Studying Lobbying Influence in the European Parliament using Twitter data
CS-433: Machine Learning, Project 2

Mohamed Allouch, Kamil Czerniak, Benedek Harsanyi
Ecole Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—Unclear relationships between politicians and lobbyists are a problem in a democratic political framework. Using various methods, we attempt to detect links between Twitter activity of Members of the European Parliament and lobby groups. The resulting models are an improvement over basic text embedding analysis.**

I. Introduction

Relationship between business and politics is a well-known problem within democratically-elected governments. The influence lobbyists have on politicians, either through reaching out to politicians themselves or through various forms of gratification, may move power away from the electorate to these groups. Various regulations like lobbying registries or limits on what lobbying activity is allowed an attempt to control this phenomenon so that it doesn't become a threat to democratic order. However, lobbyist organizations may attempt to circumvent both lobbying legislation and public scrutiny by utilizing other groups or accounts to spread their message. One such medium through which this may happen is a social platform called Twitter, where users can write short posts (”tweets”), some of which may relay posts from other accounts. In this project, we seek to explore the relationship between tweets of Members of European Parliament (MEPs) and known lobby groups and through this attempt to find ”unseen” links. The goal is to detect these unknown relationships between MEPs and lobby groups, helping bring these relationships to light.

II. Data Exploration

A. The Retweet Graph

The dataset contains tweets of 677 Members of the European Parliament (MEPs) of 8th term (2014-2019) and 3024 lobby groups, alongside metadata related to tweets and referenced tweets. A tweet can be referenced by either republishing it without comment (retweet), republishing it with a comment (quote retweet) or replying to it. As the dataset contains all tweets made by given users in the years 2014-2019, we also have a lot of tweets that do not refer to anyone or refer to accounts outside of our interest (in this situation: MEP accounts for lobby tweets and vice versa). On this base, the retweet graph has been built as follows:

1) The tweet must have originated from an MEP account and referenced a lobby account or the tweet must have originated from a lobby account and referenced an MEP account.
2) When such a tweet is found, we create two nodes in the graph (if they do not exist already), named after Twitter handles of both tweet creator and referenced person. Each node has two one-hot features - ”isMep” and ”isLobby”
3) We create an edge between two created nodes, with every edge containing data on the language of the tweet, whether it is a retweet, original tweet ID, information about referenced tweet (ID of referred tweet and type of reference, e.g., a retweet or a reply) and from which file the original tweet originates.

The result is a multigraph, with each edge representing a tweet of a user referencing another user. As it can be noticed, every node can either be an MEP node or a lobby node and we only create edges if there is a tweet from an MEP to a lobby account (or vice versa) - hence, the graph is bipartite and the split into two groups can be done using features on the node. Data on the edge help in filtering by language and whether the tweet was a retweet (compared to, e.g., a response) and direct reference and source file, which helps in tweet text lookup.

After this preprocessing, the result is 2339 nodes (588 MEP nodes and 1751 lobby nodes)

B. Processing the tweets

We restricted ourselves for English-language tweets, making 5143438 observations in total. The dataset consists of two parts, there are 474361 tweets from Members of the European Parliament and 4669077 tweets from lobbies. The mean character length of the tweets is 132.42 with a standard deviation of 60.92. Firstly preprocessed the textual data using the Natural Language Toolkit (nltk) library[1]. The preprocessing pipeline consists of case-folding, lemmatization, tokenization, stopword, and punctuation removal. A popular idea in machine learning is to represent words by vectors this technique is called embedding. The main advantage of pre-trained embeddings is that they can be used for basically any desired ML algorithm. We decided to use fastText [2] for this step in our pipeline, which produces a word embedding of 300 dimensions. The embedding of a tweet is given by the sum of the embeddings of the words in it.

III. Unsupervised Methods

Clustering of graph vertices is a task related to community detection within networks. The goal is to create a partition of the vertices, taking into account the topological structure of the graph, in such a way that the clusters are composed of strongly connected vertices. Graph clustering techniques are very useful for detecting strongly connected groups in a graph. Inside each cluster, it is more likely to find common patterns shared by users such as political subjects, country, or industrial fields...

A. *Louvain algorithm for graph clustering*

The Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network. The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

After running the algorithm on the bipartite graph, we discover 11 clusters, in which two clusters are not relevant (In each cluster, there is only two nodes, one lobby and one Mep that retweeted each other)

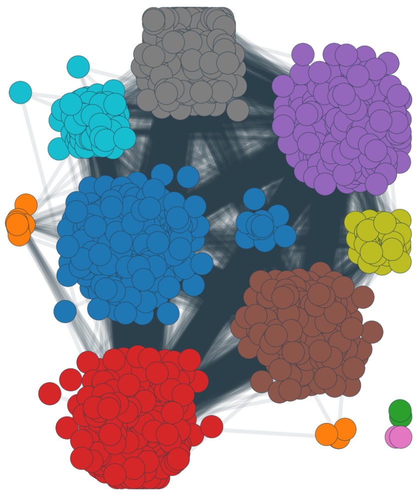


Figure 1. Clustered bipartite graph.

B. *Topic discovery within each cluster*

Topic analysis is a Natural Language Processing (NLP) technique that allows us to automatically extract meaning from text by identifying recurrent themes or topics. For our project, we used topic modeling techniques, topic modeling is an unsupervised machine learning technique. This means it can infer patterns and cluster similar expressions without needing to define topic tags or train data beforehand. We opted for LDA (Latent Dirichlet Allocation) [3] to classify text in a tweet to a particular topic, It builds a topic per tweet model and words per topic model, modeled as Dirichlet distributions. To do so, tweets pre-processing is made and includes:

- Tokenization: Split the tweets into words. Lowercase the words and remove punctuation.
- All stopwords are removed.
- Words are lemmatized — words in third person are changed to first person and verbs in past and future tenses are changed into the present.

After running the model with 10 topics as a hyper parameter, we get the following topics for each cluster:

Cluster	Topic
0	Energy
1	Human rights
2	Irrelevant cluster
3	Health
4	Technology
5	Irrelevant results
6	Irrelevant results
7	Climate
8	Ecology
9	Brexit
10	Sports
11	irrelevant cluster

LDA algorithm can be applied quickly and easily, but there's a downside: in some cases the results are inaccurate (cluster 5 and 6).

IV. Supervised Methods

A. *Link prediction*

Our goal is to build a machine learning model that, based on the tweets of two users, can accurately predict if a retweet link exists between them. Given 580 MEPs and 1731 lobbyists, there are 1003980 possible retweet link but we only observed 13632 actual links, which means our dataset is really unbalanced. We use a classical supervised binary classification approach. Note that the number of MEPs and lobby groups also shrank since we excluded non-English tweets and isolated nodes.

B. *Baseline Model*

Given the bipartite graph G we will construct edge weights w between all possible MEP-lobby pairs. Let $u \in U$ and $v \in V$ be two nodes, then we denote the retweet embeddings by $X_u \in \mathbb{R}^{300\times d_u}$ and $X_v \in \mathbb{R}^{300\times d_v}$, where d_u and d_v are the number of tweets of the user. We define the weight between these two nodes by the maximum similarity distance between the tweets' of the users, which is

$$w_{uv} = \max(X_u^T X_v) \in \mathbb{R}$$

Note that our baseline model does not require any learning, we can use the resulting weights for classification on all the data, with a given θ threshold. Our classifier can be written as $f(u,v) = \mathbb{1}_{\{w_{uv} \geq \theta\}}$.

C. Logistic Regression

Similar to the baseline model we will construct an edge feature vector out of the tweet matrixes. Let us denote the sum embedding of the ith tweet of user u with the vector $x_{ui} \in \mathbb{R}^{300}$. For a given node we define

$$w_u = \sum_i x_{ui}$$

that is the sum of embeddings of tweets made by a given user. When considering an edge, we can stack these weight vectors, resulting in a edge weight between users u and v being $w_{uv} \in \mathbb{R}^{600}$. Using these edge weights we can classify edges using logistic regression. Logistic regression is represented as a one-layer neural network, with sigmoid function used as an activation function. We are using Adam optimizer and binary cross-enthropy loss function. We use train-test split of 80%-20%: 20% of positive edges (edges in graph where we do have a link between users) are left for testing and similar size (not proportion) is used for negative edges (where such link does not exist) - similar with testing sets and proportion of 80%. We use L_2 norm to normalize weights, to limit overfitting.

In order to determine the learning rate for optimizer and number of epochs used, we prepared a script (`cross_validation.py`) that iterated over multiple values of the learning rate, regularization coefficient (known as weight decay) and epochs and based on this combination it assessed resulting models using average AUC (area under the graph). We use 5 folds for cross-validation, based on training data, to figure out what hyperparameters to use. Following our experiments, we determined that best AUC could be achieved for a learning rate of 0.001, regularization coefficient of 0.01 and 1000 epochs.

D. Graph Convolutional Network

In our two previous model we did not take into consideration the underlying graph structure, we dealt with the edge prediction task independently for every edge. One possible approach of managing ML tasks on non-Euclidan structured data is using a so called Graph Convolutional network (GCN) [4] model, which can be shown to be a generalization of the CNNs on arbitrary graphs [5].
Given a graph $G = (V, E)$, Let us denote the adjacency matrix with $A \in \mathbb{R}^{N \times N}$, the node features of G with $X \in \mathbb{R}^{N \times D}$, where $N = |V|$ and D is the number of features. By adding self-connections we define $\tilde{A} = A + I$ and the corresponding diagonal matrix $\tilde{D} = \sum_j \tilde{A}_{ij}$. The GCN layer is defined as function $f_W(X, A)$ parametrized with $W \in \mathbb{R}^{D \times C}$, where C indicates the number of channels. Let us denote the output vector corresponding to node i with h_i, the propagation rule for a node can be written as

$$h_i = \sigma(b + \sum_{j \in N(i)} \frac{e_{ji}}{\sqrt{\tilde{d}_i \tilde{d}_j}} x_j W)$$

where d_i is the ith diagonal element of $\tilde{D}$, and σ is the activation function. Using matrix notation we can rewrite it as

$$f_W(X, A) = H = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X W).$$

It is possible to stack multiple layers on top of one another. The GCN layer creates node embeddings by aggregating neighborhood information from previous layers. The node representations can be used for link prediction, the probability of the existence of an edge between node i and j is given by the score function $\psi(h_i, h_j) = y_{ij}$, which can be a simple inner-product or a neural network, which input is the concatenation of the two vectors.

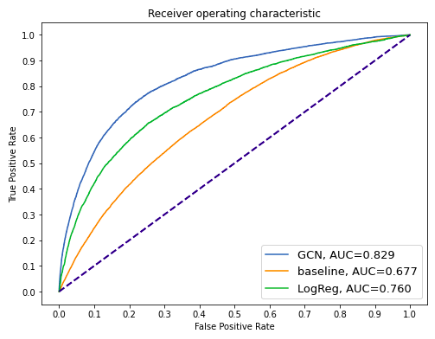
E. Training the GCN model

Real-world examples of graphs are usually sparse, meaning there will be significantly more non-edge examples in our dataset. Similar to the Logistic Regression approach, we will hide 20% of the edges from the graph and treat them as test edges. In order to balance this set, we will sample the same amount of negative edges and include them in the test set as well. The remaining graph will be considered as our training set. The initial node representations are given by the average embeddings of the corresponding tweets of the user. We pass the embeddings to the GCN and use the score functions for prediction. The model is trained with the binary cross-entropy loss. In our implementations, we used Pytorch [6] and the Deep Graph Library (DGL) [7], which is a popular package for performing dense tensor operations on graph-structured data.

V. Results

The optimal hyperparameters were chosen both for the logistic regression and the GCN model with 5-fold cross-validation. This includes learning rate, weight decay rate, number of epochs and the number of channels for the GCN model. For a random selection of edges, a random number generator with seed 42 was used - this was meant to ensure the reproducibility of the results. We retrained the best models and evaluated them on the test set. All of our models output can be considered as probability vectors of the binary label, meaning with different classification thresholds we get different results. In order to evaluate the models, we compute the true-false positive rates for different thresholds and plot the receiving operating characteristics. We use the maximum value of the geometric mean of true-false positive rates to determine the best threshold. We report the f1-score, the area under the model's ROC curve (AUC) and the accuracy achieved with our chosen threshold.

Method	f1-score	AUC	accuracy
Baseline	0.043	0.677	0.627
Logistic Regression	0.363	0.760	0.702
GCN	0.465	0.829	0.757



VI. Conclusions and future work

The AUC metric is significantly greater than 0.5 lets us conclude that even our baseline model performs better than predicting randomly. Applying deep learning methods to the task seems to be the most beneficial approach, although it is easier to interpret a logistic regression model. There are many possible benchmark frameworks for dealing with non-Euclidean data. GCN is one of the simplest models from the rich family of message passing-based graph models. A further improvement in our model could be to change the convolutional layer to another state-of-the-art layer, such as GraphSage, Graph Attention Networks or Weisfeiler-Lehman GNN [8]. Another popular approach for link-prediction tasks with the GCN model is using graph variational autoencoders [9]. Another limitation is determining the initial node embeddings on our graph. The danger of taking the average embeddings is that we might lose a lot of important information. Instead of that one might try to apply Attention mechanism on the tweets to get a more robust initial representation [10].

References

[1] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[5] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, pp. 3844–3852, 2016.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[7] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs." 2019.

[8] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[9] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf