machine learning method for visualization of high-dimensional data. It converts similarities between data points to joint probabilities [10].

2.7.1 Results

We use the process mentioned in the Section 2.7 on all of our datasets described in Section 2.4.

In general we can say that the lower number of clusters is the better the evaluation scores are. Although we can not certainly say the number of clusters we can see that there are certain differences between the data thus quite fine clusters can be created. From this we can deduce that it exists a pattern or structure that allow us to classify networks.

In this report, we insert the result graphs just for the main datasets. All the other graphs including some clustering visualization can be found in the GitHub repository in the file directory "/Jupiter/clustering/results".

In the following figures we can see number of clusters and scores of indexes for each tested pair of hyperparameters. We use all combinations of following values of hyperparameters:

- **Epsilon**: [0.1, 0.3, 0.5, 1, 1.5, 2]
- **Min_pts**: [2, 3, 5, 8]

First three Figures 4 5 6 are for *Marta_Rovira* dataset.

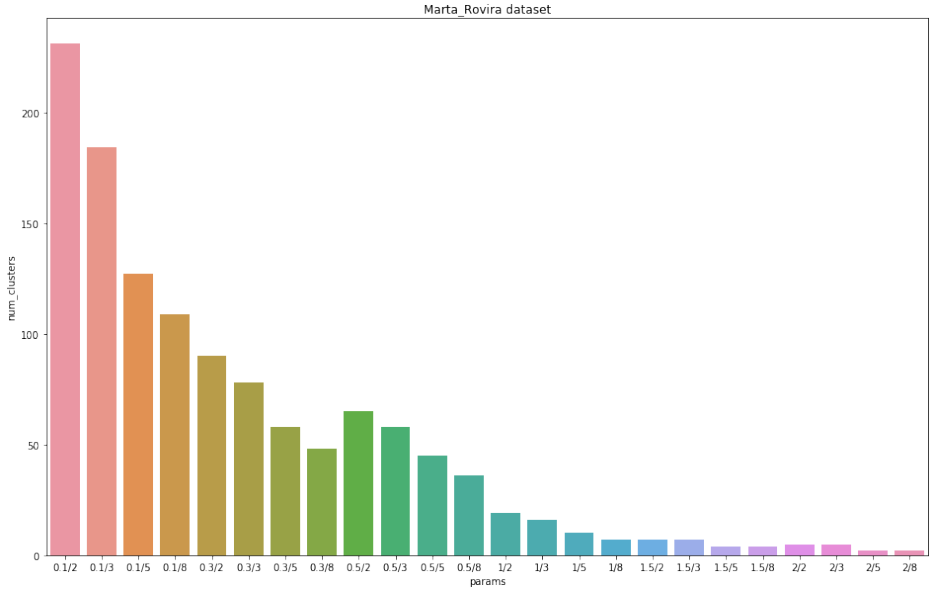

Figure 4: Number of clusters across the different hyperparameters for *Marta_Rovira* dataset.

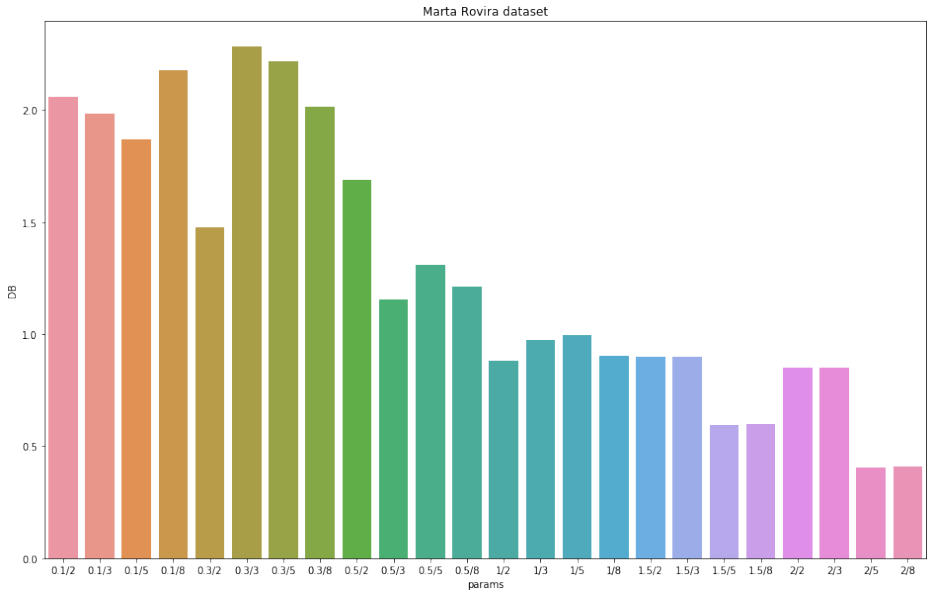

Figure 5: Davies-Bouldin index score compared across the different hyperparameters for *Marta_Rovira* dataset.

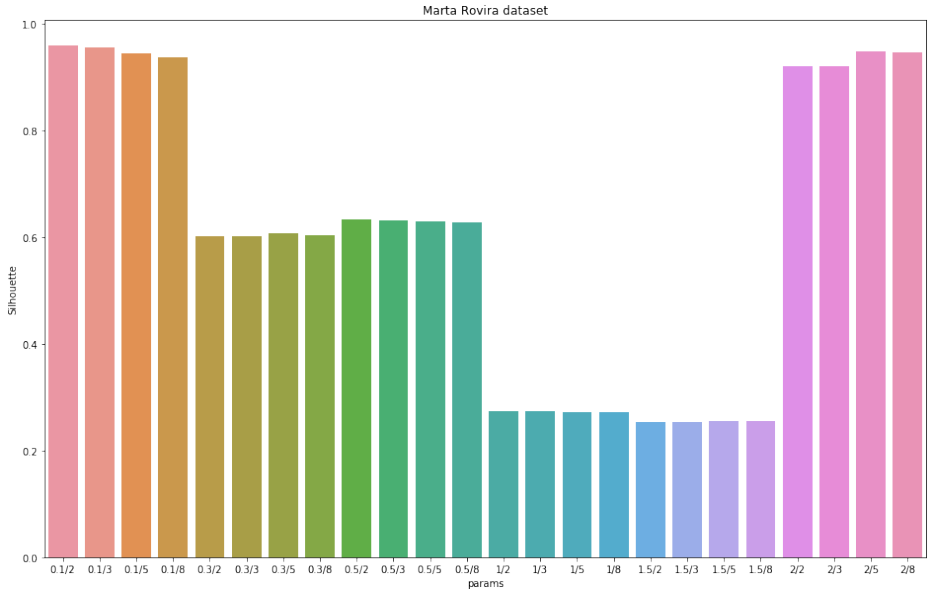

Figure 6: Silhouette coefficient score compared across the different hyperparameters for *Marta_Rovira* dataset.

The next three Figures 7 8 9 are for *Vaga8Nov* dataset.

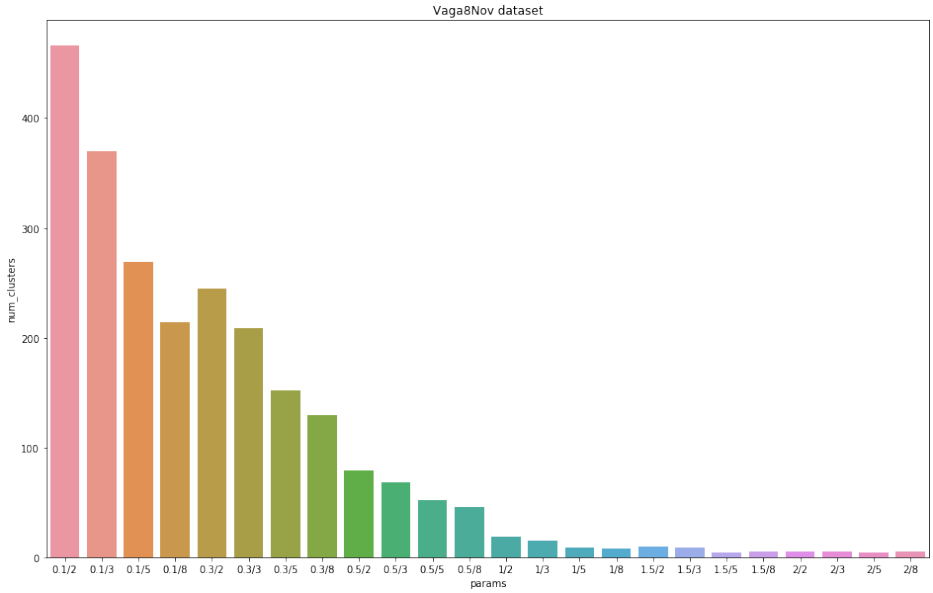

Figure 7: Number of clusters across the different hyperparameters for *Vaga8Nov* dataset.

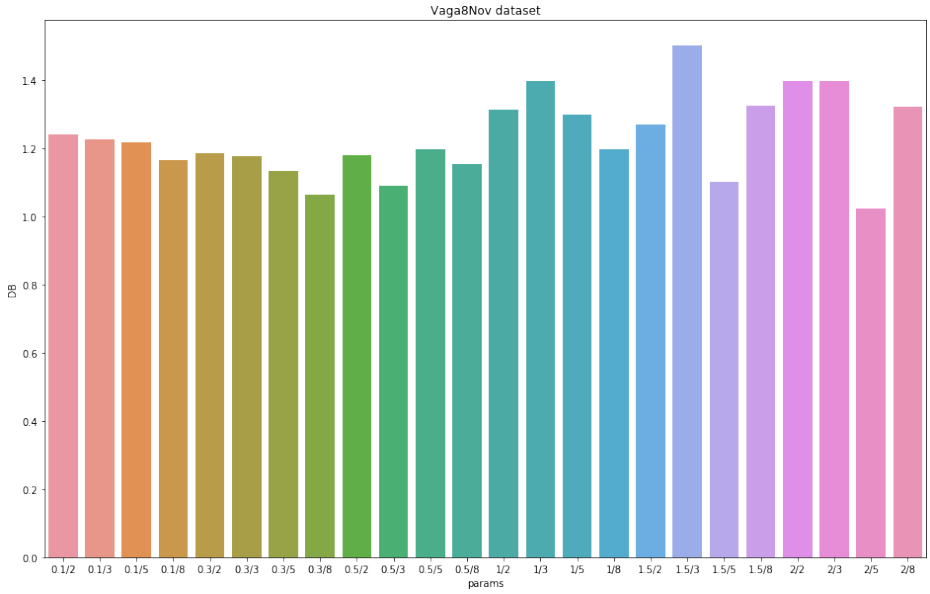

Figure 8: Davies-Bouldin index score compared across the different hyperparameters for *Vaga8Nov* dataset.

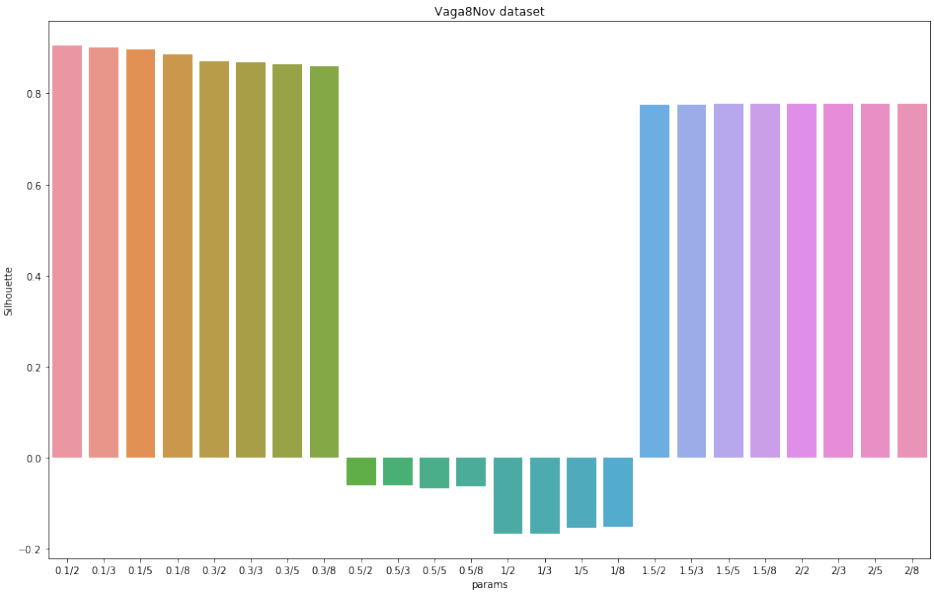

Figure 9: Silhouette coefficient score compared across the different hyperparameters for *Vaga8Nov* dataset.

The last set of Figures 10 11 12 are for *Nochebuena*.

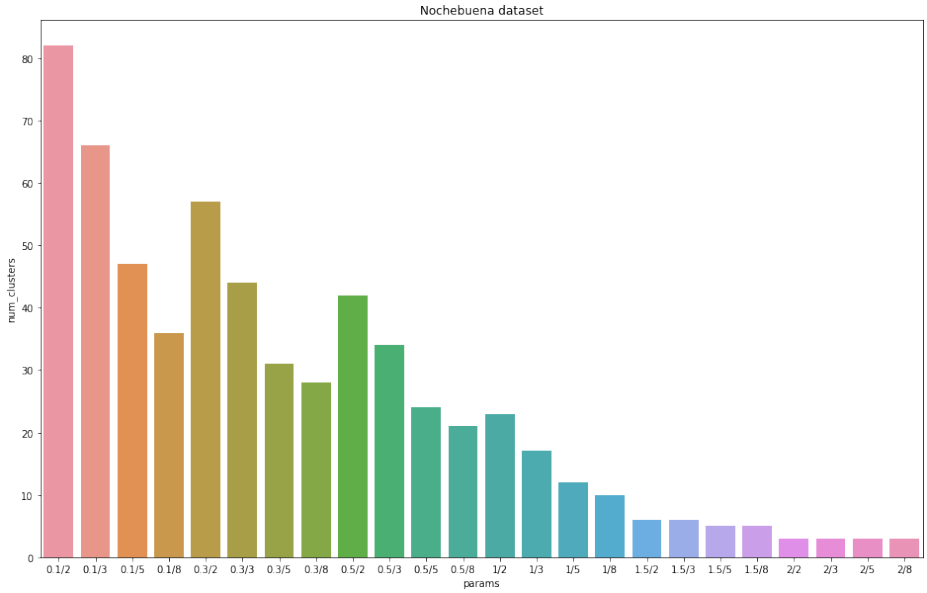

Figure 10: Number of clusters across the different hyperparameters for *Nochebuena* dataset.

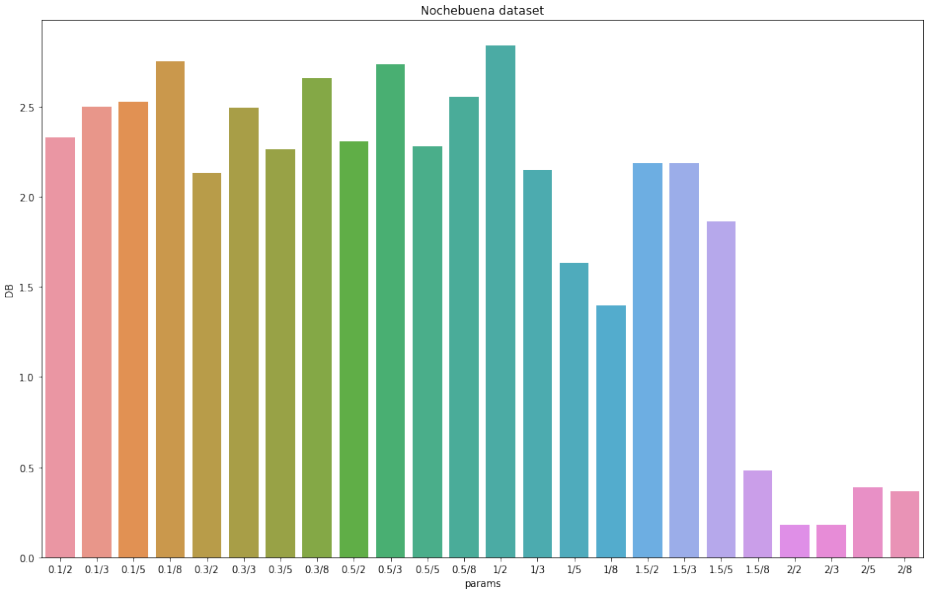

Figure 11: Davies-Bouldin index score compared across the different hyperparameters for *Nochebuena* dataset.

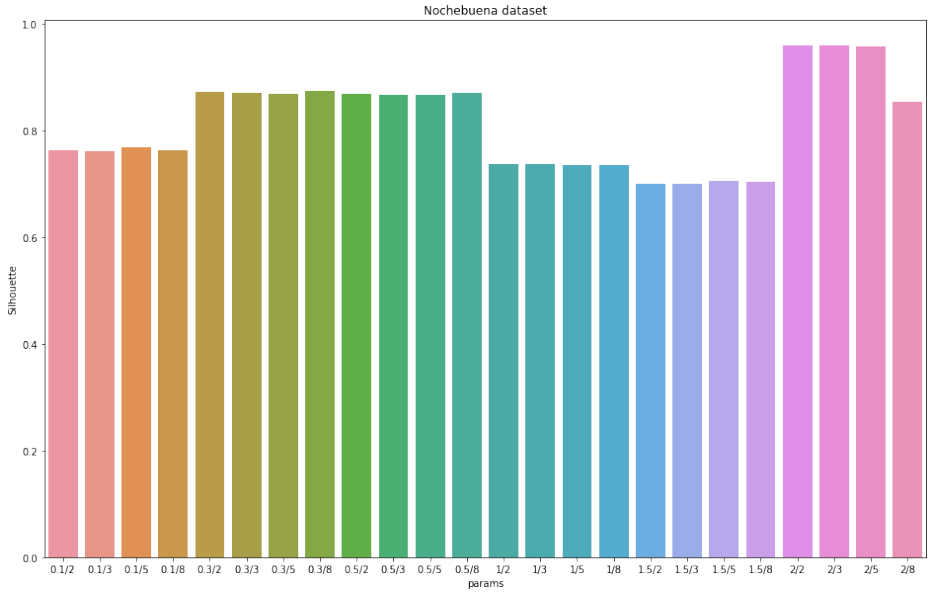

Figure 12: Silhouette coefficient score compared across the different hyperparameters for *Nochebuena* dataset.

2.8 Unsupervised Learning using GCNN Implementation

Deep Learning is usually applied to tasks that involve data in the Euclidean space. Several methods have been developed to process this kind of data, these methods usually achieve good results even when data from Non-Euclidean sources is converted and processed with these methods. In the context of Computer Science, graphs are a very useful and well studied data structure, we can reduce many different problems to graphs to understand them better and use graph theory to get a better intuition about them. Classic Deep Learning methods are not designed to deal with graph data structures; however, recent studies are extending these classic methods to graph structures. In this section we make use of some of these new methods to get some intuition about the underlying nature of our graph data.

After some clustering of the data we proceed to describe the process of implementing a semi supervised model in a neural network with the objective of finding and underlying structure or general results that will allow us to classify networks. For this process we will use **Pytorch Geometric**[4], a library built in top of PyTorch to perform deep learning operations with Graph Data Structures. This library implements some state of the art convolutional algorithms for graphs that we will test in this project. Furthermore, the library also implement pooling and regularization layers among other useful tools for the construction of neural networks.

We use the **.graphml** results obtained in Section 2.5 and read them into a **Data** object

of python geometric. We make sure that relevant values for node features are assigned to each node in the new **Data** structure. These features are the same as in the chapter 2.7.

Then we proceed to implement a neural network model based on Figure 13. The end result might vary because of the nature of our data, but the general process should follow the guidelines established in that model. We perform convolution, followed by pooling, and then convolution followed by some kind of readout process.

The result of our neural network could be a Graph structure, or a vector of features that characterize the Graph. We shall decide which of these is relevant or more appropriate to classify our graph into conflict or non-conflict network.

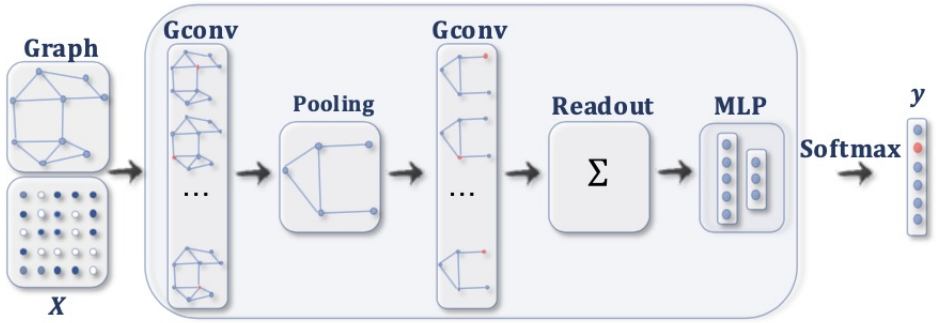

Figure 13: Neural Network for Graph Classification

2.8.1 Exploration of Layer Choices

Since we admit negative for some of our features, we are not using any rectified regularization unit.

We choose the **Graph Convolution Layer** defined by Thomas N. Kipf and Max Welling [7]. The formulation of convolution applied by the author takes a spectral approach by using the Adjacency and the Laplacian Matrix of the graph to compute eigenvectors and perform convolutions on that space. However, this method implies expensive matrix multiplication, so the authors decide to use an efficient approximation that they describe in their paper. Reviewing some Graph Theory, we see that the eigenvectors of the Graph Laplacian matrix can be used to produce a graph layout that captures interesting physic properties, we believe there is an element of this theory that motivated the authors into applying this spectral method in their convolution formulation. Furthermore, other authors seem to have been also motivated to propose spectral methods for convolution.

To be able to apply our neural network to the huge size of twitter data samples we need a way to down-sample our graph while also capturing its main properties. So, a reasonable choice of **pooling layer** is necessary. For this purpose we test two approaches:

SAGPool [8]: In this approach nodes are dropped based on a learnable projection

score. The authors use the concept of **self-attention** that allows this pooling method to consider node feature and also graph topology. Since for our data it is important to consider both of these dimensions, this seems like a natural choice.

TopKPool [9]: Nodes are adaptively selected to form a smaller graph based on a trainable projection vector, and projecting them on 1D; thus, allowing for the application of k-max pooling for node selection.

Then, our neural network will have the following structure:

1. Graph Input
2. Graph Convolution Layer
3. SagPool (where we decrease the number of nodes by half)
4. Graph Convolution Layer
5. TopKPool (where we decrease the number of nodes by half)

Finally, we perform a *manual* average pool over all the nodes, which can be seen as our *Readout* operation. At this point we could be tempted to attempt prediction by introduction a *Multi Layer Perceptron* or any kind of Machine Learning gizmo that could train some weights in order to give us a probability on the type of network or even the level of conflict present in our network. However, here is where the lack of data played a crucial role in the continuation of our project. From *shady* sources we were able to get big data sets where it was possible to confidently identify clusters and get good metrics, but these are only 3 data sets, those mentioned in Section 2.4. We tried to get more data sets but it proved impossible due to API consumption restrictions from Twitter.

The bottom line is that we did not have enough data to train a prediction model, so instead we will explore the *raw* results of our neural network and evaluate if there is any intuition into the correctness of our current app

2.8.2 Results

We sample each of our datasets 100 times independently and then record the results in a different file for each sample. We do not perform any learning at this points, as we are only exploring possible results. Then, we proceed to unify these different results in a single dataframe in *R* and plot the results for each feature.

First we take a look at the results for our three main and more reliable datasets. We can see these results in Figures 14 15 16 17. Across all these figures we see that *Vaga8Nov* clearly stands out, and *Marta_Rovira* and *Nochebuena* stay somewhat close together.

We know that *Vaga8Nov* is the dataset that has more conflict, followed closely by *Marta_Rovira*. However, *Nochebuena* does not present conflict. The clear difference of the results for *Vaga8Nov* seems to indicate that our setting is identifying some underlying

structure for this dataset, but perhaps not the type of structure that we are interested in, because *Marta_Rovira* and *Nochebuena* should not be that close together.

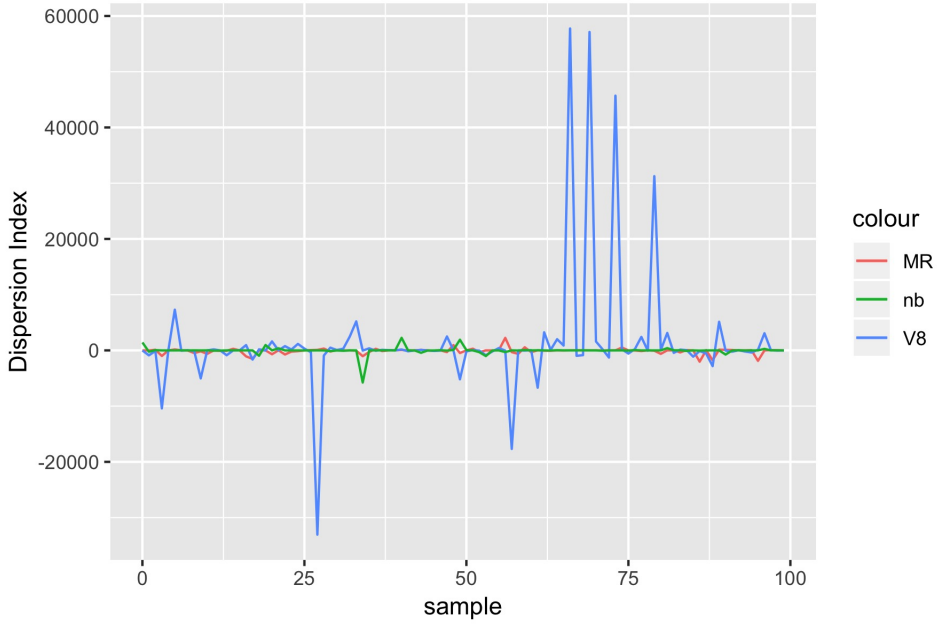

Figure 14: Dispersion Index compared across experiments for our main 3 datasets.

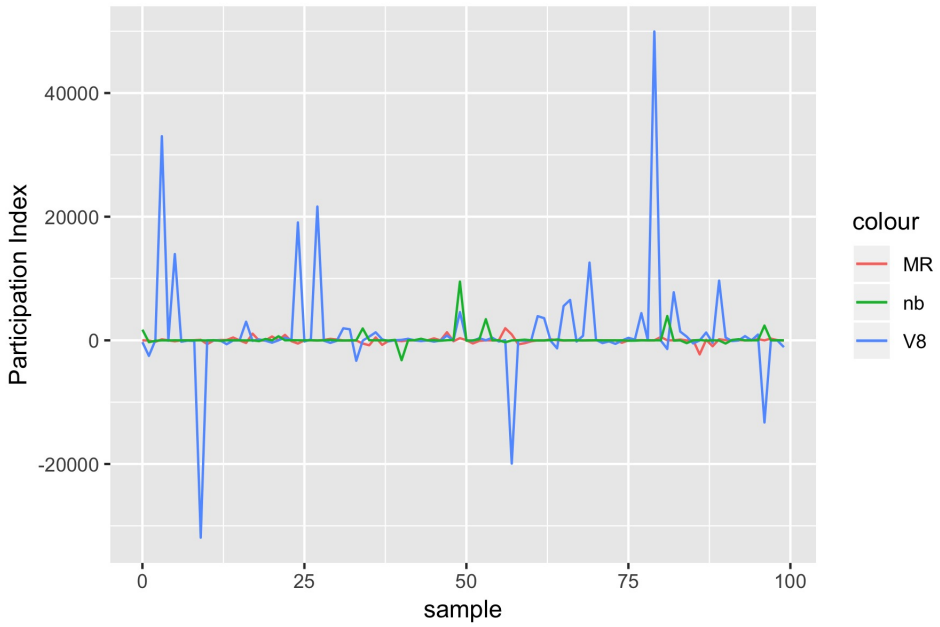

Figure 15: Participation Index compared across experiments for our main 3 datasets.

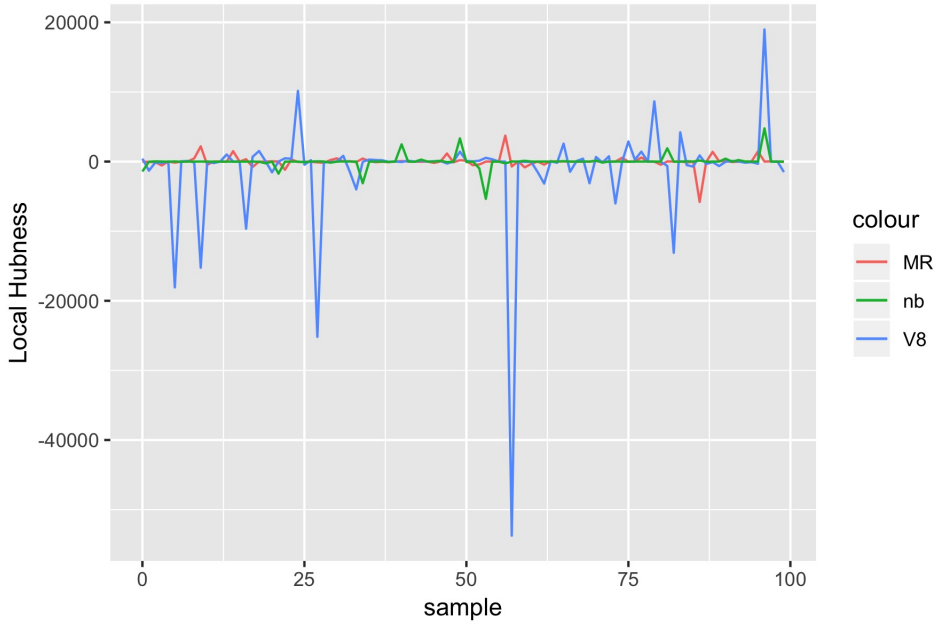

Figure 16: Local Hubness compared across experiments for our main 3 datasets.

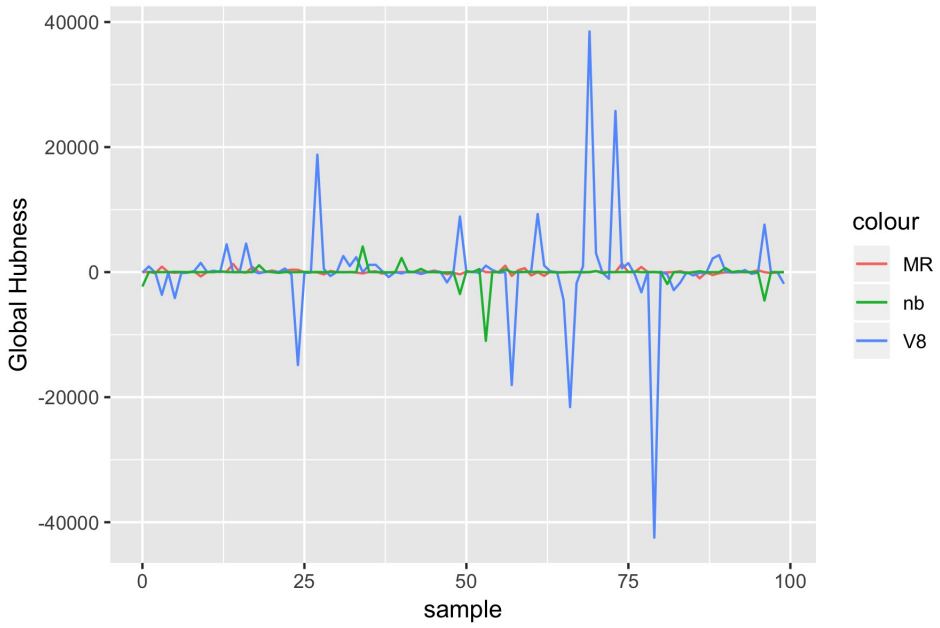

Figure 17: Global Hubness compared across experiments for our main 3 datasets.

The results shown in Figures 18 19 20 21 correspond to the datasets that we tried to mine from the *Twitter* API. We see that *primavera* stands out but we are unable to identify the reason, it was supposed to be a network without conflict given that it is just a regular hashtag with random people, then *sant_jordi* and *quarantine* also show some relevant difference but then again, they are supposed to be just regular hashtags. Anyway, we assume that these are just odd patterns more related with the sparse nature of the graph that represent these small datasets.

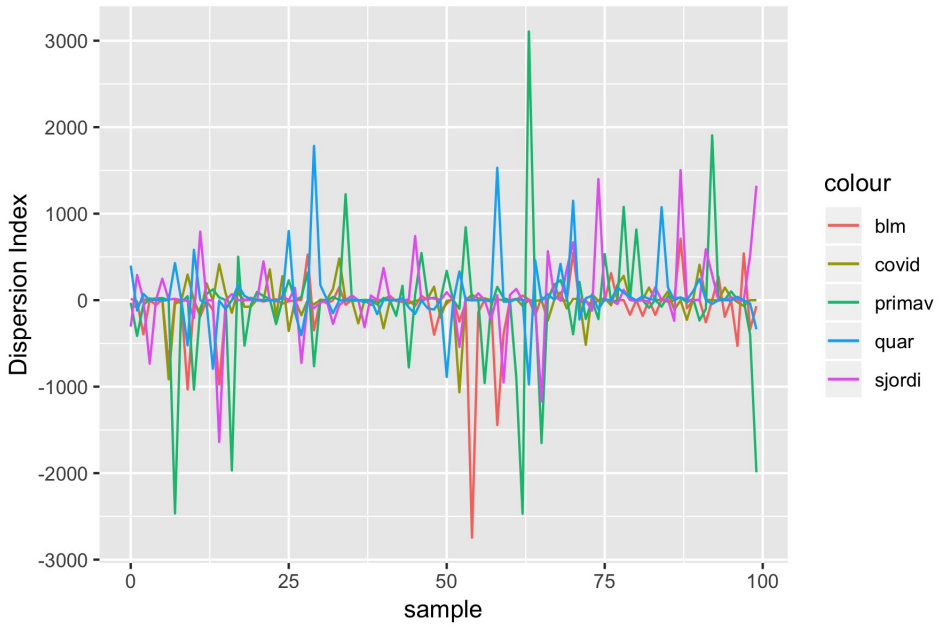

Figure 18: Dispersion Index compared across experiments for our other datasets.

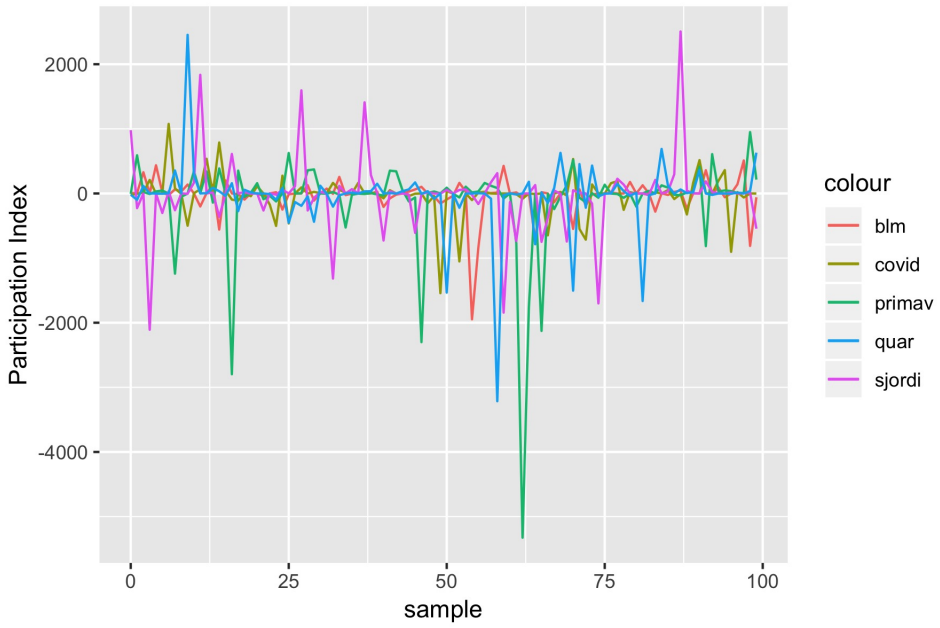

Figure 19: Participation Index compared across experiments for our other datasets.

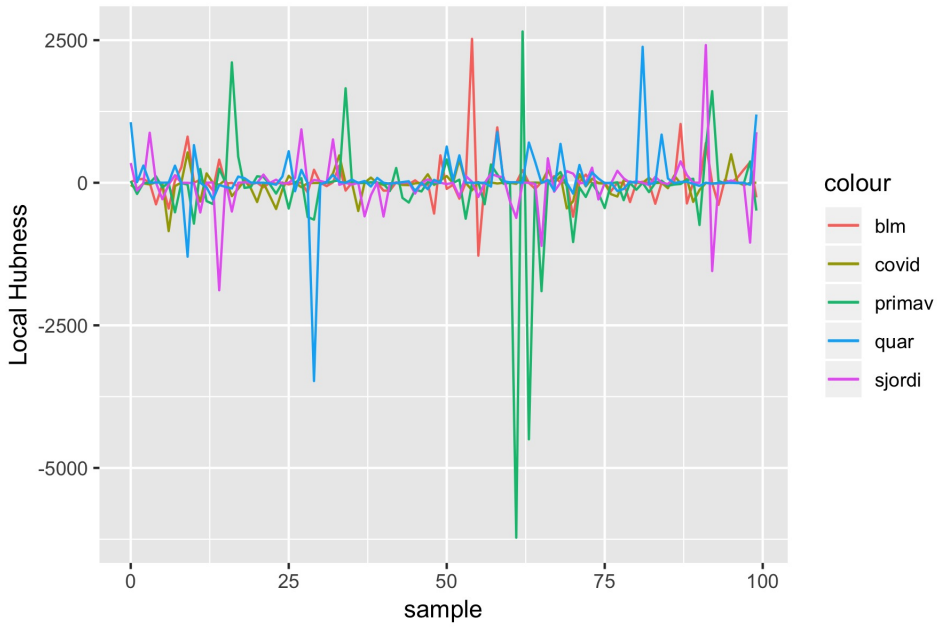

Figure 20: Local Hubness compared across experiments for our other datasets.

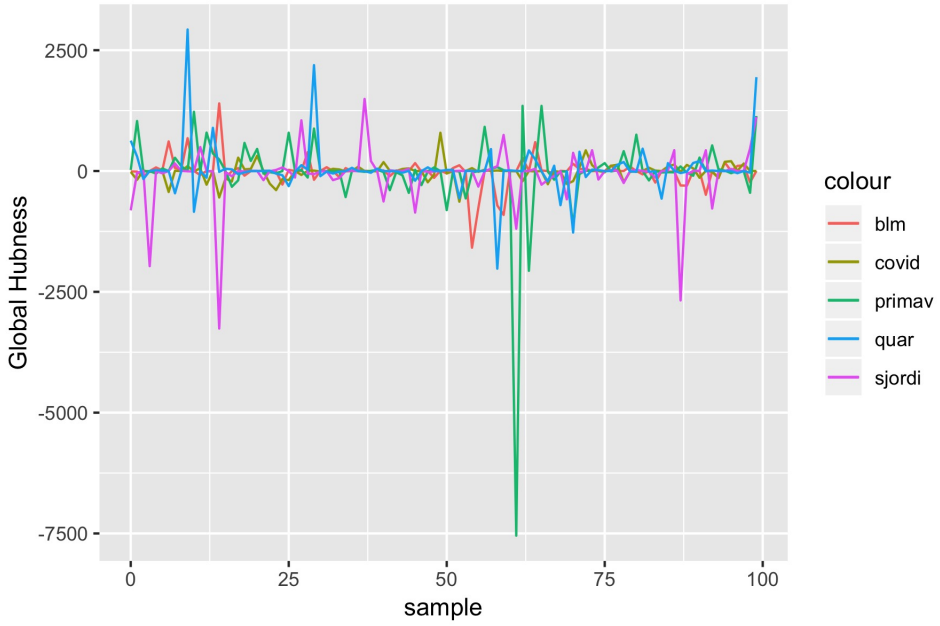

Figure 21: Global Hubness compared across experiments for our other datasets.

2.9 Comments

Although the paper [6] presented a definition of metrics aimed at undirected networks, there is a reference to an experiment performed comparing the results using the in-degree and out-degree of the same network. We used the description of that experiment for our own experimentation. However, it is important to mention that the community finding is always performed using the out-degree.

The fact that the definition of p_i does not include cases where the out or in-degree is 0 is a consequence of these definitions being developed for undirected graphs. We concluded that the best workaround that $p_i = NA$ when the degree is 0, since it seems to be most logical sound assigned value. However, we maintained the original 0 value to see its influence in the results.

The work and experiments performed as part of this project are merely the first step in a much wider and ambitious project that aims at using the metrics presented in this report to divide nodes into roles that best describe the nature of the participation of people in interactions done through *Twitter* during relevant social events. This might result in some changes in the definitions of these metrics or even the introduction of new ones. Furthermore, the exploration work performed in the field of Machine Learning and Deep Learning are part of the same effort that has as objective to find a way to categorize interaction networks effectively.

The implementation of the pre-processing analysis tool was done using C++, this language allows for efficient performance and since most of the calculation of metrics is independent among nodes, there are opportunities for parallelization. The larger experiment consisted of approximately 100k nodes and 450k edges. Processing this amount of information took around 20 min, that included reading the input, processing Infomap, processing Infomap result, metric calculation, and output writing. Then, for the Deep Learning exploration part we used mainly the library *Pytorch* which is a framework that envelopes complex computation using GPUs; although, in our case we only used the configuration for CPUs.

The repository [3] also includes the file "/code/analysis/data_process.Rmd" with some R code to process the output (.graphml) generated by the program. The output format also allows for other network specialized software to easily read the output and process it.

The results presented in Section 2.8.2, specially those about our 3 main datasets, seem to point that our model is capturing patterns that help identify networks where there is conflict. The visual analysis performed by itself shows clear differences that we believe can be enhanced by the actual implementation of the loss function and iterative learning. However, a review of the methods implemented in **pytorch-geometric** shows that we will need to implement our own layers for the *Readout* procedure of the model.

References

[1] `https://mapequation.github.io/infomap/`

[2] M. Rosvall, D. Axelsson, C.T. Bergstrom. *The Map Equation*, Eur. Phys. J. Special Topics 178 13 - 23, 2009.

[3] `https://github.com/wuruchi/commub`

[4] `https://pytorch-geometric.readthedocs.io/en/latest/`

[5] `http://scikit-learn.org/stable/modules/clustering.html`

[6] Florian Klimm, Javier Borge-Holthoefer, Niels Wessel, Jürgen Kurths, Gorka Zamora-López. *Individual node's contribution to the mesoscale of complex networks*, New Journal of Physics 16, 2014.

[7] Thomas N. Kipf, Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*, ICLR 2017.

[8] Junhyun Lee, Inyeop Lee, Jaewoo Kang. *Self-Attention Graph Pooling*, Proceedings of the 36th International Conference on Machine Learning 2019.

[9] Hongyang Gao, Shuiwang Ji. *Graph U-Nets*, Proceedings of the 36th International Conference on Machine Learning 2019.

[10] Maaten, Laurens van der and Hinton, Geoffrey, *Visualizing data using t-SNE*, Journal of machine learning research 2008.

[11] Slobodan Petrovic, *A Comparison Between the Silhouette Index and the Davies-Bouldin Index in Labelling IDS Clusters*, 2006

[12] Ram, Anant and Sunita, Jalal and Jalal, Anand and Manoj, Kumar, *A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases*, International Journal of Computer Applications 2010