Recommendation Tool for Twitch
Final Project Report

Antoine Dargier
antoine.dargier@student-cs.fr
SDI mention - CentraleSupélec

Johary Ramamonjy
johary.ramamonjy@student-cs.fr
SDI mention - CentraleSupélec

Thomas Douilly
thomas.douilly@student-cs.fr
SDI mention - CentraleSupélec

ACM Reference Format:
Antoine Dargier, Johary Ramamonjy, and Thomas Douilly. 2023. Recommendation Tool for Twitch Final Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

Contents

*

1 ABSTRACT

In this project, we tried to compare different link prediction algorithms in the context of mutual follow in a social media called Twitch. The dataset used contained several data on the links between users as well as some specific info on each account (creation date, number of views, ...). We tested three famous link prediction algorithms using:

(1) Graph Similarity, with classic and more recent features
(2) Skipgram Method with Random Walk
(3) Graph Neural Networks

We focussed mainly on the accuracy and the confusion matrix obtained for each of these algorithms on a testing dataset, made of the initial one, in which random edges have been deleted. After tuning the parameters of each model, we were able to reach an accuracy of 80.9 % in the first case and 79.5% in the second one. We also observed that the features given in the dataset for each node (each user) do not seem to have any interest here, as new features such as Soundarajan- Hopcroft index, which allow us to take them into account, had no real impact compared to more classical features focussing only on edges characteristics. This also led to the fact that our Graph Neural Networks implementation, inspired by a Node Prediction GNN, and so taking node features as well into account, did not succeed to converge (leading therefore to random results). To put it in a nutshell, it seems that in our context, the best results were obtained with classic Graph Similarity methods, using classic features and the recent DICN index.

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 INTRODUCTION & MOTIVATION

2.1 Introduction

. Twitch is a social media dedicated to live video streaming. Launched in 2011, the service has been bought in 2014 by Amazon for nearly 1 billion dollars. It has now more than 140 million active users, among which a little less than 10 million are regularly streaming content. The position of this service, even though much "smaller" in terms of users than its competitors on video (Youtube) or classical social media (Facebook, Twitter), makes it a very interesting environment to study.

Figure 1: Twitch logo

Such as other social media, Twitch adds to the simple content-sharing principle a whole range of tools for users to interact with each other. In particular, the one interesting us in this study is the possibility for a user to "follow" another one. The whole environment around Twitch is indeed mainly organized in communities centred on content creators called "streamers". In each of these communities, people interact with each other and mutual follows are becoming quite frequent.

Studying these clusters have a lot of interests and can be very useful both for content creators (so that they can define more easily the preferences of their followers) as well as for advertisers (which may decide to focus only on some communities according to the promoted product according to their commitment to Twitch).

If we decided to work on Twitch, where the growth of 'communities' is particularly important, this work and its use cases described in the last paragraph are expandable to every other social media.

2.2 Motivation

. We were very motivated by this topic for various reasons. First of all, all three of us are Twitch users, and we wanted to see how we could characterise the relationships between users. Twitch is very much a community, so it's extremely interesting to look at this kind of topic. There are researchers who have modelled these clusters on the French Twitch, which groups communities around streamers and gives very interesting maps [2].

Moreover, we felt a real lack of recommendations on the platform. Indeed, unlike other social networks, no creator or community recommended us on the platform. As the platform continues to grow, it becomes increasingly challenging for users to discover new streamers and form connections with them. This is where link prediction comes in, as it enables us to predict which users are likely to form connections with each other. By leveraging machine learning techniques, we aim to develop an efficient and accurate link prediction model that will help Twitch users discover new content and form meaningful connections on the platform. Ultimately, our motivation is to improve the overall user experience on Twitch and make it easier for users to connect with each other.

Finally, as we said in the introduction, there are also real advertising issues. By targeting communities and users more precisely, the platform can offer and sell much more precise and targeted advertising space.

3 PROBLEM DEFINITION

. With millions of users and thousands of streamers, it can be challenging to discover new content that aligns with the user's preferences. The project aims to address this challenge by developing a **recommendation tool** that can suggest relevant content to users based on their interests and past behaviour

The recommendation tool will rely on **link prediction** techniques to identify possible mutual follows between users. The approach will involve analyzing the connections between users, identifying patterns in their behaviour, and predicting possible future connections. By predicting possible mutual follows, the recommendation tool can suggest relevant content to users based on their interests and past behaviour.

Currently, Twitch does not offer a recommendation tool, which makes it challenging for users to discover new content on the platform. The project aims to fill this gap by developing a recommendation tool that leverages link prediction techniques to suggest relevant content to users. The tool has the potential to enhance user engagement on the platform and make it easier for users to discover new and exciting content.

4 RELATED WORK

. We will try in this project to use different methods and compare their performances and efficiency. Therefore, the first step of our project is to make a quick review of the existing methods used in published papers.

We found that plenty of methods exist in order to achieve link prediction between nodes of a graph. In the following paragraph, we formulate a brief history of them. More technical details will be given in the following section.

First, we can note the Graph Similarity method, published in 2011 in [8], which is one of the well-known techniques used in graph analysis and could totally fit with our objective in this project. Then, later on, in 2018 and 2019, a lot of new methods surged with the quick development of deep learning. In particular, we read about the use of features built with deepwalk and skipgram algorithms in [1], or also the development of a specific deep-learning network with the Graph Neural Network [11], a new architecture of model, entirely built in order to deal with graph data and seems to be a very interesting and useful tool that we want to test in the case of link prediction in our project.

5 METHODOLOGY

5.1 Dataset

. To achieve these comparisons, we will use a public dataset based on real-life data to confront these theoretical models with practical data. In this project, the one we are going to use to feed and test our model can be found in the Git repository [6] and is linked to a prior work made by the authors [7]. It contains a list of more than 150K nodes and about 7 million edges that represent a Twitch user and a mutual following relationship respectively. The edges are therefore undirected, as we only consider mutual relationships between users.

Each of the nodes, and therefore each user, is defined by a certain number of features, namely :

- An integer identifier
- The language used by the user
- Some data about the characteristics of the account (if the user is mature, affiliated with Twitch, if the account is not used anymore, ...)
- The number of views of the account
- The dates of account creation and last connection to it.

According to the author of this Git Reposit, this graph has a 0,0005 - density and a 0,0184 - transitivity. Prior to any work on this project, we will start by checking if these assumptions are true or not.

The whole dataset was very huge, so it was too difficult for us (and our computing devices) to train our model on the whole of it. Therefore, we decided to truncate it, so that to keep a smaller but still representative dataset. Therefore, we decided to limit ourselves to Dutch-speaking users of the platform Twitch. This subset is still made of :

- 674 nodes standing for the 674 Dutch-speaking Twitch users
- 226801 links: 4459 mutual relationships and 222342 non-relationships

Another step will consist in checking if the data is clean or if it needs some pre-processing before being used, in order to not bias the results we will obtain by the end of this project.

5.2 Data preparation

. In this project, we will seek to predict links between users, what we call mutual follows. To do this, from the dataset, we will follow the following steps:

- Overview of the dataset (see the previous subsection)
- Random deletion of some links and separation of the whole dataset into train and test sets
- Implementation of the prediction algorithms
- Analysis of the results and comparisons (see the next section)

Deleting some links and splitting the data. In order to train and test our model implementations, we will need to create, from the raw data, two different sets of edges and non-edges, the first one being used to train all the models built and the second one will be used to compare the predicted results with the ground truth and therefore compute metrics to evaluate the relevance of their use in the context of our project.

To do so, we stored in two lists all the edges (as a tuple of nodes) and the non-edges, before shuffling both of these lists and finally splitting each of them in two with a 0.8/0.2 rate. The first split of both lists will be mixed to form the training data, whereas the second slit of both lists will form the testing data.

Implementation of the prediction algorithms. To achieve the prediction task of this project, we have chosen to implement three different methods along this project. The final goal of it being to compare each of these algorithms to each other. These three methods are explained in the following subsections :

5.3 Graph similarity method

. The first method will use **graph similarity** [8], which is a widely used technique in graph analysis. Graph similarity involves comparing the properties of nodes to identify similarities and differences. The approach will leverage the properties of nodes and edges in the graph to identify patterns and make link predictions. In particular, we will combine usual graph similarities coefficients, such as degree centrality, betweenness centrality, Jaccard coefficient, preferential attachment and Adamic Adar index, with more recent ones such as Soundarajan-Hopcroft index [9] or Direct-Indirect Common Neighbours (DICN)[8]. In particular, we are going to consider three scenarii which will be compared one to each other:

1. In the first case, the edges' features will be composed of 6 classical features, namely: degree centrality (for both extremity nodes), betweenness centrality, Jaccard coefficient, preferential attachment and Adamic Adar index
2. In the second case, we will only consider Soundarajan-Hopcroft index, based on two possibly interesting node feature: the "maturity" of the channel, which allows separating recent accounts to older ones and if the channel is "affiliate" to Twitch.
3. In the last case, we will only consider DICN.

These similarities (or least similarity) are then computed for each couple of nodes of the training and testing datasets. A classical Extreme Gradient Boosting Classifier (XGBoost) is then fed with the features of the training dataset as well as with the corresponding class of each couple of nodes (edge or non-edge). Finally, after this training step, the XGBoost is able to make guesses on the class of tuples of nodes of the testing dataset.

Soundarajan-Hopcroft index. The Soundarajan-Hopcroft index is a measure introduced by S. Soundarajan and J. Hopcroft in 2012 in [9]. It is based on the classical common neighbours' metrics defined for two nodes i and j by :

$$d_{ij} = |\Gamma_i \cap \Gamma_j|$$

, on which a new term is added, giving the final coefficient defined by :

$$SH_{ij} = d_{ij} + \sum_{\omega \in \Gamma_i \cap \Gamma_j} f(\omega)$$

where f is an indicator function, which returns 1 if ω belongs to the same community as i and j and 0 otherwise.

Direct-Indirect Common Neighbours coefficient. The Direct-Indirect Common Neighbours coefficient or DICN is a coefficient introduced by Ahmad Zareie Rizos Sakellariou in [8]. Based on more 'classical' indexes (node degree and common neighbours indexes), it proposed to introduce the notion of second-order neighbour in the calculation.

Basically, the coefficient between two nodes i and j is computed by the following formula :

$$DICN_{ij} = (1 + CN_{ij})(1 + Corr_{ij})$$

where CN_{ij} is the Common Neighbours coefficient between nodes i and j and $Corr_{ij}$ is the Pearson correlation coefficient of two vectors N_i and N_j on the $UN_{ij} = \{z|(N_i[z] > 0)||(N_j[z] > 0)\}$ set.
For a given node i, the $N_i[z]$ coefficient (for z in the set of node indices) is defined by :

$$N_i[z] = \begin{cases} \mathrm{d}_i & \text{if z=i} \\ \mathrm{CN}_{iz} + 1 & \text{if } \mathrm{v}_z \in \Gamma_i \\ \mathrm{CN}_{iz} & \text{if } \mathrm{v}_z \in \Gamma_i^{(2)} \\ 0 & \text{otherwise} \end{cases}$$

where d_i is the degree of node i, Γ_i the set of neighbours of node i and $\Gamma_i(2)$ the set of second-order neighbours of node i.

5.4 Deepwalk method

. The second method will use **deepwalk/skipgram-based features**, which is a graph embedding technique that aims to learn a low-dimensional representation of a graph.

DeepWalk is a method for learning latent representations of nodes in a graph. The idea behind DeepWalk is to learn these representations by generating sequences of nodes based on random walks through the graph and then applying a word embedding technique such as skip-gram to these sequences. This method comes from NLP methods, where we create embeddings according to the context to have a more precise vectorization of words.

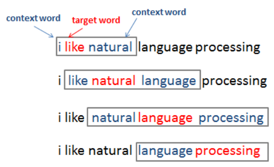

context word
target word
context word
i like natural language processing
i like natural language processing
i like natural language processing
i like natural language processing

Figure 2: Word2Vec Skipgram Method for 3-grams

In the skip-gram method, the goal is to learn the embeddings of words in a way that words that appear in similar contexts have similar embeddings. Similarly, in the case of DeepWalk, the goal is to learn embeddings of nodes in such a way that nodes that are more likely to occur in the same random walk sequences have similar embeddings.

To achieve this, DeepWalk generates random walk sequences starting from each node in the graph. These random walks are generated by performing a fixed number of steps, where at each step, the next node is chosen randomly from the neighbours of the current node. These sequences of nodes are then treated as "sentences" and a word embedding technique such as skip-gram is used to learn the embeddings of nodes.

The skip-gram method maximises the probability of predicting the context nodes given a central node. The central node is represented by its embedding, and the context nodes are represented as one-hot vectors. The probability of predicting the context nodes is then calculated using the softmax function.

By applying the skip-gram method to the sequences of nodes generated by random walks, DeepWalk learns latent representations of nodes that capture the graph's structure. These representations can then be used for various downstream tasks such as node classification, link prediction, and graph visualization.

Here we will clearly list the different steps built for this kind of model:

- data preparation: creation of the graph and sets of training and test with edges and non-edges
- creation of a function to generate random walks from the root and of length L
- creation of the deepwalk method: for each node, N times, generate the random walks of length L
- creation of the embeddings with Word2Vec library with the walks, specifying the window size ws
- features: from each set, creation of the embeddings for each edge with the previous functions
- model: training of a LogisticRegression model to classify them
- fine-tuning: try different values of L (length of walks), N(number of walks) and ws (window size) to find the optimized parameters

This method works quite well but has some limitations. First of all, it only relies on the structure of the graph and does not take into account the features of the nodes. In our study, we had features that we could have used. However, they were not very relevant for link predictions (number of viewers or creation date). With more relevant features, it would have been necessary to take them into consideration in our study. Depending on the size of the selected walks, this method may be very local, and therefore not capture global graph information that could be relevant. Having global information about the density of the graph or the presence of large attractor nodes could be very useful to improve performance. Moreover, this method is difficult to implement in large networks. For our study, we had to limit the analysis to the Netherlands, which does not have too many nodes (about 200,000 nodes). To fine-tune the parameters, this model already took a lot of time, sometimes with a few hours of training. It, therefore, seems unthinkable to use this type of method for larger networks. Finally, if new nodes are added, the whole process has to be started again: we cannot reuse the previous model. This poses many problems because we know that these types of networks are "alive", so they are constantly changing between new users, old users, and new and deleted links.

5.5 GNN method

. Finally, the project will focus on a deep learning method that uses **graph neural networks (GNNs)**. GNNs are a type of neural network that is specifically designed to work with graph data structures. They can be used to model complex relationships in the graph, learn representations of the nodes and edges, and make predictions based on the learned features.

By implementing and comparing these three distinct methods, the project aims to determine the most effective approach for making predictions on graph data. The project will provide insights into each method's strengths and weaknesses, and help identify the best technique for different types of graph data.

The SAGE-GNN implementation we are going to use in our project is based on a pre-existing Graph Neural Network implementation dedicated to Node Prediction and inspires itself from the works of M.Zhang [10] [3] and the following webpages citing the SAGE: [4] and [5].

Our model will be therefore built this way :

- A first GCN Convolution Layer with an output size set to 16
- A ReLU layer
- A Dropout layer
- A second GCN Convolution Layer with an output size set to 16
- A step where the feature vectors of each target node is multiplied by the feature vectors of each source node
- A concatenation Layer
- A sigmoid Layer

This construction allows us to take into account the node features and only them, contrary to all the implementations done so far which mainly used coefficients based on relationships between nodes. The results of it will therefore be based - only - on the relevance of those features in our study case. If they are not relevant, chances are high that this method will not be relevant either.

In order to train our model, we used the Binary Cross-Entropy Loss, which is particularly relevant here as the goal of our CNN is to classify each set of nodes between "Edge" and "Non-Edge" classes.

6 EVALUATION

6.1 Metrics

. To compare the performance of these algorithms, we will use classical classification metrics. We will be able to establish confusion matrices and compare the accuracy of the methods used. Here, it does not seem necessary to avoid false positives or negatives in particular, since there are no particular issues in the recommendations.

One problem to be aware of is that in addition to the removed links, the models we are going to test all across our work may be able to predict new links on the existing base, and this should not reduce their performance.

Finally, to have a complete comparison between all the methods we plan to use during this project, a comparison of the computation time of each one may also be made in order to see in a more accurate way whether one of the methods is way better than the two others or not.

6.2 Results

In this part, we will expose the different results we obtained throughout this project, according to the method used.

6.2.1 Graph similarity. First of all, we implemented the Graph Similarity model described in the previous sections. Before making any experiments on it, we will need to finetune the parameters of our XGBoost classifier. In particular, two main parameters could have an impact on the final results: the number of estimators used and the learning rate.

In the following Figure 3, we computed the accuracy value on the testing set according to the value of these two parameters :

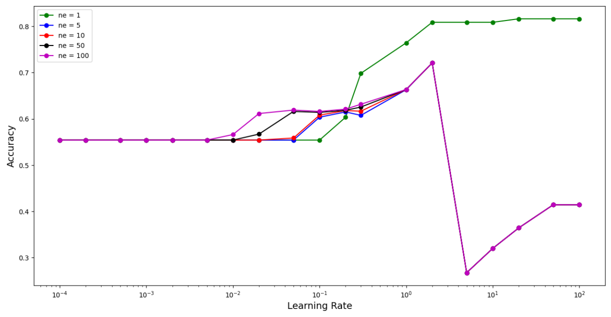


Figure 3: Fine-tuning of learning rate and number of estimators for the Graph Similarity method

As we can see on it, increasing the number of estimators seems to lead to a decrease in the overall accuracy of the model. On the contrary, the learning rate giving the best results seems to be 2. In the following experiments, we will keep these two values to increase the chances to get better results.

Classical similarity coefficients. As a reminder, we wanted first to assign to each couple of nodes a feature under the form of a vector, in which each coordinate is one of the following classical similarity index: degree centrality (for both extremity nodes), betweenness centrality, Jaccard coefficient, preferential attachment and Adamic Adar index. After classification with XGBoost, we obtain the following results summed up as a confusion matrix in Figure 4

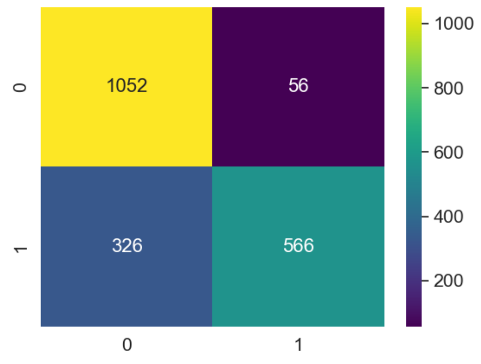


Figure 4: Confusion Matrix of the Graph Similarity Method when using a bunch of classical coefficients

As we can see on this confusion matrix, the results are already pretty good, as the total accuracy is 80,9%. Moreover,

We will now explore new features to see if they improve or not this result.

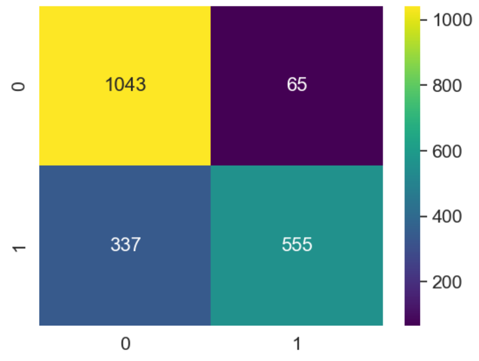


Figure 5: Confusion Matrix of the Graph Similarity Method when using Soundarajan- Hopcroft index associated with the 'Affiliate' tag

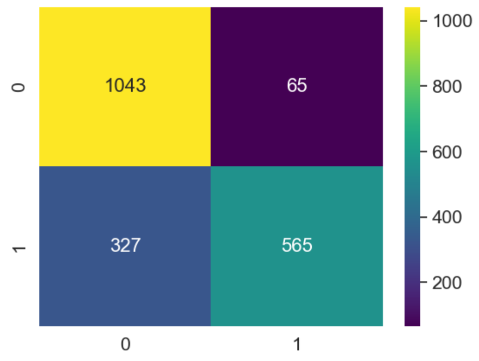


Figure 6: Confusion Matrix of the Graph Similarity Method when using Soundarajan- Hopcroft index associated with the 'Mature' tag

Soundarajan-Hopcroft index. The results obtained with the Soundarajan-Hopcroft metrix is summed up in the following Figures 5 :

As we can see on it, the results are really similar to those obtained in the previous case with classical coefficient, with accuracy reaching respectively 79.9% and 80.4%. The proximity to the classical coefficients is here particularly notable and will be discussed in the following subsection.

Direct-Indirect Common Neighbours [8]. Finally, we also used the Direct-Indirect Common Neighbours (DICN) as our last feature used to characterize each set of nodes. The final confusion matrix is in Figure 7.

This last feature gets results that are of the same order of magnitude as the previous ones with an overall accuracy on training test set to 80,6%. What is admittedly notable is the number of True Positives which is slightly higher (580) than in the previous case. Therefore, it seems that this index seems to have the best results so far.

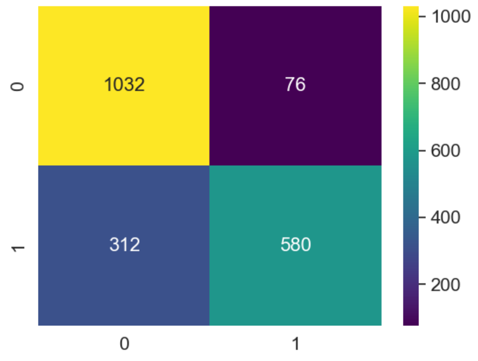
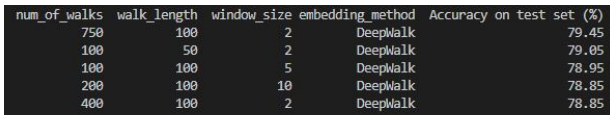


Figure 7: Confusion Matrix of the Graph Similarity Method when using DICN index

Moreover, as a last experiment, we also tried to gather all the previous indexes on a single vector, to see the behaviour of the XGBoost Classifier. The results obtained this way are identical to those obtained while using only the DICN index in Figure 7.

6.2.2 Skipgram-based Features. As we have seen in the presentation of the model, we have fine-tuned the model to have the best possible performance. We played with three parameters: the length of the walks, the number of walks, and the window size. Here is what our results show for the five best models:



```
num_of_walks  walk_length  window_size embedding_method  Accuracy on test set (%)
         750          100            2         DeepWalk                     79.45
         100           50            2         DeepWalk                     79.05
         100          100            5         DeepWalk                     78.95
         200          100           10         DeepWalk                     78.85
         400          100            2         DeepWalk                     78.85
```

Figure 8: Model Fine-tuning

Then, for the best model, we obtained the following confusion matrix:

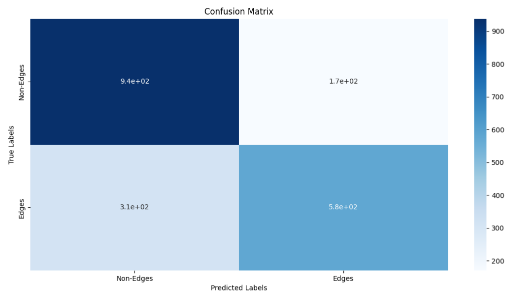


Figure 9: Confusion Matrix of best model

We, therefore, obtain very good results with an accuracy of around 79.5%, despite all the limitations of this model that we have seen previously.

Model	Accuracy	
Graph Similarity method	Classical	80.9%
	Soundarajan- Hopcroft	79.9 − 80.4%
	DICN	80.6%
Deepwalk	with Skipgram	79.5%
GNN		−%

Table 1: Accuracy Results on Training set obtained for each method tested in this Project

6.2.3 Graph Neural Networks (GNN). The Graph Neural Network implementation we managed to code was not satisfactory in our case. Indeed, on the training dataset, it was not able to converge as we can see on Figure 10.

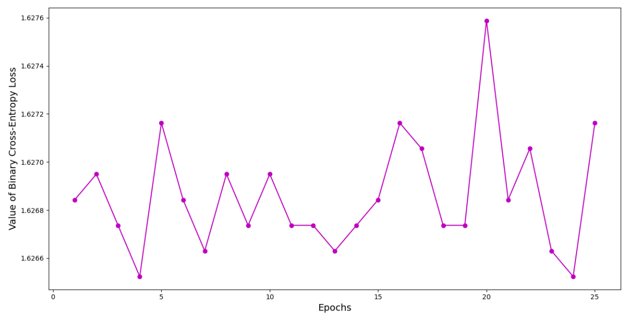


Figure 10: Evolution of the loss of the GNN during the training process

Even though this result is quite disappointing, as GNNs are often said to be top-of-the-art techniques in graph manipulation, it is still full of interesting lessons, that will be discussed later.

6.3 Discussion

Here is a table to recap all our results:

Graph Similarity. During the experiments we led with the Graph Similarity method, we observed astonishing, but interesting behaviours that seem important to discuss.

First of all, during the parameters tuning process, we observed that, in our case, increasing the number of estimators used by the XGBoost classifier had worsened the accuracy obtained on the testing dataset. Two hypotheses came to our mind while facing this ;

- Whether the aggregation process of the Classifier between all estimators is not adapted to our problem, leading the classifier to mix "good" predictions, with "worse" ones, resulting in a decrease of the accuracy on the testing dataset.
- Or, the model could be so unstable (potentially due to the small number of features used each time) that according to the estimator used, the prediction becomes random. However, due to the fact that the overall accuracy remains good in all the cases we tested, this hypothesis may not be favoured.

The use of only one estimator allowed us to reach a really high computation speed, allowing us to build features and achieve the classification in around 120 seconds (10 minutes for the DICN index, for which there exists no *networkx* dedicated function and was coded from scratch).

Then, on the overall accuracy obtained in each experiment on the testing dataset, we observe that (on 2000 items), the order of magnitude never changes, being always around 80%, which is very good. Moreover, in all cases, the model tends to predict well when an edge does not exist: the number of False Positives being between 13 and 18 times lower than the number of True Negatives. That indicates that the model tends to favour a "Negative" classification in contested cases. That behaviour is not so surprising as, during the feature-building process, in all nodes where a failure occurred (for example because the node does not have any edge linked after the training/testing dataset split), we decided to assign the edge a null feature vector by default.

Finally, the results obtained for each kind of feature used are also very rewarding. Indeed, two separate coefficient types have got good results: first, the "classical" coefficients, which maximized the overall accuracy on the testing dataset, with an 80.9% peak. On the other hand, the DICN index, which still has a satisfying 80.6% accuracy, also seems to behave better in predicting existing edges: the True Positive against False Negatives ratio reaches here 1.86, while it was only at 1.73 with classic coefficients.

However, the fact that all accuracy values remain at the same high also indicates that the added value of Soundarajan-Hopcroft and DICN methods are not so high in our study case.

- The Soundarajan-Hopcroft index allowed us to take into account two node features: "maturity" and "affiliation". The lack of different behaviour compared to classical coefficients proves to us that these characteristics are not really important to determine whether two users are likely to follow one other.
- The DICN allowed to take into account the second-order neighbourhood of each node. This time, the lack of difference could indicate that in the Twitch environment (compared to other social media such as Facebook or Twitter), friends of friends are not particularly encouraged to follow one other.

Skipgram-based Features. As we have seen in the last subsection, Skipgram-methods allowed us to reach pretty good results, with a very good accuracy rate on the testing set at 79.5%, which is similar to the one obtained with the Graph Similarity method.

What is nevertheless notable here is the breakdown of predictions in the Confusion Matrix 9. Contrary to Graph Similarity, the number of True Positive is quite high (580), equivalent to what was obtained with the DICN index (Best performance with graph similarity), but the number of True Negatives is a bit lower than expected (940), while it was around 1030 previously. This can probably be explained by the random walk process and more balanced paths between edges and non-edges.

Graph Neural Networks. As shown in Figure 10, we did not manage to make our Neural Network converge. Some reasons can be evoked for this failure that could be pretty interesting in our case :

- First of all, our GNN implementation contains a step which is not a built-in function of a specific package. Therefore, the computation time is really high (more than 10 minutes for a single epoch). This could also have prevented the gradient to back-propagate throughout the model in a satisfactory way.
- Then, as explained in the Graph Similarity part, the node features may not be very relevant to guess whether two users are related or not. As our GNN implementation fully relies on these, this could explain why it does not succeed to categorize each couple of nodes according to these data only.
- Finally, maybe the node features could be sufficient to make predictions, but it would have needed specific pre-training we did not succeed to identify.

With all these hypotheses and remarks, we came to the conclusion that Graph Neural Networks were maybe not the best algorithm to use in our Twitch Edge Prediction task.

7 CONCLUSION

. To put it in a nutshell, the project we are going to lead along the MLNS course consisted in implementing three different algorithms dedicated to link prediction: Graph similarity with different kinds of features (older ones and more recent ones), deepwalk/skipgram methods and Graph Neural Networks (GNN). These methods have been applied in a real-life case of link prediction on the social media Twitch based on a publically available dataset depicting mutual "follow" relationships between a subset of Twitch users during the year 2018.

While we were able to build satisfactory models of Graph Similarity and Deepwalk methods, the GNN implementation did not work on our data, probably because of its composition.

During the study, we mainly focussed on the study of accuracy based on a testing dataset as well as the study of the different components of the confusion matrices obtained. We found very satisfactory results for Graph Similarity methods, in particular while using classical features (Degree centrality for both extremity nodes, Betweenness centrality, Jaccard coefficient, Preferential attachment and Adamic Adar index) or a more recent one called Direct-Indirect Common Neighbours, which succeed in maximizing both the overall accuracy (80%) and the number of True Positive (580). Similar results were also obtained with Deepwalk method, with an overall accuracy just behind it.

All these methods only used graph characteristics without focusing on node features that were present in the Dataset. We also used the Soundarajan-Hopcroft index on these node features to see if they could have an impact on the final prediction, but the results were not significant enough to see any impact. In our Twitch use case, it seems that there is no clear relationship between the proximity in terms of the number of views, maturity, ... and the probability to follow each other.

Therefore, contrary to other social media (such as Facebook), where high importance is given to putting people with the same characteristics together, it is less the case in Twitch (at least with the characteristics of the dataset). We would therefore advise classical methods when dealing with Twitch, such as Graph Similarity with Classical similarity coefficients or DICN (which is specifically designed for these kinds of use cases), which are also quicker, than more complex and advanced methods such as GNN or Skipgram-based features.

REFERENCES

[1] Pedro Almagro-Blanco and Fernando Sancho-Caparrini. 2019. Improving skip-gram based graph embeddings via centrality-weighted sampling. (2019). DOI: 10.48550/ARXIV.1907.08793.

[2] Nicolas Bouchaib. [n. d.] French Twitch Cartography. https://first-link.fr/cartographie-de-donnees-de-twitch-france-en-juin-2022/. [Online; accessed 21-04-2023]. ().

[3] [n. d.] Chapter 10 Graph Neural Networks: Link Prediction. https://graph-neural-networks.github.io/static/file/chapter10.pdf. [Online; accessed 04-22-2023]. ().

[4] [n. d.] Graph Neural Networks: Link Prediction (Part II). https://blog.dataiku.com/graph-neural-networks-link-prediction-part-two. [Online; accessed 04-22-2023]. ().

[5] [n. d.] Link Prediction using Graph Neural Networks. https://docs.dgl.ai/en/0.8.x/tutorials/blitz/4_link_predict.html. [Online; accessed 04-22-2023]. ().

[6] Benedek Rozemberczki. [n. d.] Twitch Gamers. https://github.com/benedekrozemberczki/datasets#twitch-gamers. [Online; accessed 03-03-2023]. ().

[7] Benedek Rozemberczki and Rik Sarkar. 2021. Twitch gamers: a dataset for evaluating proximity preserving and structural role-based node embeddings. (2021). arXiv: 2101.03091 [cs.SI].

[8] [n. d.] Similarity-based link prediction in social networks using latent relationships between the users. https://www.nature.com/articles/s41598-020-76799-4#Sec4. [Online; accessed 03-03-2023]. ().

[9] Sucheta Soundarajan and John E. Hopcroft. 2012. Using community information to improve the precision of link prediction methods. *Proceedings of the 21st International Conference on World Wide Web.*

[10] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. (2018). arXiv: 1802.09691 [cs.LG].

[11] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: a review of methods and applications. (2018). DOI: 10.48550/ARXIV.1812.08434.