# CLASSIFICATION OF ASSETS AND INVESTORS IN A FINANCIAL BIPARTITE GRAPH

**Guillermo Bescos**
Stanford University
gbescos@stanford.edu

**Axel Moyal**
Stanford University
axmoyal@stanford.edu

**Pablo Veyrat**
Stanford University
pveyrat@stanford.edu

December 11, 2019

## 1 Introduction And Objectives

Institutional investors are required by law to disclose their investments every quarter in 13F forms filed with the SEC. This disclosure provides a gigantic amount of information regarding interactions between investors and assets. As in e-commerce, these interactions can be modeled by a temporal weighted bipartite graph where nodes are linked by the capital invested. If the structure of e-commerce bipartite graphs has been widely studied to produce recommendations or assess the accuracy of reviews, studying a network of financial investors and their assets as a graph is far less common. Tackling financial data with a graph-centered approach could be particularly original and innovative for investment management. It could answer a real need faced by finance professionals: knowing which stock will soar in the future or in which company Warren Buffet will invest next are questions that are likely to interest many of them. Interestingly, these problems can be easily reformulated in a graph theoretical context: the first resembles a classification problem while the latter can be reformulated as link prediction. Given the complex nature of financial data which depends on a lot of exogenous factors like geopolitical conflicts which can affect stock prices, it may be harder to solve these classification and link prediction problems in finance than in e-commerce.

In this paper, we combine the structural data of a financial graph with a wide amount of possible node and edge features to perform node classification. Our goal is to identify profitable companies to invest as well as talented fund managers that are worth following.

## 2 Related Work

The use of graphs in quantitative investment is not entirely novel. For instance, in the context of Markovitz's Modern Portfolio Theory [1], L. Stankovic and al. ([2]) performed spectral clustering to detect clusters of highly-correlated stocks and develop an investment strategy to split wealth across those. Yet, this kind of initiatives to adopt a graph-theoretic mindset in finance remains scarce. There have been really few attempts to adapt the new cutting edge methods in node classification to finance. Several algorithms developed for other subjects than finance could have real applications in this field.

**The REV2 algorithm** created by Kumar and al. (in [4]) is a state-of-the-art solution in the iterative classification field. Its main goal is to identify "good" and "bad" users in e-commerce websites. By good, the authors mean fair users who make reliable ratings which reflect the goodness of the product they are evaluating. While this algorithm helped identify many fraudsters in e-commerce websites, it could have wider applications in finance where it may help to identify talented portfolio managers by opposition to incompetent ones. One limit of the REV2 algorithm is that it is hard to incorporate external node features to the method: it just takes into account the structure of the graph. In a financial context, more than knowing who invested in which company, it might be interesting to use other information such as the financial results of the company, its growth, its EBITDA, as well as the portfolio size of the investor.

In graph analysis, many methods have also been developed to take into account both the graph structure as well as external nodes and edge features to perform classification tasks. The way some of these methods operate is through node embedding. It consists in mapping every node of a graph and their features into a finite euclidean space and then running classification algorithms on the nodes' images. **GraphSAGE** (in [11]) relies on neural networks to generate node embeddings based on each node local neighborhoods and features. Like in any graph convolutional networks, nodes aggregate messages from their neighbors using neural networks. In GraphSAGE, the aggregation step is a generalized neighborhood aggregation, contrary to basic graph convolutional networks where neighbors' messages are averaged.

**Graph Attention Networks** (in [15]) is another type of method that uses graph neural networks to get node embeddings. Contrary to GraphSAGE, in this method, at each layer, during the aggregation phase, not all neighbors are considered equally important. The algorithm learns arbitrary importances to the different neighbors of each node in the graph. In a finance graph, investors may have portfolio sizes with very different orders of magnitude. To this extent, they may not all be equally important. A method like GAT which learns the importance of each investor (node) could outperform all other approaches cited above.

In this project, we apply the classification methods described above and tune them to make them more efficient with our financial bipartite graphs.

# 3 Dataset

## 3.1 Description

The dataset has been suggested by Arnab Chakrabarti. It comes from the Sharadar company and has been provided by Quandl. This dataset consists of several different tables. The main table we use is called **SF3: Core US Institutional Investors**. It gathers all the info about 6442 institutional investors and their various investments in 13434 companies for the last 26 quarters (from June 2013 to September 2019). Each line in this table represents an investment from an institutional investor to a company in a given quarter.

Since we have way less data for the last quarter (September 2019) than for the others, we do not to keep this quarter in our studies. While many different types of investments are present in our data, for simplicity reasons, we limit ourselves to stocks, excluding all other types of securities. As the main SF3 file is too heavy, we also split it in 26 different ones, in order to have one file per quarter.

To define edge weights, node features and outputs, we aggregate and join data from different Quandl tables. In particular, a table named **DAILY**, gives daily metrics about companies that serve as features for stock nodes. Another table **SEP** contains equity prices that we use to calculate log-returns for our companies. Lastly, a table called **SF1: Core US Fundamentals** provides companies' financial ratios for different periods and thus other features to use for stock nodes.

## 3.2 Pre-Processing

Even if the data is supposed to be reliable, it still needs to be cleaned and pre-processed. We pre-process the table SF3 with the Pandas library. In graph libraries such as SNAP, nodes are identified by an integer. So we first attribute a unique ID to each investor and asset. There are some absurd and irregular values in the columns. For example, in the filings of a same quarter, not all investors declare the same price for a similar investment. We normalize the prices by taking for each ticker and for each quarter the median of the prices observed for the same ticker during the concerned quarter. We use a robust aggregation method such as the median because there may be deviations and outliers (price declared equal to 0 or a price which is abnormally too high) that make the use of the max, min, and mean functions irrelevant.

Even after normalizing the prices, there are some tickers which have a price equal to zero at some periods: we filter them out. Last, there are some occurrences of missing data in our tables: we remove all the rows with missing data.

## 3.3 Graph Modeling

The information in our main SF3 table can be represented as a weighted bipartite graph with edges between investors (like for instance Berkshire Hathaway or the Vanguard Group) and their investments (Microsoft, Apple, Wells Fargo, General Motors). In this weighted bipartite graph, each edge between an investor and its investment has a weight equal to the weight of the investment in the investor's portfolio. For example, if an investor I has just two investments (2M of stock ticker A and 3M of stock ticker B), then the edge weights for the edges $I \to A$ and $I \to B$ are $0.4$ and $0.6$ respectively. Note that the weights are calculated based on dollar amounts with normalized prices. So, the sum of weights of the out-edges of an investor is $1.0$. Because investments are disclosed every quarter, the bipartite graph we work on evolves over time. In particular, the weights of the edges between investors and assets depend on the quarter considered.

As detailed in introduction, the goal of this project is to identify stocks in which professional investors should invest. The state of the investments among institutional investors at a time $t$ does not give a lot of signals about the trends, the market dynamics and the evolution of investors' portfolio. Studying only a graph with such structure would provide way less insights than a common analysis of equity metrics, or of stocks' log-returns from one quarter to another. One relevant signal to perform our classification tasks is whether an investor has sold or bought an asset in the current time-frame, or whether the value of this asset has increased to the point that it represents a higher share in the investor portfolio. To incorporate such signals in the graph structure, we work on $\Delta$-graphs where the nodes are the same as in the type of graphs described above. In these graphs, edge weights are equal to the difference between the respective edge weights from the graphs of quarters $t$ and $t-1$. To use for our node features, we also computed a table of log-returns for each ticker and for each period $(t-1, t)$.

## 3.4 Data Analysis: Visualization and Statistics

From the log-scale curves (Figure 1), we can see that the number of nodes in the distribution decreases with the degree until converging to $0$. Its power-law shape is similar to that of the graphs observed in class, especially the social networks graphs.

Figure 1: Out-Degree and In-Degree distribution of the Graph from the filings of June 2019

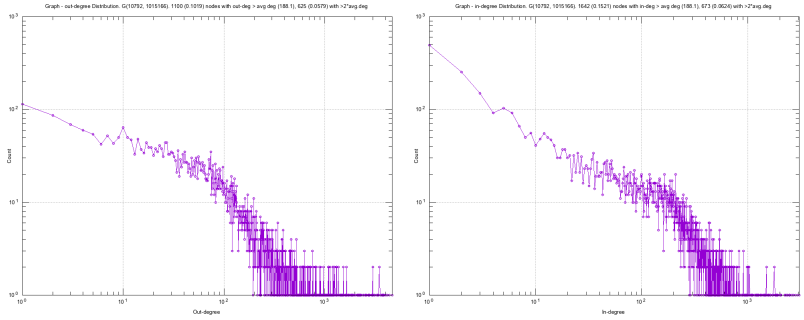
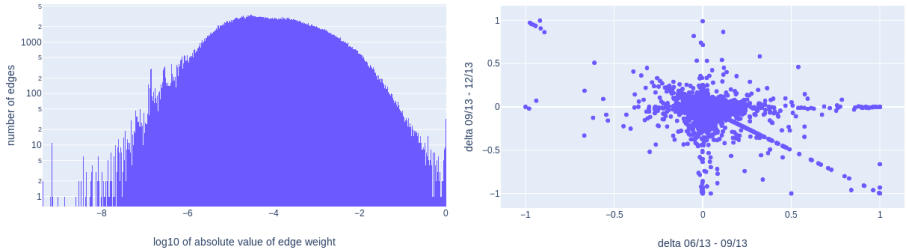


Figure 2: Key Metrics About the Data

| Metric | June 2019 | All the Quarters | March-June 2019 | All the Deltas |
|---|---|---|---|---|
| Avg In-Degree (Stock Nodes) | 169.6 | 142.3 | 186.6 | 125.2 |
| Avg Out-Degree (Investor Nodes) | 211.2 | 221.9 | 237.2 | 257.8 |
| Avg Edge Weight | $4.74e-03$ | $4.50e-03$ | $-4.62e-05$ | $5.98e-05$ |

Figure 3: $\Delta$-Edge Weight Distribution of a $\Delta$-Graph | $\Delta_{t+1}$-Edge Weight vs $\Delta_t$-Edge Weight




Due to the normalization operations made when pre-processing, the average in-degree in $\Delta$-graphs differs from the average in-degree of quarter graphs (Figure 2). We also note that the average edge weight in $\Delta$-graphs is very small compared to the edge weight in snapshot graphs.

The $\Delta$-edge weight distribution in the $\Delta$-graph from March to June 2019 (Figure 3) confirms this trend. One of the peaks of the distribution is at $10^{-4}$ which corresponds to small re-balancing transactions. There is also a peak at around 1, which corresponds to investors with undiversified portfolios.

For a pair of investor/stock represented by a point, the right plot in Figure 3 shows the $\Delta$-edge weights at quarter $t+1$ (y-axis) as a function of the weight at quarter $t$ (x-axis). It highlights the investors' changes of attitudes towards an asset. This graph allows to distinguish three categories of investors. The large majority of investors is around the point $(0,0)$. It means that they are very diversified and that they do not change the distribution of their portfolio drastically. Other groups are along the $x$-axis and the $y$-axis. For those on the $x$-axis, it corresponds to investors that bought a certain amount of the stock and held onto it for the whole period. Finally, many investors are along the line $y=-x$. It means that they bought a stock, but liquidated their position during the period that followed. Such a classification of investors could prove useful to solve one of the challenges of our approach: for investors with high share turnover, the quarterly snapshot of their stocks does not provide any information of their real activity. If we were able to detect them by using the distribution of $\Delta$-weights, we could remove them and work on the remaining groups of investors.

## 4 Iterative Methods

Can we use the collective knowledge of our roughly 6000 investors to find out well-performing stocks? In our review of recent papers, we were interested in techniques currently used in e-commerce graphs, particularly concerning product reviews. Indeed, in our problem each $\Delta$-edge weight can be seen as a rating of a certain stock. A positive (negative) $\Delta$-edge weight for a pair investor/stock means the stock represents a greater (lower) part of their portfolio at time $t$ than at time $t-1$. This is identified as a good (bad) rating of the stock. The assumption here is that investors hold on to stocks that they think are going to soar and sell those that they think are going to crash. For this part, the word "user" will denote an investor and "product" will refer to the investments.

### 4.1 Model Evaluation

#### 4.1.1 Kendall $\tau$

In this part, we try to build a time changing "goodness" score $G(p,t)$ for each product $p$. Because we are interested in predicting future returns, we look at the following statistics based on Kendall's $\tau$.

We choose Kendall $\tau$ as a measure of correlation instead of the traditional Pearson correlation based on covariance because it is more robust to outliers:

$$\hat{\tau} = \tau(G(p,t), L^{(p)}_{t-1\to t})$$

where $G(p,t)$ is the goodness score calculated from $\Delta$-graph $(t-1,t)$ and $L^{(p)}_{t-1\to t}$ is the log-return during period $(t-1,t)$. And:

$$\hat{\tau}^{+} = \tau(G(p,t), L^{(p)}_{t\to t+1})$$

Both are calculated by averaging over all the products and all the available time periods. Intuitively $\hat{\tau}$ is the in-sample correlation and $\hat{\tau}^{+}$ is the out-of-sample correlation. If we were to apply our results to create a trading-strategy, only the second one would be relevant.

#### 4.1.2 Accuracy

Another metric we use is the accuracy when classifying stocks and investors. To compute it, we have to define labels for our nodes. Since we are interested in predicting future performance, for the labels of a period $(t-1,t)$, we consider the profitabilities of the period $(t,t+1)$. At each period $(t-1,t)$, for stocks, we assign a label 1 to nodes which have a positive profitability in the following period.

While the profitability of a stock can easily be computed from its price after adjusting for dividends, it is harder to define the profitability of an investor based on our data. Actually, we do not know with our dataset what investors do between two successive quarters. Short term investors' portfolio might change drastically. To this extent, we make a very strong simplistic assumption and consider that investors update their portfolio only every quarter. With this hypothesis, we define investors' profitability as $< x_t, p_{t+1}/p_t >$ with $x = (x_1, ...x_n)$ the investment weights ($1 = \sum_i x_i$) and $p = (p_1, ..., p_n)$ the vector of stocks prices at each instant. With that in mind, to label investors at a period $(t-1,t)$, we split them depending on whether they achieve a profitability over one (earned money) in the period that follows.

One problem has been that depending on the period considered, our data ends up being sometimes really unbalanced. For example, we notice that during crisis events in which we expect most stocks to have negative returns and thus investors to lose money like in the period between September and December 2018, we can have more than 95% of our nodes labeled 0. Unbalanced data can cause problems because machine learning and iterative models often give too much importance to the dominant category. In such periods, our model could for instance achieve a 95% accuracy just by predicting 0s. In order to solve this issue and have balanced data we compute the median returns and profitabilities at each period and give the label 1 to stocks which performed better than the median of stocks' returns and to investors which were more profitable than $50\%$ of the investors during the following period.

The accuracy of a binary classifier is then:

$$a = \frac{TP + TN}{P + N}$$

where TP is the number of true positives, TN is the number of true negatives, P is the number of positives and N is the number of negatives.

Here, we consider the binary classifier $\mathbb{1}_{G(p)>0}$ for products and $\mathbb{1}_{F(u)>0}$ for users where $F$ is the fairness score that we introduce below.

### 4.2 Baseline performance

For a baseline approach, we consider aggregating the ratings of all the users into the "goodness" of the product. That is:

$$G(p) = \frac{\sum_{u\in In(p)} w(u,p)}{|In(p)|}$$

We observe a high correlation $\hat{\tau} = 0.45$) between log-returns during $(t-1,t)$ and the goodness during the same period. Intuitively, it shows that investors tend to hold on to their winners and sell their losers. This is actually a pretty interesting result, that somehow seems to go against the traditional finance adage : "Sell high, buy low". It is also possible that the causal link is actually reversed and the stocks drop or soar according to the behavior of the users.

However, the correlation drops to $\hat{\tau}^{+} = -0.002$ if we compare future log-returns with the goodness of stocks. We cannot expect any significant accuracy with such a low correlation.

### 4.3 REV2 algorithm

The REV2 algorithm brings another piece into the puzzle. Designed to detect fraud in e-commerce ratings, this algorithm innovates by outputting, in addition to goodness for each product, a "fairness" score that characterizes how reliable are a user's ratings as well as a "reliability" score for every edge. In its simplest version, these are calculated iteratively by application of the following formulas until convergence.

$$F(u) = \frac{\sum_{p\in Out(u)} R(u,p)}{|Out(u)|}$$

$$R(u,p) = \frac{\gamma_1 F(u) + \gamma_2(1 - \frac{|w(u,p) - G(p)|}{2})}{\gamma_1 + \gamma_2}$$

$$G(p) = \frac{\sum_{u\in In(p)} R(u,p)w(u,p)}{|In(p)|}$$

where $\gamma_1$ and $\gamma_2$ are parameters. We run this algorithm with $\gamma_1 = \gamma_2 = 1$ and with the initial values:

$$F(u) = 0.5, R(u,p) = 0.5, G(p) = 0$$

However, we realize that the final goodness scores of the raw code are really similar to the baseline. Here are our conclusions:

- We need to initialize properly the nodes. If at the first iteration we trust each user equally, we cannot expect to do much better than the baseline. Perhaps we can initialize the "fairness" scores according to how reliable the ratings of the users in the past have been.
- REV2 is tailored to the needs of fraud detection. It focuses on detecting users whose behaviour is radically different from the majority of the users. This framework does not seem to translate properly to our node classification problem. An investor that disagrees with the crowd might actually be valuable in our case.

### 4.4 Iterative Voting Algorithm (IVA)

We introduce some modifications to REV2. To keep things simple, we get rid of the reliability rating, considering that any review from a user is as reliable as the user is fair. We define $F(u,t)$, $G(p,t)$ (fairness and goodness at time $t$) as:

$$F(u,0) = 0 \text{ (neutral prior)}$$
$$f(u,t) = \sum_{p \in Out(u)} w(u,p,t-1) L^{(p)}_{t-1 \to t}$$
$$F(u,t) = \alpha F(u,t-1) + (1-\alpha) f(u,t)$$

In other words, $F(u,t)$ contains an exponential moving average of the user $u$'s success at predicting good products. We define the goodness of a product by averaging the product of ratings and fairness of a subset of users $\mathcal{U}(t)$:

$$G(p,t) = \frac{\sum_{u \in In(p) \cap \mathcal{U}(t)} w(u,p,t) F(u,t)}{|In(p) \cap \mathcal{U}(t)|}$$

We end up with an architecture close to an RNN with $F$ as the hidden state. However, instead of training the neural network to get the weights, we choose them in advance based on domain knowledge.

Figure 4: Fairness distribution at the last period

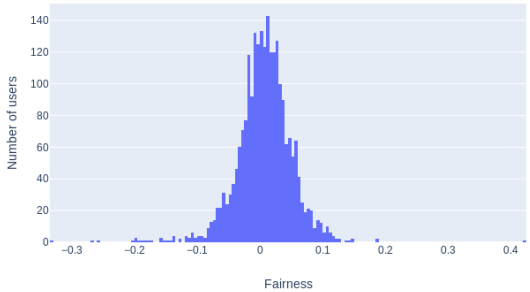
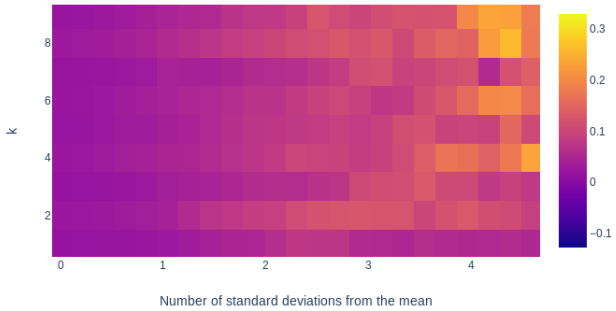



Figure 5: Fairness autocorrelation for different values of $(k, \beta)$




### 4.5 Statistical Significance of Fairness

How predictive of their future performance is the track record of investors? For our algorithm to work, that needs to be the case. In this part, we develop statistical tools to put test that hypothesis.

In the Efficient Market Hypothesis, the markets are unpredictable, therefore the performance of actors should be the same as of monkeys that choose stocks at random. However if we take a look at the distribution of final fairness we observe that the distribution is extremely fat tailed (Figure 4). We obtain 14 values at over 4 standard deviations from the mean (less than 1 for a Gaussian with the same standard deviation). This encourages us to think that these distribution outliers might have a statistical edge over other actors. In other words, even if most of the players might be trading randomly, there is a minority that can be detected and that actually knows how to play the game. We define

$$\mathcal{U}_\beta(t) = \{u \text{ s.t. } \frac{|F(u,t) - \bar{F}(u,t)|}{\sigma_t(F(u,t))} > \beta\}$$

This introduces a non-linearity in the model. The other important parameter involved in calculating fairness is $0 < \alpha < 1$. It can be rewritten as $\alpha = e^{-\frac{1}{k}}$ where $k$ is positive real number. Intuitively, $k$ can be seen as the length of the temporal window in the exponentially weighted average.

We plot $\tau(F(u,t), f(u,t+1))$ during the first 50% percent of time periods for different values of $(k, \beta)$ (Figure 5). As it can be seen, $\tau$ is positive across all parameters, but becomes more important as $\beta$ and $k$ increase, reaching 0.3. We also test against the hypothesis of 0 Kendall $\tau$ by using that under the null of no correlation:

$$\tau \sim \mathcal{N}(0, \frac{2(2n+5)}{9(n^2-n)}) \text{ where } n \text{ is the sample size}$$

At a 95% threshold, the autocorrelation is significant, which encourages to use the algorithm to classify product nodes.

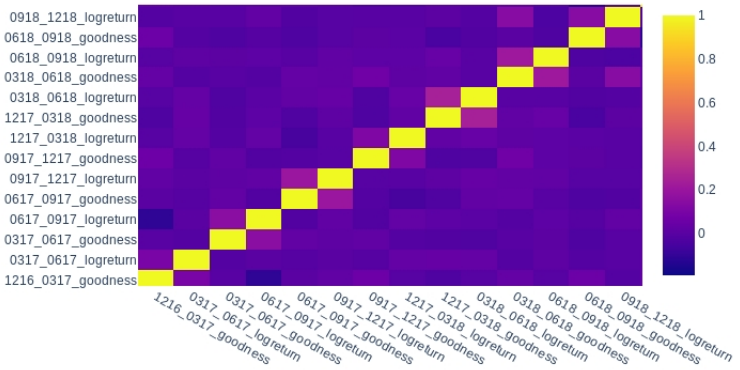
It might seem surprising that we choose a high $\beta$, therefore only taking into account the opinions of a small number of users. However, because of the high outgoing degree of users, we find that the wide majority of products have available ratings from the subset of users $\mathcal{U}_\beta$.

### 4.6 Results

Figure 6: Accuracy Results with Iterative Methods

| $\beta = 4, k = 8$ | 12/16→ 03/17 | 03/17→ 06/17 | 06/17→ 09/17 | 09/17→ 12/17 | Avg (23 $\Delta$-Graphs) |
|---|---|---|---|---|---|
| Accuracy | 0.544 | 0.606 | 0.712 | 0.582 | **0.581** |
| $\tau^+$ | 0.09 | 0.15 | 0.20 | 0.11 | **0.16** |

Figure 7: Correlation heatmap of goodness and log-returns across different periods




We apply the algorithm above to our dataset with hyperparameters $k = 8$, $\beta = 4$ and $\mathcal{U}(t) = \mathcal{U}_\beta(t)$. In other words, for each period we aggregate the reviews of the most trustworthy users to predict future returns. The assessment metrics are displayed in the table Figure 6 above. Figure 7 shows the correlation between goodness and future returns at different periods. The only significant non-zero correlations are in the diagonal (because any variable is perfectly correlated to itself) and in the cells on the superdiagonal and the subdiagonal that correspond to $\hat{\tau}_+$. We can see that across all the periods, $\hat{\tau}_+$ is positive.

These results are promising. Because our classes are of same size, any value statistically greater than $0.5$, the accuracy we would get with a random classification model, provides a good classifier.

These accuracy results can be put into perspective. In finance, the order of magnitude for accuracy metrics are generally quite low compared with other fields like computer vision. In order to understand how much money an accuracy increase represent, let us consider the following simplistic model in which there are only two types of stocks: good stocks and bad stocks. Good stocks have the same return $r_1$ and bad stocks the same return $r_2$ with $r_1 > r_2$. We note $a$ the accuracy achieved with our model. To this extent, if we consider the market as a random investor, we get with our model a return superior to the market:

$$(a-0.5)(1+r_1)-(a-0.5)(1+r_2) = (a-0.5)(r_1-r_2)$$

With for example the average accuracy of $0.58$ as the one we get on average, $r_1 = 10\%$ and $r_2 = 0\%$, the model would be $a = 0.07 \times 0.1 = 0.8\%$ better than the market in general. This is not negligible at all in terms of returns.

### 4.7 Statistical Test for the results

In order to check the robustness of the results for one given quarter, we can compute a confidence interval. The probability $p$ that the algorithm labels accurately one node can be modeled by a Bernoulli law $B(p)$. As well, $p$ can be interpreted as the true accuracy of our model. Consequently, we define the following normal variable

$$Z = \frac{\sqrt{n} \times (p_n - p)}{\sqrt{p_n \times (1 - p_n)}}$$

With $p_n$ the accuracy observed which corresponds to the empirical probability of true labelling. The confidence interval at the $95\%$ level for the accuracy is then:

$$[p_n - 1.96 \times \frac{\sqrt{p_n \times (1-p_n)}}{\sqrt{N}}, p_n + 1.96 \times \frac{\sqrt{p_n \times (1-p_n)}}{\sqrt{N}}]$$

$N$ corresponds of the total number of investors and investments for the period considered.

If we compute the values for one given quarter, we get that the accuracy belongs to $[p_n - 1\%, p_n + 1\%]$. However, we cannot use this approach for the overall accuracy because depending on the quarter the probability of true labelling is different. The confidence interval would be higher. In the end, the confidence intervals we get confirm that the accuracy values are significantly higher than $0.5$, the accuracy of the baseline random model.

# 5 Graph Neural Networks

Graph neural networks can also be used to perform supervised node classification on our bipartite graphs. We rely here on different graph neural networks methods that we adapt to our specific case of a weighted-bipartite graph.

## 5.1 Inputs

As explained above, what matters in this classification framework is whether an investor buys or sells a stock and not what quantity of assets he holds. Similarly to iterative methods, the inputs for our graph neural networks are $\Delta$-graphs, where edges correspond to changes of proportion of capital injected by the out-node in the in-node during the given periods. For our neural networks to work efficiently, we consider these $\Delta$-graphs as undirected.

In our graph neural networks, the two types of nodes of our bipartite graphs (stocks and investors) need to be distinguished. To do so, we use different features and thus different initial embeddings for stocks and investors.

### 5.1.1 Node Features

For the features of stock nodes, we use the following financial ratios: diluted earning per share, return on equity, enterprise value over EBITDA, price to earnings ratio, sales per share, price to sales ratio, price to book value, debt to equity ratio, dividend yield, free cash flow per share, and log returns.

We willingly take only intensive ratios because performance should not depend on the company size. Intensive ratios have the advantage to be homogeneous which make them eligible to be compared across companies of different sizes.

In addition, we decide to take only few features to avoid collinearity issues between them. For example, we could include the EBIT of our companies as a feature, but since it is highly correlated with EBITDA and might add collinearity between features, we chose to keep it out. We also choose not take into account qualitative characteristics like the economic sector. With $N$ different sectors, we would have to add $N-1$ binary features. Adding such categorical features could therefore lead us to increase substantially the number of features for our nodes. This would maybe overemphasize the importance of the sector over the other intensive features we mentioned above.

Contrary to the stocks, less features are kept for investors tensors. The only features used are the portfolio size, the number of investments, as an indicator of diversification, and the profitability of the investor over the last period.

### 5.1.2 Input Tensors and Labels

From these features, we have to define input tensors to be fed in our graph neural networks model. One constraint imposed by Pytorch is that these tensors need to have the same size for both investor and stock nodes even if they do not have the same number of features. To do so, we use tensors of size $(N_{Stock} + N_{Investors}, 1)$, where $N_{Stock}$,$N_{Investors}$ represent the number of stock features and investor features ($Features_{Stocks}$, $Features_{Investors}$).

In this approach, we represent investor nodes' features vectors by the tensors $[0] \times N_{Stock} + [Features_{Investors}]$ and stock nodes' features vectors by $[Features_{Stocks}] + [0] * N_{Investors}$. With such a definition, at every layer of our graph neural networks, nodes' embeddings keep the same size, regardless of the nature of the nodes.

Since we perform here supervised node classification, we also define node labels. We use the same labels as in the iterative methods we discussed earlier.

## 5.2 Network Architecture

The objective here is classifying lucrative investments and successful investors. Obtaining nodes embeddings at several layers is not sufficient for this task. This is the reason why we use the embeddings we get with our GNN models as new features for classification. What we end up doing is to first use a neural network architecture similar to GraphSAGE or GAT to generate embeddings followed by a softmax multi-linear perceptron to classify those embeddings.

In Pytorch, we use a cross entropy loss $L$ for both investors and stocks:

$$L = -\sum_{c \in M_I} \sum_{i \in I} y_{i,c} \times log(p_{i,c}) - \sum_{c \in M_S} \sum_{i \in S} y_{i,c} \times log(p_{i,c})$$

where the $M_I$ are the different classes of investors and $M_S$ the classes of stocks. $p_{i,c}$ defines the probability of the node $i$ to be in the class $c$. It is important to remember here that, although we do not have to modify Pytorch's implementation of the loss function to distinguish stocks and investors because their input tensors have the same shape, labels were defined differently for our two types of nodes.

### 5.2.1 GraphSAGE

The first network architecture we use is GraphSAGE. Contrary to what was done in the PSET2, we have to take into account the edge weights in the messages that are passed at each layer to a node from its neighbors. To implement these steps, we use the Pytorch geometric paradigm: message passing, propagation and update.

Since our graph is perfectly bipartite, with no investors (or nodes) being connected together, it is crucial to note that investors embeddings are iteratively computed from the investments embeddings and vice versa. We note $I_v^k$ the embedding tensor at layer k of the investor defined by the node $v$ and $S_u^k$ the embedding tensor at layer k of the stock defined by the node $u$. At every layer, we apply the

following operations:

$$\mathbf{S^k_{N(v)}} = AGG_k(\mathbf{S^{k-1}_u}, u \in N(v))$$

$$\mathbf{I^k_{N(v)}} = AGG_k(\mathbf{I^{k-1}_u}, u \in N(v))$$

For $AGG_k$, the aggregation function of neighbors embeddings, we use an expression with the following form, depending on the weights $weight(u, v)$ between the considered node $v$ and its neighbors $u$:

$$AGG_k(v) = \sigma(\sum_{u \in N(v)} \mathbf{h^{k-1}_v} \times weight(u, v))$$

with $\mathbf{h^{k-1}_v}$ the previous layer embeddings of the neighboring investments or stocks.

Next, we compute:

$$\mathbf{I^k_v} = \sigma(\mathbf{W_k} * CONCAT(\mathbf{I^{k-1}_v}, \mathbf{S^k_{N(v)}}))$$

$$\mathbf{S^k_v} = \sigma(\mathbf{W_k} * CONCAT(\mathbf{S^{k-1}_v}, \mathbf{I^k_{N(v)}}))$$

#### 5.2.2 GAT

Another method we use is Graph Attention Networks. Similarly to GraphSAGE, we slightly have to change the initial implementation made in class to take into account the fact that edges are weighted. In particular, using the form of attention mechanism from the PSET2, we apply at every layer the following operations:

$$\mathbf{I^k_v} = \sigma(\sum_{u \in N(v)} weight(u, v)\alpha_{vu}\mathbf{W_k}\mathbf{S^{k-1}_u})$$

$$\mathbf{S^k_v} = \sigma(\sum_{u \in N(v)} weight(u, v)\alpha_{vu}\mathbf{W_k}\mathbf{I^{k-1}_u})$$

where $\alpha$ is the attention mechanism defined in the PSET2.

### 5.3 Training and Testing Methodology

From these graph neural network models, we carry out several experiments on different types of training sets.

#### 5.3.1 Nodes Mask

Firstly, we work on single $\Delta$-graphs for the training and the testing of our algorithms. In this approach, for each $\Delta$-graph, we randomly mask $20\%$ of the nodes of the graph. These masked nodes are ignored by the neural networks during the training phase. The accuracy is then measured based on the labels predicted for masked nodes.

This approach is however not perfectly relevant for what we are trying to achieve. Our goal is from a single $\Delta$-graph to be able to predict which investors and stocks are likely to be among the $50\%$ most profitable ones in the period that follows. This mask approach is easier to implement in Pytorch. It is as if in a given period we knew the future profitabilities of almost all the stocks and investors except a few of them. We then train the model on the labelled nodes to predict how these unknown assets and investors will perform.

This methodology remains helpful as a sanity test to evaluate how good our models can be at predicting and to see if our features are relevant.

#### 5.3.2 Two periods Train/Test

The other approach we adopt is to train our models on one $\Delta$-graph and to use as a test set the $\Delta$-graph from the following period. Even if two consecutive graphs may have slightly different structures, in graph neural networks the same aggregation parameters are shared for all nodes: we can therefore generalize models to entirely unseen graphs. For instance, we use the $\Delta$-graph for the periods between December 2016 and March 2017 as a training set for the prediction on what happened between March 2017 and June 2017 (the test set). This methodology is more in line with our objective to predict the future prices of stocks.

### 5.4 Results

After tuning our hyperparameters, we manage to evaluate the accuracies of our different models. Among other things, we notice that our algorithms converge quite rapidly and set the number of epochs to $100$. We also set a $0.0008$ learning rate, a $0.5$ dropout and the number of layers to $2$.

The accuracies measured on the two types of train/test methods and the two types of networks for the $\Delta$-periods of the year 2017 can be found in Figure 8. For each period, in the approach in which we train on one graph and test on the other graph, the train set has been the graph of the preceding $\Delta$-period. For example, in the "Two Periods" value for the column $03/17 \rightarrow 06/17$, the train set has been the graph of the $\Delta$-period between December 2016 and March 2017.

Figure 8 also shows the average accuracy of each model, computed as the mean of the accuracies we obtained in the 23 $\Delta$-periods ranging from June 2013/September 2013 to December 2018/March 2018.

Figure 8: Accuracy Results for Graph Neural Networks

| Accuracy | 12/16→ 03/17 | 03/17→ 06/17 | 06/17→ 09/17 | 09/17→ 12/17 | Avg (23 $\Delta$-Graphs) |
|---|---|---|---|---|---|
| GraphSAGE (Mask) | 0.597 | 0.675 | 0.636 | 0.714 | 0.624 |
| GAT (Mask) | 0.627 | 0.593 | 0.582 | 0.615 | 0.597 |
| **GraphSAGE (Two Periods)** | 0.614 | 0.613 | 0.533 | 0.485 | **0.556** |
| **GAT (Two Periods)** | 0.582 | 0.435 | 0.554 | 0.389 | **0.531** |

The values we obtain are on average superior to $0.5$. Since in each of our graphs, we tuned our choice of labels to have exactly $50\%$ of 1 and 0, this suggests that our models perform better than a random choice for the prediction of labels.

With the "Mask" methodology, we get accuracies around $0.6$. These results are higher than when we train our model on one $\Delta$-graph and test it on the following one. This was somehow expected because in this "Mask" approach we are working on only one graph from one period to predict. Even though this approach would be useless in the real world for investors looking for stocks to invest in, it supports the coherence of the models we built. It shows that by studying our financial data as a weighted-graph, we manage to leverage some correlations between the behaviours of different investors and to capture some of the market signals these investors send when they reallocate their portfolio across different stocks between two periods of time.

For both methodologies, the results from GraphSAGE seem to be better in most of the cases than those from GAT. One reason for this may be due to the fact that GAT tries to learn importances to the different neighbors of each node, importances which are already specified by the edge weights of our graphs. It increases the number of parameters and can lead to overfitting.

With our potentially usable train/test approach on two different periods, on average over all our periods, the accuracy we measure with GraphSAGE is $0.556$ and with GAT $0.531$. There are still occurrences in which we do not perform as well as a what a random model would do. To this extent, these models cannot be applied yet to a real portfolio building context.

Concerning the accuracy obtained in single quarters, the confidence intervals are the same than with our iterative methods $[p_n - 1\%, p_n + 1\%]$. This confirms that in most periods the accuracies we get are significantly superior to 0.5.

With the financial two state model described in the iterative method part, we saw that an average accuracy of $0.55$ for a financial was not negligible. It generates a return which can potentially be $0.5\%$ higher than the market. However, while our graph neural network take into account several financial features, we still notice that the iterative method gives a better accuracy.

### 5.5 Discussion, Potential Improvements and Future Work Directions

There may be several reasons for the results we obtain, and there are several elements we could play on to improve these results. First of all, for our nodes features, we selected few features and did not take into account discretionary events linked to the activity of the company. With our dataset, we did not have access to many non financial factors (number of customers, market share, patents, industry trends or intensity of rivalry) which are often taken into account when valuating a company. Exogenous factors like the evolution of the interest rates are not considered as well.

In addition, in our data analysis, we managed to identify different kinds of investors depending on the frequency of their investments. In particular, we saw that there was a class of very short term investors which are less sensitive to the corporate finance data we use. These investors maybe prevent us from discerning correlations and trends with our neural networks. The failure of the very strong simplistic assumption which considered that investors update their portfolio only every quarter could explain why our models are struggling to predict future outcomes correctly. With smaller $\Delta$-periods with a range of a day or a week, our GraphSAGE and GAT models could maybe capture more of the signals sent by market actors.

Another cause for the results we obtained may have to do with the way we computed edge weights. In particular, by normalizing the weights as we did, while we thought of it as a way to capture how investors rebalance their portfolio, it lead us to imply that small investors are as important as big investors in our graphs. We thought that we could counterbalance this assumption by adding portfolio size as a node feature for investors, yet this may have not been enough. If we were to continue working on this project, it would interesting to test different approaches concerning the definition of edge weights in our graphs.

Concerning our models in themselves, the weight matrices that we train at each layer are the same for both investors and tensors. The way we made the distinction between the two types of nodes was to use input tensor vectors at layer 0 with the same shape but with completely different values for the features. This may not have been sufficient. To better distinguish stocks and investors, one solution could be to train at each layer different weight matrices $\mathbf{W_{k,1}}$ and $\mathbf{W_{k,2}}$ in order to have:

$$\mathbf{I_v^k} = \sigma(\mathbf{W_{k,1}} * CONCAT(\mathbf{I_v^{k-1}}, \mathbf{S_{N(v)}^k}))$$

$$\mathbf{S_v^k} = \sigma(\mathbf{W_{k,2}} * CONCAT(\mathbf{S_v^{k-1}}, \mathbf{I_{N(v)}^k}))$$

where $I_v$ is the investor defined by the node $v$ and $S_u$ the stock defined by the node $u$.

If we had more time, we could also try to implement the solution proposed by He and al. (in [15]) in September 2019 in which they describe Bipartite Graph Neural Network (BGNN), a novel domain-consistent model to efficiently handle bipartite graphs.

## 6 Conclusion

Our project is a unique attempt to analyze finance data with graphs. Our goal was to classify assets and investors using SEC filings.

First, the exploration of the dataset (SEC filings) and the analysis of the network structure this dataset entailed gave

us interesting results. In particular, we managed to distinguish different types of investors based on how they were rebalancing their portfolio between quarters.

After that, we have implemented two types of classification models of nodes in our dataset: an unsupervised one, with the iterative voting algorithm, and a supervised one, with Graph Neural Networks.

Even if it does not take into account external financial features, the IVA iterative method provides very encouraging and interesting results. We have seen that the Kendall's tau is relatively high and that the generated "goodness" is correlated to the profitability. Obviously, the "goodness" score cannot be used to solely predict the price of future stocks. However, thanks to the correlation, it can be used as an additional feature in statistical models. For instance, quantitative hedge funds often use regression models from diverse source of data to predict future stocks and might be interested by new features which provide information gain.

As for the Graph Neural Networks, the results are less convincing but still encouraging. On average the models we built perform better than the random model. We have found out several promising directions that could help us improve our models to make them usable by institutional investors. Graph Neural Networks have shown to be intricate models difficult to interpret. Yet, we are very delighted to have worked with these challenging algorithms. It was an opportunity to discover the Pytorch library and to improve our deep learning knowledge.

In the end, although we only focused on two types of methods to solve our unique problem of classification of investors and nodes with graphs, there are a lot of classification algorithms like probabilistic relational classifier or belief propagation which we could have used as well if we had more time.

## 7 References


[1] H. Markowitz. Portfolio Theory. *Portfolio Selection*, Journal of Finance, vol. 7, no. 1, pp. 77–91, 1952.

[2] B. Scalzo Dees, L. Stankovic, A. Constantinides, D. Mandic. A Graph-Theoretic Framework to Diversification. 2019

[3] V. Boginski, S. Butenko, P. M. Pardalos. On Structural Properties of the Market Graph. In *Innovations in Financial and Economic Networks*, A. Nagurney, Ed. Edward Elgar Publishers, pages 29–45, 2003.

[4] S. Kumar, B. Hooi, D. Makhija, I. Disha, M. Kumar, C. Faloutsos. REV2: Fraudulent User Prediction in Rating Platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* , pages 333-341. 2018

[5] J. Leskovec, A. Grover. node2vec: Scalable Feature Learning for Networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[6] N. Benchettara, R. Kanawati, C. Rouveirol. Supervised Machine Learning Applied to Link Prediction in Bipartite Social Networks. In *Advances in Social Networks Analysis and Mining (ASONAM)*, 2010 International Conference on, pages 326–330. IEEE, 2010.

[7] M. Al Hasan, V. Chaoji, M. Zaki. Link Prediction using Supervised Learning.

[8] J. Leskovec, L. Backstrom. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[9] J. Kunegis, E. De Luca, S. Albayrak. The Link Prediction Problem in Bipartite Networks.

[10] S. Kumar, F. Spezzano, V.S. Subrahmanian, C. Faloutsos. Edge Weight Prediction in Weighted Signed Networks.

[11] W. Hamilton, R. Ying, J Leskovec. Inductive Representation Learning on Large Graphs. CoRR,abs/1706.02216, 2017.

[12] B. Perozzi, R. Al-Rfou, S. Skiena. DeepWalk: Online Learning of Social Representations.

[13] T. Mikolov, K. Chen, G. Corrado, J. Dean. Efficient estimation of word representations in vector space. *CoRR, abs/*1301.3781, 2013.

[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Benguo. Graph Attention Networks. 2017.

[15] C. He, T. Xie, Y. Rong, W. Huang, Y. Li, J. Huang, X. Ren, C. Shahabi. Bipartite Graph Neural Networks for Efficient Node Representation Learning. 2019