# Identifying Industries in Economic Networks
Using Network Characteristics to Identify Industry Types of Unlabelled Nodes

Brian Reed

## I. Introduction

Following the financial crisis, economists increasingly turned to network science to understand interdependencies between firms. Many economists focused on financial networks that captured which firms were invested in which other firms. Elliot, Golub, and Jackson [7], for instance, focus on cascades in asset values. Other economists focused on production networks, which capture the firm-firm transactions through which firms buy goods and services. Acemoglu et al [1], for instance, describe how network structure can mediate or exacerbate how widely output shocks spread.

There is an increasingly rich theoretical literature on economic networks. The empirical literature is more limited, especially the empirical literature on production networks. This paper is part of a broader project that aims to use a network of firm-to-firm transactions in a small country to get a better sense of how shocks spread from firm to firm.

### A. Our Goal & Why it Matters

For 70% of the nodes in our dataset, we know which of 16 industries the firm belongs to. Ideally, we would be able to fill in some of the missing values so we can get more power in our estimation. Here we evaluate the effectiveness of several classification algorithms in guessing the industry of firms in the dataset for whom we do not have this information. We conduct this task primarily to help in our downstream estimation process. However, this task also raises some interesting intellectual questions. If we are able to guess industry type based on network characteristics alone, this could help to open up a literature on the structural roles of industries within a production network.

Acknowledgements: I have been working on this dataset with Robi Palovics and Ben Hannel in the SNAP group. This classification work is my own, but I shared some of background data prep steps with the two of them. I followed many of the keras and sklearn guide pages to construct the classification techniques, and drew from Jason Brownlee's blog and others for guidance and best practices.

Code is available at: https://github.com/brianreed21/229project

### B. ML Framing

We treat this as a supervised classification problem with 16 classes. The input will be features of the firm that we extract from a network of firm to firm transactions. We then use experiment with multinomial logistic regression, KNN vote, and a basic neural net to predict the industry of the node.

## II. Related Work

Increasingly, mainstream economists are considering the role of network structure in determining economic outcomes. Jackson has focused on a range of topics, such as the aforementioned financial networks and the role of social networks for determining economic outcomes [12]. Some papers have focused specifically on characterizing different aspects of production networks, such as [3]. Until recently, network approaches were the domain of complexity scientists like Cesar Hidalgo [11].

On the network and computer science side, there is a growing field of graph representation learning (see [10] for a summary). These approaches including embedding approaches like factorization, which draws on matrix factorization techniques, and random walk approaches, which traverse the network and learn statistics about it. There are also graph neural networks, which try to characterize "messages" passed between nodes, typically a subgraph. We will use one of these approaches, node2vec[9], to achieve a lower dimensional representation of our network. In addition to these network-specific approaches, there is a rich literature on improving outcomes in classification tasks, for instance , which looks at techniques to get around the issue of imbalance in classes. It seems that so far, however, little work has been done at the intersection of graph representation learning and economics.

## III. Overview of the Dataset

We have records of every transaction above 50,000 euros coming from or going to a firm in a small Europen country for the period from 2008-2017. Each entry contains an anonymized label for the firm that is sending the money as well as the firm that is receiving the money. These labels are consistent over time, which allows us to track firms. There is a timestamp on each transaction. For our downstream analyses, we will focus on transactions that only involve domestic, non-governmental, non-financial firms, so we adopt a similar focus here. A simple transaction is depicted in Figure 1.

Source:
Firm A
$
Target:
Firm B

Fig. 1. A transaction consists here of Firm A (buyer) sending money to Firm B (supplier) in exchange for a good or service.

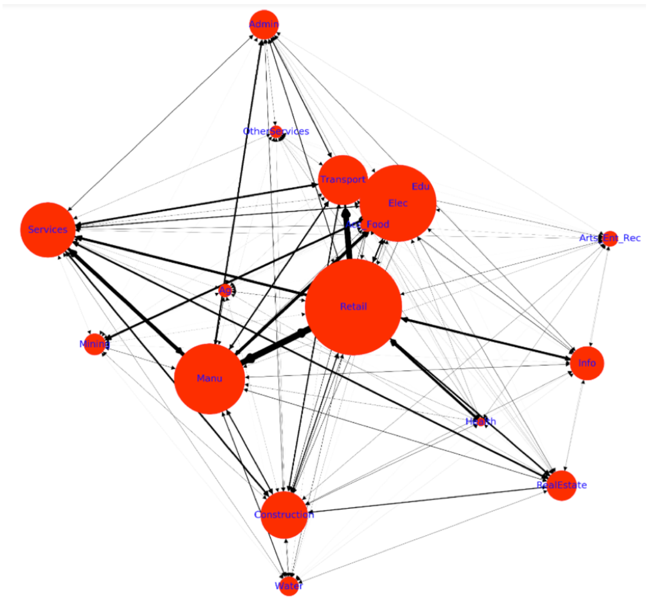

Fig. 2. Industry to industry transactions network across all years. We focus on a firm to firm network, but this helps give a sense of connections between different industries. The size of the nodes is proportional to the amount of edges coming in to the industries, and edge width is proportional to the value passing over it.

### A. *Network Construction & Train, Validation, and Test Splits*

We construct one network for every year in our dataset. This network contains a single, unique edge for every unique firm-firm transaction. We construct a directed and an undirected version of the unweighted graph and extract different features.

We focus on a firm to firm transactions network, but for reference, we plot an industry to industry version of our data in Figure 2. We can see several patterns here that should help inform our classification task. Retail, electricity, and manufacturing are the largest players by transaction value, while arts, entertainment, and recreation is particularly small.

We have 96,633 firm-year observations, which we randomly assign to the training, validation, and test sets in an 80/10/10 split. Per convention, we iteratively develop our model using the training and validation sets. Once we have settled on a model, we can evaluate its performance on the test set. We standardize the data in each set by subtracting off the mean and dividing by the standard deviation.

### B. *Feature Extraction*

We extract features for each node in each year. This means that if a firm shows up in the dataset for multiple years, it will show up in our dataset multiple times as well, though its features will be different each time. We consider a combination of hand-engineered and machine-learned features that describe a node's position within the graph. This allows us to gain some insight into the extent to which we can differentiate industry types based solely on the network context of a firm.

The hand-engineered features include the in-degree, out-degree, PageRank Centrality, and local clustering coefficient of the node. Our machine-learned features include a 20-dimensional embedding of each node, obtained using Leskovec and Grover's node2vec algorithm [9].

*node2vec:* This algorithm finds embeddings for each node by optimizing the objective function below, where we have a set of nodes $V$, a some sampling strategy $S$, and a neighborhood of node $u$ that is found under sampling $S$. In effect, it maximizes the probability of finding its neighbors given the embeddings $f$ ([9]; also referencing course notes from CS224W):

$$\max_f \sum_{u \in V} \log Pr(N_s(u)|f(u))$$

It conducts this optimizing using stochastic gradient descent on the log-likelihood function. It finds the information for this optimization problem by conducting a second-order random walk through the network, where it is second order because it remembers where it came from. We use an implementation from the node2vec package with the default settings for $p$ and $q$, which means that we favor neither local nor more distant parts of the network. We use this to learn 20 dimensional embeddings for each node, using the networks for each year.

### C. *(Im)Balance Across Classes*

As is common in multi-class classification problems, we have many more instances of some classes than of others (Figure 3). We consider one approach

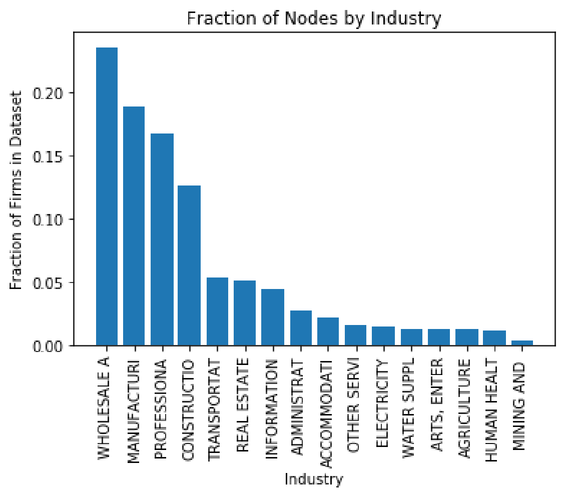

Fig. 3. Fraction of node-years in each industry.

to improve the balance across classes, though our methodology performs reasonably well without this.

## IV. Methods

We proceed by treating this as a supervised classification problem. We briefly considered treating this as a semi-supervised problem, as we know some of the labels and do not know others. However, the unsupervised approach usually is applied to cases where we have more unlabelled examples than labelled examples, and here we have about two times as many labelled examples as unlabelled ones.

Here we consider evaluation metrics, our three different classification approaches, our tuning process for our neural network, and one rough approach to account for class imbalance question.

### A. *Evaluation Metrics*

In evaluating the performance of different classification algorithms, it is helpful to take into account methods beyond just accuracy. This is because a model can be "accurate" and still be a poor classifier if the data are predominantly composed of one type of example, as then the classifier could achieve a high accuracy by always predicting that type.

We consider three metrics in addition to accuracy: precision, recall, and F1. We consider the macro versions of each of these, which give us a sense of average performance across classes. We consider industries $j \in J$ and nodes $i \in I$; formulas adopted from [15].

*1) Accuracy:* Accuracy tells us how many nodes we predicted correctly:

$$\frac{\sum_j \sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} = j\}}{|I|}$$

*2) Precision:* In the binary case, precision is the odds of a Type I error. In our case, precision penalizes us if we say a node is of a given industry when it is not.

$$\frac{1}{j}\sum_j \frac{\sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} = j\}}{\sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} = j\} + \sum_i 1\{y^{(i)} \neq j, \hat{y}^{(i)} = j\}}$$

*3) Recall:* In the binary case, recall is the odds of a Type II error. In our case, it penalizes us when a firm is of a given industry and we say it is not:

$$\frac{1}{j}\sum_j \frac{\sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} = j\}}{\sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} = j\} + \sum_i 1\{y^{(i)} = j, \hat{y}^{(i)} \neq j\}}$$

*4) F1:* This is the Harmonic mean of Recall ($R$) and precision ($P$). It penalizes algorithms with precisions or recall values that are in the extremes:

$$\text{F1} = \frac{2RP}{R+P}$$

### B. *Classification Methods*

*1) Multinomial Regression:* The multinomial regression is our baseline result here. We estimate parameters $\theta_1, ..., \theta_{k-1}$ to calculate the probability that an example is of a given class given our data. This results in a vector of probabilities for every example, with the assignment going to the class of maximum probability.

$$h_\theta(x) = E\left[\begin{bmatrix} 1\{y=1\} \\ \vdots \\ 1\{y=k-1\} \end{bmatrix} |x;\theta\right] = \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_k^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_j \end{bmatrix}$$

We implement this using the logistic regression function within sklearn's linear model, which has a multinomial option. This solver trains the model according to a cross entropy loss function, where for every training example, the loss is given by:

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} 1\{y = j\} \log \hat{y}_j$$

*2) KNN Vote:* As another baseline, we use the KNN vote procedure in sklearn. This procedure finds the closest $k$ neighbors to each given point, and assigns that point the same class as the plurality of its k-nearest neighbors. Here, we just use $k = 15$ as a baseline, but this is a tunable parameter. In this case, we use the Euclidean distance metric. In this case, say we have features $i \in I$ for each observed node $x$ and "unobserved" $y_i$:

$$dist(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

*3) Basic Neural Network:* Against these two baselines, we compare a densely connected sequential model in Keras. In this model, every node is connected to every node in the next hidden layer. We use ReLU activation in each of the hidden layers, and we use a softmax in the final hidden layer. Each ReLU has form:

$$g(z) = 1\{z \geq 0\}$$

The softmax has the form given in the multinomial logit case for some input $x$. We train the model using a cross entropy loss function, again as in the multinomial case.

*Experiments:* We start with a basic neural network that has 10 nodes with 10 hidden units each. The performance on this was not significantly better than in logistic regression, with, for instance, accuracy values around 0.25 and similar loss (sub-figure 1 in Figure 4).

Following this, we dramatically increased the number of hidden units in each layer, upping this to 1000 units in the first hidden layer and 500 in the remaining few. It's worth revisiting this decision and experimenting with adding more hidden layers instead of units, but our performance dramatically increased, although we dramatically overfit to the training set (sub-figure 2 in Figure 4).

We next added dropout regularization, at first with a rate of 0.5 in each layer, meaning that it randomly dropped 0.5 of the neurons. This caused our training and validation curves to more closely track each other, though we saw accuracy peak and then slightly decrease over time (Figure 4, center).

Following this we added learning rate decay of $1 \times 10^{-6}$, from our baseline of 0.001. Our final model generates the curves in sub-figure 4 in Figure 4. There is still evidence of overitting to the training set, but our results did generalize to the test set, as discussed below.

Throughout this process, we experimented with different optimizers and settled on Adam, as it seemed to converge the fastest. Our loss function is cross entropy, as in the multinomial case above.

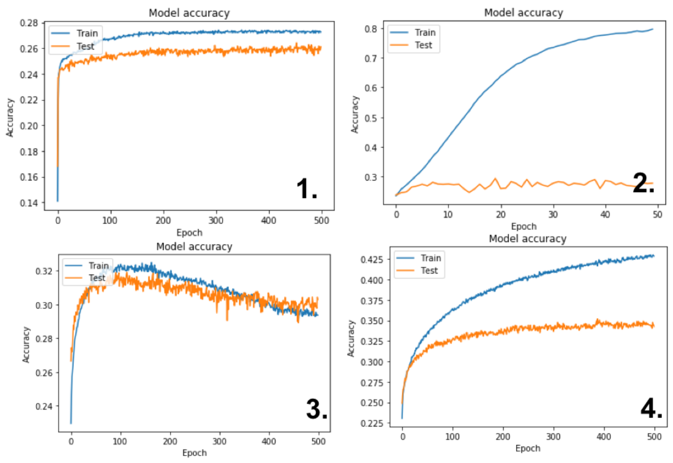

Fig. 4. Tuning process. Baseline model is 1; more complex model is 2; more complex model with dropout regularization is 3; more complex model with dropout and learning rate decay is 4. Blue cruves are training and orange are validation.

*Balanced Model:* We use the hyperparameter settings from this tuning process to train a model using a balanced version of the data. We balance in particular the training data for this approach by up and down-sampling different industries so there are 5,000 observations in each. This is a crude alternative to reweighting as described in [6]. We then predict on the standard validation and test sets.

## V. Results

First, we present a table of the accuracy, precision, recall, and F1 scores of the different models (Table I). We note a few things. First, the accuracy of each model was better than random (6.67%), which is reassuring. The accuracy of the logistic regression is comparable to the accuracy of the KNN vote, but the precision and recall are both very low, as is its Macro F1 score. This performance is at least partially explained by the fact that the logistic regression only made predictions across the 2 classes that were most common in our data set. Conversely, the neural nets and the KNN approaches generated predictions across all classes.

| | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Logistic Regression | 0.257 | 0.084 | 0.0723 | 0.0778 |
| KNN Vote | 0.285 | 0.244 | 0.118 | 0.159 |
| Balanced Neural Network | 0.136 | 0.23 | 0.15 | 0.181 |
| Final NN | 0.501 | 0.31 | 0.799 | 0.45 |

TABLE I
EVALUATION METRICS. BASELINE RESULTS ARE FOR THE LOGISTIC REGRESSION AND KNN VOTE, AND OUR FINAL RESULT IS WITH A TUNED NEURAL NET, DESCRIBED IN SECTION IV-B3.

We note that our balancing method made our model performance demonstrably worse across all metrics than our other neural network, though its F1 score was better than KNN vote and logistic regression. The performance of this model was likely sabotaged by the fact that we had to drop more than a third of our sample, and repeat other observations to replace some that were dropped.

We can dig into the results from our simple neural network a bit more by plotting a confusion matrix (Figure 5). The rows of this matrix detail the actual industry that the nodes belong to, and the columns detail the industry that the model predicted. Along the diagonal, we can see the recall for each industry.

We see that the recall values are fairly volatile here, bottoming out at 41% for industry label 6, which is wholesale and retail trade, and 42% for industry label 11, which is professional services. On the other hand, our recall is perfect for industries 0 and 1, which are agriculture and mining. It is interesting to note that our recall values do not strongly correlate with the number of samples in each industry. This could suggest that there are strong underlying differences in the observations that are driving the results.

## VI. Discussion

With more time to address this problem, we would experiment with additional reweighting schemes as described in [6]. After finding a way to better account for the class imbalance, we could devote additional energy to better tuning the neural network in order to improve our classification performance. Further, if we wanted to change the problem and simply optimize our predictions, rather than trying to make our best predictions using only network data as here, we could include further information from a firms' balance sheets.

In terms of the interpretation of these results, it is interesting that we were able to get such high recall values for certain types of industries. This suggests that there may be certain aspects of their placement within the network that clearly differentiate one type of firm from another, which is an idea potentially worth exploring further. One way to get additional insight into the factors driving this would be to potentially include different types of features and examine how our performance differs. For instance, we could train our node2vec embedding in a way that favors local features, and we could separately train our embedding in a way that favors more distant features of the network, then use this to comment on which types of features may be more relevant to our prediction problem.

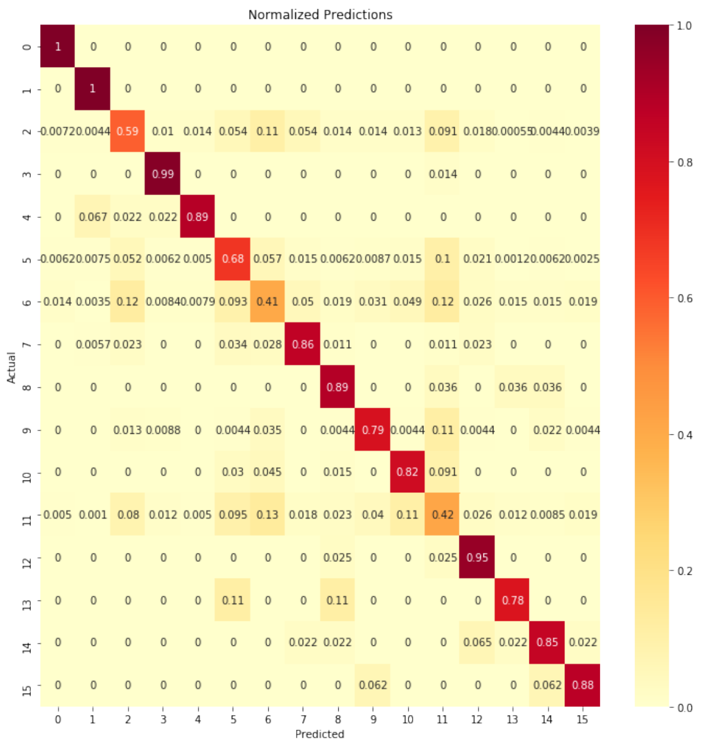

Fig. 5. Confusion matrix for test set predictions on our best-performing neural network, normalized by number of times each industry appeared.

## References

[1] Acemoglu, Daron, et al. "The network origins of aggregate fluctuations." Econometrica 80.5 (2012): 1977-2016.
[2] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, Exploring network structure, dynamics, and function using NetworkX, in Proceedings of the 7th Python in Science Conference

(SciPy2008), Gel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 1115, Aug 2008.

[3] Atalay, Enghin, et al. ”Network structure of production.” Proceedings of the National Academy of Sciences 108.13 (2011): 5199-5202.

[4] Chollet, F. (2015) keras, GitHub. https://github.com/fchollet/keras

[5] Elior Cohen, Implementation of node2vec algorithm. https://github.com/eliorc/node2vec.

[6] Cui, Yin, et al. ”Class-Balanced Loss Based on Effective Number of Samples.” arXiv preprint arXiv:1901.05555 (2019).

[7] Elliott, Matthew, Benjamin Golub, and Matthew O. Jackson. ”Financial networks and contagion.” American Economic Review 104.10 (2014): 3115-53.

[8] Fernando Prez and Brian E. Granger. IPython: A System for Interactive Scientific Computing, Computing in Science Engineering, 9, 21-29 (2007), DOI:10.1109/MCSE.2007.53 (publisher link)

[9] Grover, Aditya, and Jure Leskovec. ”node2vec: Scalable feature learning for networks.” Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[10] Hamilton, William L., Rex Ying, and Jure Leskovec. ”Representation learning on graphs: Methods and applications.” arXiv preprint arXiv:1709.05584 (2017).

[11] Hidalgo, Csar A., and Ricardo Hausmann. ”The building blocks of economic complexity.” Proceedings of the national academy of sciences 106.26 (2009): 10570-10575.

[12] Jackson, Matthew O. Social and economic networks. Princeton university press, 2010.

[13] J. D. Hunter, ”Matplotlib: A 2D Graphics Environment”, Computing in Science Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[14] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[15] Sokolova, Marina, and Guy Lapalme. ”A systematic analysis of performance measures for classification tasks.” Information Processing Management 45.4 (2009): 427-437.

[16] Stfan van der Walt, S. Chris Colbert and Gal Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37 (publisher link)

[17] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010) (publisher link)