Guilherme Henrique Resende de Andrade

A Study on Graph Representation Learning for Automated Classification of Instagram Public Profiles

Advisor:
Fabricio Murai - Computer Science Department

Federal University of Minas Gerais
Institute of Exact Sciences
Computer Science Department

Belo Horizonte

2019/2

Abstract

This study assesses the use of graph representation learning methods in automatic classifying of Instagram public profiles. The methods are trained and validated on the Instagram dataset crawled by SmartData group from Politecnico di Torino over the 2019 European Elections that took place between May 23rd and 26th of the same year.

Keywords: Node2Vec, Graph Attention Networks, Graph Convolutional Networks, GraphSAGE, Geometric Deep Learning, Graphs, Instagram, Classification, Representation Learning.

Contents

Figures Summary

1 Introduction

Graphs are mathematical models capable of naturally representing a series of real world problems. Due to its consistency and usefulness, graphs have become ubiquitous in practical scenarios. However, considering the vast amount of data in real applications, graphs assume an intractable size and requires increasing computational power to process them.

One of the key activities when dealing with Machine Learning problems is to define the representation of the input. Frequently, choosing the wrong representation can lead to drastic performance damages. The input representation can be manually or automatically designed, however, this study will only consider automatic approaches often referred to as Representation/Feature Learning.

In this work, we employ an exploratory approach towards the use of graph representation learning to automatically classify Instagram profiles. Considering there is a wide range of families and methods regarding this topic, we decided to select the best performing models from some categories. We train and validate the embeddings with the dataset crawled and published by SmartData group from Politecnico di Torino.

Finally, we present some analysis over the best model results alongside explanations regarding the performance achieved when presenting the embedding to a simple linear classifier.

2 Dataset

The dataset used in this work was crawled by the SmartData[1] group from Politecnico di Torino. The records were collected in the period before and after the 2019 European elections that took place between May 23rd and 26th of the same year.

According to Trevisan et al. (2019), the data extraction started from 50 manually listed public figures which provided links to other popular profiles over the social network. New profiles were automatically added to the group of tracked users as long as they presented a minimum number of followers, 10 thousand in this case. The data was extracted from public Italian profiles and encompass five different categories, namely, Politics, Music, Shows, Sports, and regular users, each comprising respectively, 102, 53, 33, 75, and approximately 2 million non-tracked profiles. Each tracked user has many intrinsic features such as the number of profiles followed, the number of followers, the number of posts, and so on. However, none of the untracked users presents such information.

Instagram provides three atomic operations for users to perform over posts: comments, likes and shares. This dataset keeps track solely of comment interactions, which accounts for roughly 8 millions interactions in over 20 thousand different posts.

[1] ⟨https://smartdata.polito.it/instagram_monitoring/⟩

3 Related Work

In the last years, many applications where the core activity is built upon graph modeling have taken place in society, for example, Facebook, Instagram, and Google Scholar. Considering that extracting information from these structures can improve commercial models performances, as well as increase companies' earnings, efforts have been spent in improving the way this information is modeled and presented.

Hamilton, Ying e Leskovec (2017b) and Gurukar et al. (2019) present surveys on the representation learning literature and point many benefits and shortcommings of the listed methods. Based on these surveys, we decided to study the greatest performing model from Shallow Encoders, Convolutional Encoders, Attention Encoders and Inductive Encoders, namely, Node2Vec, Graph Convolutional Networks, Graph Attention Networks and Graph SAGE.

3.1 Node2Vec

Grover e Leskovec (2016) focused on developing an approach capable of capturing the diversity of connectivity patterns while maintaining some of the original characteristics in the new representation space.

However, the proposed approach heavily relies on a search strategy and, depending on which technique is employed, the resulting representation can either embed information from similar subnetworks or similar structural roles. The more topology information a method can grasp, the better the representations it will yield. Hence, as a way to extract information from both subnetworks and structural roles, Node2Vec employs biased Random Walk for sampling nodes that performs steps based on probabilities.

The sampling is simply a way to cast subgraphs into an input shape capable of being presented to a fully connected layer of a neural network latter on. This layer is responsible for mapping the nodes seen by the Random Walk to a latent space representation.

In Bengio et al. (2003), the representation intent was to instill information about the use and meaning of a given word, while modeling the conditional probability distribution of the whole language. Node2Vec algorithm is built upon the same objective function. However, instead of maximizing the probability of a word given its context, it models the probability of the context of a node, i.e the K nodes sampled immediately before and after it, given the node.

Formally, the objective function Node2Vec wants to maximize is described by

$$\underset{f}{\operatorname{argmax}} \sum_{v \in V} \log P(N_G(v)|f(v)), \tag{3.1}$$

where V is the set of all vertices comprised in a graph G, $N_G(v)$ is the neighborhood of the vertex v, and $f : V \rightarrow \mathbb{R}^d$ is a mapping function from nodes to their feature representations. However, to simplify the optimization problem, two modifications are proposed: (1) conditional independence (3.2), that is, the occurrence of a neighbor node does not influence the probability of any other neighbor node,

$$P(N_G(v)|f(v)) = \prod_{n_i \in N_G(v)} P(n_i|f(v)) \tag{3.2}$$

and (2) Symmetry (3.3) which ensures that the probability of n_i given v is the same as v given n_i:

$$P(n_i|f(v)) = \frac{\exp\left(f(n_i) \cdot f(v)\right)}{Z_v}, \text{ where } Z_v = \sum_{w \in V} \exp\left(f(w) \cdot f(v)\right), \tag{3.3}$$

where $\cdot$ if the dot product. One issue that arises from the formulation of Z_v is its computation cost for large problems. Hence, to reduce the time complexity, an approximation is performed by using Negative Sampling, an approach proposed in Mikolov et al. (2013) that performs substantially fewer updates over the weights increasing reducing significantly the execution time.

$$\max_{f} \sum_{v \in V} \left[-\log Z_v + \sum_{n_i \in N_G(v)} f(n_i).f(v) \right] \tag{3.4}$$

Hence, the resulting objective function is (3.4), which is optimized through Stochastic Gradient Ascent.

3.2 Graph Convolution Networks (GCN)

Kipf e Welling (2016) approaches the representation learning problem over a semi-supervised perspective where nodes' labels are considerably scarce. Some of the previously proposed works try to optimize a joint loss function like

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}, \text{ where } \mathcal{L}_{reg} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2, \tag{3.5}$$

where $\mathcal{L}_0$ and $\mathcal{L}_{reg}$ correspond to, respectively, the supervised and unsupervised loss, λ is a weighting factor, X_i and X_j are node feature vectors, $f(\cdot)$ is a neural network-like differentiable function and $A \in \mathbb{R}^{N \times N}$, the adjacency matrix.

Nonetheless, (3.5) models the scenario in which all interconnected nodes have a tendency to have the same label, which is not true in some of the real world networks. In contrast, with GCN, the graph structure is directly encoded by a neural model conditioned on the graph adjacency matrix, a process that will allow the gradient from the supervised loss to build the representations of nodes both with and without labels.

A convolution operation on a graph can be described by

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta, \tag{3.6}$$

where $\tilde{A}$ is the adjacency matrix with added self-loops, $\tilde{D}$ is the degree matrix of $\tilde{A}$, X represents the input signal, the calculation $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is the normalized Laplacian Matrix, and $\Theta \in \mathbb{R}^{C \times F}$ is the convolutional filter employed in the layer.

This modeling enables graph neural networks to minimize simpler optimization functions specifically designed to multiclass classification problems, such as Cross-Entropy loss.

3.3 Graph Attention Networks (GAT)

In the last decade, many reasearch efforts have been made towards generalizing convolutions to graphs. The available solutions can be categorized either as spectral or non-spectral approaches. Even though spectral approaches reached considerably good performances, there is still one major shortcoming: the heavy dependence on the graph's structure. This dependence prohibits a model trained on a specific graph to be used in another slightly different.

Inspired by recent advancements in the field of Deep Learning proposed in Vaswani et al. (2017), Veličković et al. (2017) suggested a new non-spectral approach solely based on attention mechanisms capable of generalizing to different graph structures.

Graph Attention Networks calculate importance weights (the model attention) to both neighbors and node features. This ability allows the model to naturally encode information about the graph structure while also learning how to combine nodes' information.

Attention coefficients $\alpha_{i,j}$ are calculated according to

$$\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(\overrightarrow{a}^{\top}[W\overrightarrow{h_i} \| W\overrightarrow{h_j}]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\overrightarrow{a}^{\top}[W\overrightarrow{h_i} \| W\overrightarrow{h_k}]))}, \tag{3.7}$$

where $W \in \mathbb{R}^{F' \times F}$ is a weight matrix, $\overrightarrow{h_i} \in \mathbb{R}^F$ is the node feature vector, $\overrightarrow{a} \in \mathbb{R}^{2F'}$ is a weight vector, and $\top$ and $\|$ are, respectively, transposition and concatenation operations.

The traditional form of the GAT forward function for one node is

$$\overrightarrow{h_i'} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{i,j} W\overrightarrow{h_j}\right), \tag{3.8}$$

where $\overrightarrow{h_i'}$ is the layer output, and σ is an activation function. Note that the calculation of attention considers a node's neighborhood, including itself.

According to Veličković et al. (2017), implementing multi-head attention to the regular model can be beneficial. From a practical point of view, many attention heads can learn to account for different things in different scenarios. For example, one of the heads can learn to pay attention to high degree neighbors, whereas the other one learns to pay attention to bridge nodes. (3.9) shows the functional form of a multi-head attention layer. As in (3.7), $\|$ represents the concatenation of vectors. However, in this case, each vector represents a different attention head.

$$\overrightarrow{h_i'} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{i,j} W\overrightarrow{h_j}\right) \tag{3.9}$$

It is worth mentioning, in the last attention layer, instead of concatenating the attention heads' vectors, the model averages them as a way to preserve the dimensions.

As for GCN, the choices of the loss function and optimization method may depend on the task under consideration.

3.4 Graph Sample and Aggregate (Graph SAGE)

Proposed by Hamilton, Ying e Leskovec (2017a), this approach tries to leverage performance by building a mixture of features and structural properties of the respective graph. These settings make the model capable of learning embeddings even in graphs with no features.

Instead of trying to optimize single node embedding vectors, Graph SAGE focuses on optimizing a set of K aggregator functions capable of extracting features from the local neighborhood of a given node as well as its features. This approach can be beneficial in the sense that they do not require optimization steps when predicting completely unseen nodes. Such a solution is extremely useful for constantly evolving graphs.

In order to build the embedding representations, the algorithm assumes the original node feature vectors as the initial embeddings, upon which aggregations are made. The first step consists in aggregating the local vicinity of a given node v into a single vector representation

$$h^k_{\mathcal{N}(v)} = \text{AGGREGATE}_k(h^{k-1}_u, \forall u \in \mathcal{N}(v)), \tag{3.10}$$

where $k \in 1...K$ is the k-th aggregation function learned by the model, $\mathcal{N}(v)$ is the set of neighbors of v, and h^{k-1}_u is the $(k-1)$-th aggregated vector from the immediate neighbors of the given vertex.

Once calculated all vertices' neighborhood intermediate representation (described in (3.10)), they are concatenated with their respective node representations and presented to a fully connected layer with a non-linear activation function σ.

$$h^k_v = \sigma(W^k \cdot \text{CONCAT}(h^{k-1}_v, h^k_{\mathcal{N}(v)})), \tag{3.11}$$

It worths citing that, to improve the computational cost, Graph SAGE - instead of using all - samples a fixed number of neighbors with a uniform sampling probability. However, this approach would cause information loss and, hence, for each step k, a different neighborhood sampling is performed.

4 Preprocessing

It is often possible to find data sources with valuable information for either academic or economic purposes. However, real world data presents value directly in its raw form. As a way to better shape the dataset described in Chapter 2 for this work, a series of transformations are performed in the data.

4.1 Cleaning

The dataset originally has 13 features to describe users profiles. After some consideration, only six of them were chosen: the number of followers, the number of profiles followed, the number of posts, the number of comments, the total number of comments from others in the user profile, and the number of self-comments. There are also 15 features to describe an interaction with a post, but only two features are considered to further purpose: the media author and the commenter.

Another important step employed in the data cleaning pipeline was to check whether or not there were null or duplicates values. Previous manipulations over the data already eliminated every invalid/duplicate entry and, consequently, no further record was removed during the current procedure.

4.2 Data Sampling

The data contains many sporadic interactions, for example, a user reacts to only one post over the entire analysis period. To reduce the computational burden and also to aggregate more significance to the remaining records' representations, users interacting less than 200 times over the entire period were discarded from further analysis. We emphasize that, all users listed under Politics, Sports, Shows, and Music were kept in the dataset even though some of they do not reach the minimum interactions threshold.

The sampling allowed us to reduce the number of users from $\approx$ 2 million to $\approx$ 1.8 thousand - roughly a 99.92% reduction - and the number of interactions from $\approx$ 8 million to $\approx$ 595 thousand.

4.3 Data Normalization

Another important step when dealing with machine learning models is to scale the input data. Besides facilitating the calculations, normalization often helps models to find better solutions and reduce the convergence time. The normalization applied here is a min-max scaling given by

$$X'_{ij} = \frac{X_{ij} - \min(X_j)}{\max(X_j) - \min(X_j)}, \tag{4.1}$$

where X_j is the column containing the j-th feature values from the input matrix X, and X_{ij} is the value of X_j corresponding to the i-th observation.

4.4 Feature Engineering

As mentioned in Chapter 2, only tracked users carry information about their profiles. However, to present the input data to the models, we would be required to first create a representation for the untracked profiles. Therefore, it would be necessary to input a neutral representation for these profiles. Instead, we opted for a manual feature engineering process.

With feature engineering we focus on building extra knowledge about the input based of the current features or information aggregated from outside the data extraction scope. In this case, we considered the information comprised in the interaction records to mold behavioural characteristics about the users. The engineered features are listed below.

- The average number of cited users;
- The average comment length;
- The average number of tags used over posts;
- The number of different profile categories a given user interacted with over time;
- The most active hour and weekday.

The engineered features were constructed to all users. Note that, when a given feature is not applicable to a specific record, it is filled with the zero value. In Figure 1, we can see the final format of the input data.

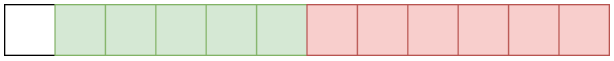

Figure 1: Final input format. The white square is a flag feature that assigns 1 to tracked users and 0 to non-tracked users, the green squares correspond to features regarding tracked users profile, and the red squares represent the engineered features

Considering all users' categories are expressed as text, it was necessary to assign integer values to the classes. As previously mentioned, there are five possible categories in the data, so labels were assigned as numbers in [0, 4].

5 Experimental Setup

We conduct experiments using the four models described in Chapter 3, namely, Node2Vec, Graph Convolutional Network (GCN), Graph Attention Network (GAT) and Graph Sample and Aggregate (GraphSAGE). Each model can be fine-tuned with respect to the embedding size, the number of hidden layers, and the number of epochs, except for Node2Vec which has, by definition, only one hidden layer.

This work considers Fey e Lenssen (2019) open implementations of GCN, GAT and GraphSAGE made available in PyTorch Geometric library. Unlike the remaining models, Grover e Leskovec (2016) provide Node2Vec's original implementation[1] which was the one chosen for this study.

- Embeddings Size: 64, 128, 256;
- Epochs: 25, 50, 100;
- Number of Layers: 1, 2, 3.

Considering the small number of nodes after the sampling process, it would not be a good approach to divide the data into train, validation and test sets. Instead, we avoid overfitting by using a Stratified 5-Fold-Cross-Validation. The experiments covered every combination of the parameters listed above and, since there are three different values for each hyperparameter, a total of 27 executions were made.

The experimental step has the intent of finding which one of the models obtains the best vector representations and what hyperparameters yield this performance. Once the best model is found, it is selected for further analysis.

The experiments were run in an Ubuntu 18.04.2 LTS machine with 8 GiB memory, an Intel Core i5-7200U CPU @ 2.5GHz x 4, 1 TB of disk memory, and without any Graphics Processing Unit (GPU).

[1] http://snap.stanford.edu/node2vec/

6 Results and Analysis

6.1 Models' Performance

As explained in Chapter 5, the experiments followed a 5-Fold-Cross-Validation approach and, hence, the results are averaged over all five executions. In Table 6.1, we can see the best F1-Macro for each one of the four models. Note that, each hyperparameter combination was executed for 25, 50, and 100 epochs and the reported performances are the maximum value from these three executions.

		Number of Hidden Units		
Models	**Number of Layers**	**64**	**128**	**256**
Node2Vec	**1**	0.185 ± 0.000	0.185 ± 0.000	0.185 ± 0.000
GCN	**1**	0.184 ± 0.000	0.184 ± 0.000	0.184 ± 0.000
	2	**0.209 ± 0.019**	0.195 ± 0.010	0.192 ± 0.006
	3	0.187 ± 0.015	0.204 ± 0.019	0.200 ± 0.013
GAT	**1**	0.185 ± 0.004	**0.189 ± 0.005**	0.185 ± 0.002
	2	0.185 ± 0.002	0.184 ± 0.000	0.184 ± 0.000
	3	0.178 ± 0.034	0.185 ± 0.023	0.184 ± 0.000
GraphSAGE	**1**	0.184 ± 0.000	0.184 ± 0.000	0.184 ± 0.000
	2	0.186 ± 0.004	0.184 ± 0.000	**0.189 ± 0.006**
	3	0.154 ± 0.066	0.184 ± 0.000	0.184 ± 0.000

Table 1: Averaged F1-Macro performances alongside with standard deviations. The best performance of each model, when existent, is highlighted in bold.

The F1-score, also known as the harmonic mean of Precision and Recall, is a metric used to calculate performance of a binary classifier. However, it can also be used to calculate the accuracy of multiclass classifiers. When dealing with multiclass problems, the metrics are calculated individually for each class and then, combined in a final result.

We decided to consider solely the F1-Macro measure in the analysis, which is a simple average over the F1 scores from each class.

Table 6.1 shows that the GCN model with two convolutional layers and an embedding of size 64 achieves the highest performance over all experimented models. Nonetheless, its performance is still considerably low and with a large standard deviation, making the differences non-statustucally significant

The models' low performance comes from the fact that, most of the time it predicts nodes as unknown users. The models are unable to learn segregating features through the training process, probably due to the data unbalance presented in Figure 2 or the lack of significant connection patterns. If we consider the definition of F1-Macro and the actual model performance, we can conclude that probably, only in a few cases the model is predicting something different than unknown users.

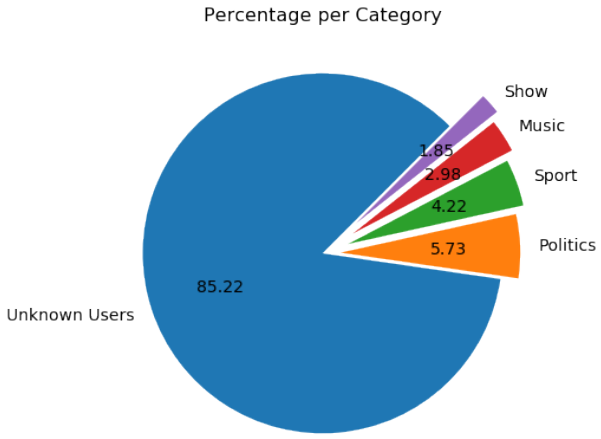

Figure 2: Categories representativeness.

With the current data distribution, a model predicting all nodes as Unknown Users would be capable of reaching an 18.4% of F1-Macro measure. From Table 6.1, we can see that all models present performances around these values.

6.2 Categories Differences

As shown in Figures 3 and 4, there is a natural differentiation on the classes' average feature values, however, 6 out of 12 features present a similar behavior along the classes of tracked users. Nonetheless, even though the tracked users may resemble each other, they are completely different from the untracked users average features values, which reinforces

the hypothesis that models are not capable of generalizing well due to data unbalancing.

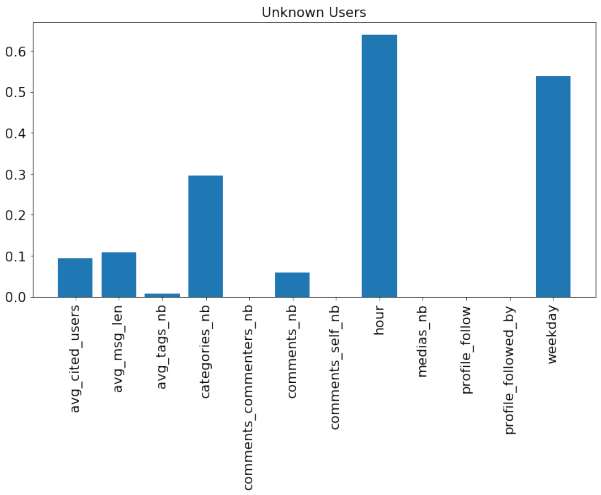

Figure 3: Mean feature values from unknown users.

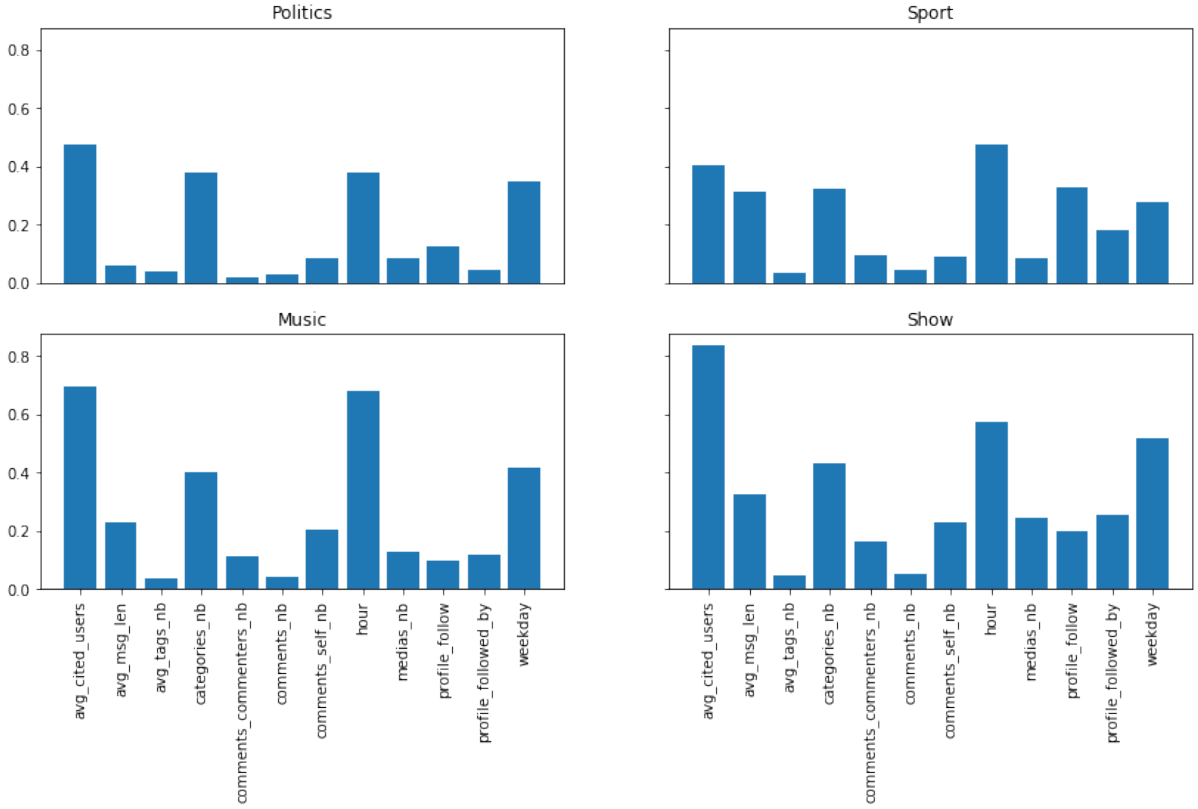

Figure 4: Mean feature values from Politics, Sport, Music, and Show.

6.3 Embeddings

When dealing with raw data, much information from the response variable is spread over the features. This interlacement can harm models' performances when it cannot be untangled. One way this information can be separated, even though not preserving the features' original meaning, is through representation learning.

Figure 5 suggests that there may be a natural differentiation between the data points. However, the available features do not seem to be good descriptors of the classes, which can have a direct impact on the final performance of models.

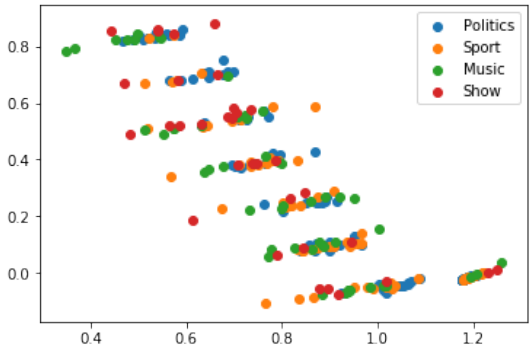

Figure 5: Raw normalized features 2D.

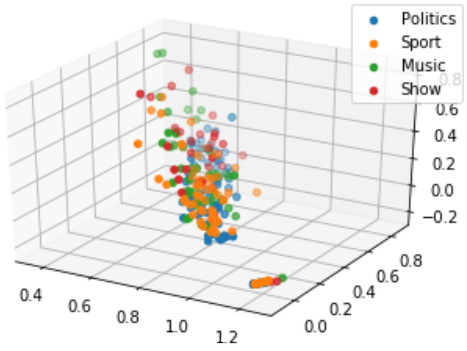

Figure 6: Raw normalized features 3D.

As we can see below, Figures 7 and 8, the representation learning procedure do not seems to be useful to increase the contrast between classes. Instead, it suggests a completely contrary effect, if compared to Figure 5.

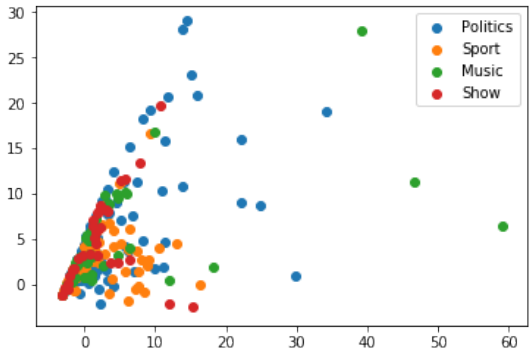

Figure 7: Best model's 2D embeddings.

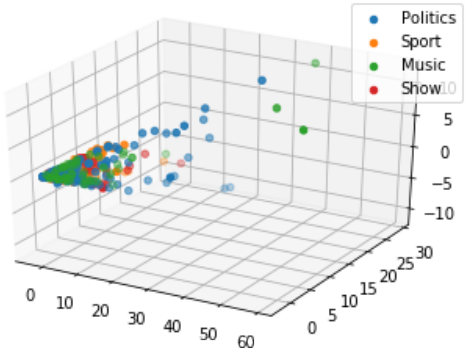

Figure 8: Best model's 3D embeddings.

6.4 Traning Error

The characteristics presented in Section 6.3 might be indicative of the model's disability to find its way to convergence. However, as shown in Figure 9, the training error history presents a monotonically decreasing error curve, which contradicts the hypothesis of the model being incapable of converging. Since train and test steps followed a 5-Fold Cross-Validation fashion, the error curve below is the average over the executions.

Figure 9: Average train error.

When analyzing the plot above, we can argue that the curve still presents a non-zero derivative. In other words, probably when trained for more epochs, the model would be able to reach better performances. Nonetheless, the model does not present any improvements when trained with both 250, 500, and 1000 epochs, even though the average train error diminishes.

7 Conclusion

In this work, we tested different graph representation learning techniques with simple classifiers. According to the results, none of the graph-based approaches performed satisfactorily.

Among the reasons that probably motivated the bad performances are the data unbalancing, the lack of descriptive features, and potentially the absence of representative connections after the graph being sampled.

For future work, we intend to experiment with different sampling techniques, exploit the feature engineering process in order to find more discriminatory information, and contrast graph-based approached with traditional machine learning approaches for the same classification problem.

References

BENGIO, Y. et al. A neural probabilistic language model. *Journal of machine learning research*, v. 3, n. Feb, p. 1137–1155, 2003. page.99

FEY, M.; LENSSEN, J. E. Fast graph representation learning with PyTorch Geometric. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. [S.l.: s.n.], 2019. page.1616

GROVER, A.; LESKOVEC, J. node2vec: Scalable feature learning for networks. In: ACM. *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2016. p. 855–864. page.88, page.1616

GURUKAR, S. et al. Network representation learning: Consolidation and renewed bearing. *arXiv preprint arXiv:1905.00987*, 2019. page.88

HAMILTON, W.; YING, Z.; LESKOVEC, J. Inductive representation learning on large graphs. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. p. 1024–1034. page.1212

HAMILTON, W. L.; YING, R.; LESKOVEC, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017. page.88

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. page.1010

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2013. p. 3111–3119. page.99

TREVISAN, M. et al. Towards understanding political interactions on instagram. *arXiv preprint arXiv:1904.11719*, 2019. page.77

VASWANI, A. et al. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008. page.1111

VELIČKOVIĆ, P. et al. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. page.1111