Twitch Ego Nets

Jad Sobhie, Sara Nour Dhina

Abstract

This is the report for our project for the Network Machine Learning class (EE-452). In this project, we investigated the binary graph classification of the ego-nets of Twitch. We first analysed our network, extracted some features and did some statistics on the graphs to clearly understand what kind of data we're working with. Then we moved to the classification part. The goal was to predict whether a user plays a single or multiple games. Github repository

1. Exploration

1.1. Dataset

Our dataset consists of an ego-nets of Twitch user. An ego network focuses one one node of the whole network and all the nodes it's connected to and the connections between them. Every node represents a Twitch user and the edges represent the friendships between the users so every user has its own ego-nets graph.

1.2. Graphs Analysis

There are 127'094 undirected connected small-world graphs in this dataset. They have no edge features and no node features. The graphs are binary-labeled (0: plays multiple games, 1: plays a single game). We started our analysis by exploring how many nodes and edges the graphs had.

1.3. Features

First, we decided to look at the two classes of graphs separately and compare some of their features.

Label	min nodes	max nodes	average nodes
0	14	52	31.68
1	14	52	27.94

Table 1: Node features: The minimum and maximum number of nodes is the same for both classes. We notice that the average number of nodes is not so different for both classes.

Label	min edges	max edges	average edges
0	13	775	80.48
1	14	786	91.85

Table 2: Edge features: The minimum, maximum and average number of edges is very close for the two classes.

We notice that the number of average edges is slightly bigger for graphs from class 1. We noticed also that the diameter of the graphs was constant (2) for both classes so we didn't find it relevant to put it here.

Label	min sparsity	max sparsity	average sparsity
0	0.02	0.478	0.08
1	0.02	0.483	0.12

Table 3: Sparsity: Once again, the values are relatively close for the two classes.

Label	min density	max density	average density
0	0.04	0.956	0.167
1	0.04	0.967	0.232

Table 4: Density: the graphs with label 1 seem to be slightly more dense

Looking at these results, we see that there are no features that are significantly different for both classes. However, as expected the average density is a bit higher for the graphs represented by class 1 (players who play a single game). One of the reason that could explain this is the fact that the shared interest of a game tends to create stronger bonds amongst players. The graphs you see in the figure below illustrates the differences in density for both classes.

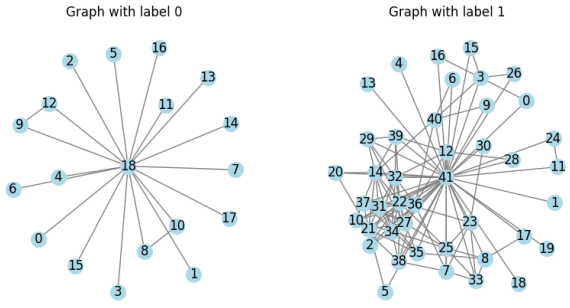

Figure 1: Randomly selected graphs from both classes

1.4. Implications for graph classification

Looking at our graphs analysis, we expect the density and the sparsity to be the most relevant features for the classification task and the diameter to be the least impactful features. However, we decided to train our models using all features also to compare our results with our assumptions.

2. Exploitation

In this section, we compare different approaches for the classification task and compare them to one another.

2.1. SVC classifier

We first trained an SVC classifier with only the graphs features that we presented in the last section: number of nodes, number of edges, sparsity, diameter and density. We only got an f1-score of 50.3 % which is a bad result considering our task, it means that our classifier performs randomly.

Therefore, we decided to add 4 additional features: degrees of the graphs, clustering coefficients of the nodes, betweenness centrality of the nodes and eigenvector centrality for the graphs. We separated our data like this: 60 % for training and 40 % for testing.

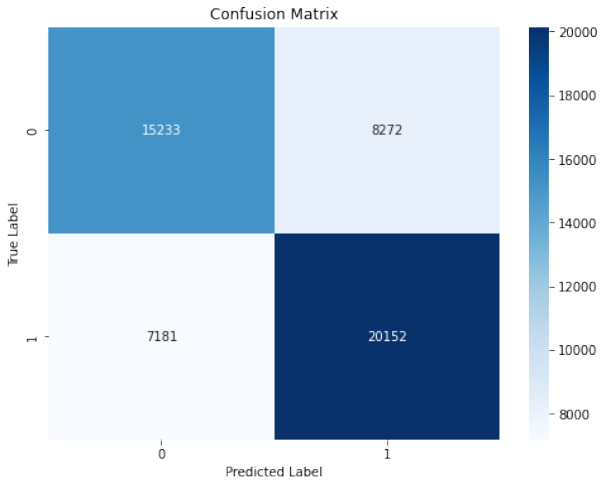

Figure 2: Confusion matrix

You can see our results in Fig. 2 when we use all the 9 features without selection. We achieved an f1-score of 69.54%.
We also wanted to see how much the f1-score would change if we only took the k-best features by using a selector. You can see in the table below the different f1-score values depending on different values of k.

k	f1-score %
1	69.51
2	69.52
3	69.54
4	69.44
5	69.39
6	69.54

Table 5: F_1 scores

By looking at the values we see that the f1-score doesn't really change for different k.

We can see in the figure below the scores of the different features. Features 2 and 4 (which correspond to the sparsity and the density) seem to have the best score but as we've seen the f1-score doesn't seem to be impacted by the features we use. One possible explanation is that we have redundant features, meaning that some of the features provide similar informations and keeping or removing them doesn't have an impact on the classification task. Also, we can see, as expected that feature 3 (which corresponds to the diameter) has the lowest score because this value is the same for all graphs of all classes.

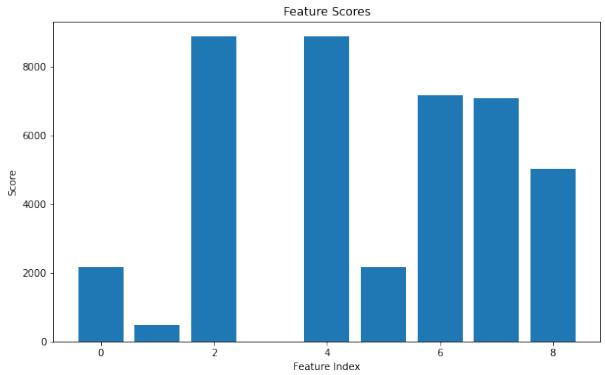

Figure 3: Scores of the different features

Based on our assumptions, we would have expected the f1-score for k=2 to be significantly higher than for other k because the density and sparsity seem to be the most relevant features.

2.2. Node embedding

We tried to train the SVC using node embedding to compare the performances with the hand-crafted features. We used node2vec. Unfortunately due to resources constraints we weren't even able to create the features for all the graphs.

2.3. *MLP classifier*

Multi-layer perceptron is a well-established and widely used algorithm for binary classification problems. Therefore, we tried to train an MLP classifier to compare it with the SVC classifier. We have tried with several combinations of features but again for some reason the performance seems not to be affected by this.

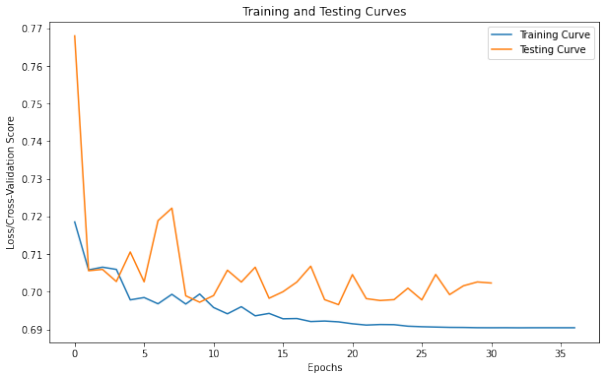

Figure 4: Training and testing errors

We can see on the plot that the error is quite big for both training and testing. Looking at the testing curve we can see that there's overfitting. Our accuracies were the following:

1. Training accuracy: **46.82%**
2. Testing accuracy: **46.53%**

These results are quite bad and it might be because of the redundancy of informations across the features. Since, we've tried with different combinations of features we don't really know how to explain our low accuracies.

2.4. *Neural Network*

The last thing we tried to complete our task was to implement a simple feedforward neural network with two layers. We decided to choose only the density and sparsity as features because again the selection of features doesn't seem to impact the performance.

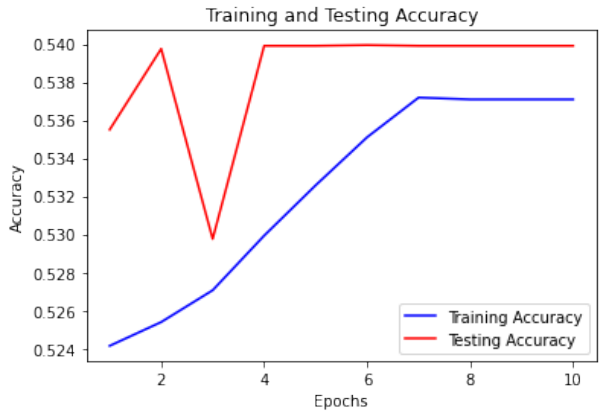

Figure 5: Training and testing accuracy

We see on the plot that the training accuracy goes up with the number of epochs but for testing we have quite a strange behaviour, it seems to get stuck after 4 epochs which might be due to overfitting.

3. Conclusion

We have tried several methods to perform the binary graph classification and the best results we got were with the SVC classifier no matter which combinations of features we used. One thing that we would need to further investigate is why the different combinations of features don't have an impact on the performances even though they have such different scores. This might be due to the fact that the features are too close for the graphs of the different classes and therefore it doesn't help to do the classification. Another thing we could to have better results is to train a graph neural network as it is commonly used for graph classification.