CS224W: Weighted Signed Network Embeddings Final Report

Jacob Hoffman
Computer Science
Stanford University
Stanford, CA 94305
jacobmh@stanford.edu

Sam Premutico
Computer Science
Stanford University
Stanford, CA 94305
samprem@stanford.edu

github: https://github.com/jacobmh1177/cs224w

1 Introduction

In our project, we explore weighted signed network (WSN) embeddings. A *weighted* network is one in which edges are not all assigned some constant value, but rather are assigned some real value from a range of possible values representing either the *strength* of the edge or some other metric encoded by the weight. A *signed* network is one in which the edges are given a *sign*, i.e. positive or negative. Both signed and weighted networks are often useful in being able to model notions of *distrust* or sentiment between two entities in a network. While previous work has focused on generating node embeddings, such work has not focused on networks that are both weighted *and* signed.

In our work, we generate novel node embeddings inspired by the skip-gram model outlined in *SNE: Signed Network Embeddings* which we augment with *fairness* and *goodness* scores proposed in the papers outlined below. With these augmented node embeddings, we train multiple softmax-classifiers and regression models on a link-prediction task. Specifically, we produce embeddings for nodes in the Bitcoin OTC and Alpha exchanges, and using these embeddings, predict the signed weights of unseen edges between users which measure trust in the network. These embeddings will potentially aid in the identification of fraudulent actors in the network. In exchanges like Bitcoin OTC, where users can exchange Bitcoin for paper currency, trust is vital. In an attempt to provide a clearer insight into the trustworthiness of different nodes in the exchange, Bitcoin OTC publishes an OTC web of trust where users can leave trust ratings for other users. However, this is an imperfect solution as explained on the Bitcoin OTC website: "it is not impossible for a scammer to infiltrate the system, and then create a bunch of bogus accounts who all inter-rate each other." We aim to improve current weighted edge prediction models to potentially correctly weight these nodes as distrustful.

2 Dataset

For our work, we make use of two weighted signed trust networks built from data collected from the Bitcoin OTC and Alpha exchanges. As mentioned previously, these exchanges allow users to rate the trustworthiness of other users in the network. The weighted signed network are directed graphs where each node is a user and an edge exists between two nodes u and v if u rates vs trustworthiness. Trustworthiness ratings are on a scale from -10 to 10 (excluding 0).[1] As discussed in section five, the edges are relatively skewed in both dataset towards weights very near 0, leading to a very high proportion of labels belonging to one of the 6 classes in the 6-class softmax classification task we perform. We visualize the distribution of fairness and goodness scores, as well as the embeddings we generate, over the two datasets in section five.

[1]These two datasets can be found at https://cs.stanford.edu/~srijan/wsn/

3 Previous Work

Our work synthesizes and builds on work in three areas: signed network embeddings, link-prediction in weighted signed networks, and fraud detection in user-rating plaforms. We present an overview of the relevant prior literature below. First, we analyze *SNE: Signed Network Embeddings* which discusses a novel method of generating embeddings for signed networks. Next, we analyze *Edge Weight Prediction in Weighted Signed Networks*, which discusses a novel method for prediction edge weights in WSNs. We then turn to the problem of fraud detection by analyzing *REV2: Fraudulent User Prediction in Ratings Platform*. Finally, we review recent work on Link-Prediction.

3.1 Yuan et al. Signed Network Embeddings

Yuan et al. generate embeddings for nodes in Signed Networks. The algorithm the authors propose to generate these embeddings is modeled after the skip-gram algorithm commonly used to generate word-embeddings which relies on word, and in this case node, co-occurrence data. The vocabulary in the network embedding algorithm is then the vertex set V. The embedding v_{v_i} for a given vertex v_i is defined as $[v_{v_i} : v'_{v_i}]$, where the former is its source embedding and the latter is its target embedding. Thus a node embedding is composed of two distinct embeddings.

For a target node v and a path of $h = [u_1, u_2, ..., u_l, v]$ of length l, the model computes the predicted target embedding of node v by linearly combining source embeddings of all source nodes along the path h with a corresponding signed-type vectors c_i:

$$\hat{v}_h = \sum_{i=1}^{l} c_i v_{v_i}$$

where $c_i = c_+$ if the edge from the ith to the $ith + 1$ node is positive, otherwise $c_i = c_-$. The authors then compute the similarity of the predicted embedding to the actual representation of the node. In order to train node representations, the authors define the conditional likelihood of a target node v generated by a path h and their edge types q based on a softmax function. The objective function is then to maximize the log-likelihood of this conditional probability. Once these nodes embeddings are generated, the authors use logistic regression to perform various tasks.

The embeddings are tested on two tasks: link-prediction and node classification. One of the datasets the authors use is a co-editing matrix of Wikipedia articles. Each edit is labeled as reverted (due to the edit being malicious) or not-reverted (a benign edit). If user i and user j co-edit articles and the majority of these edits are malicious, then a negative edge is added between user i and user j. If user i and user j co-edit articles and the majority of these edits are benign, then a positive edge is added between user i and user j. Their experiments show that their embeddings outperform the other four embedding techniques sampled, including Node2Vec.

3.2 Srijan et al. Edge Weight Prediction in Weighted Signed Networks

Srijan et al. seek to predict edge weights in weighted signed social networks. They stress that their work was the first such attempt with real-world WSN datasets. To do this prediction, the authors define two new metrics for describing nodes *fairness* and *goodness*. Essentially, *fairness* describes how fair a node is at assessing other nodes in the network and *goodness* measures how good other nodes think this node is. The authors formulate the Fairness and Goodness Algorithm (FGA) to assign these ratings to nodes in WSNs. It is first important to note that goodness depends on fairness and vice versa:

$$g(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f(u) * W(u, v)$$

$$f(v) = 1 - \frac{1}{|out(v)|} \sum_{u \in out(v)} \frac{W(u, v) - g(v)}{R}$$

FGA Algorithm

1. Initialize all nodes fairness and goodness scores to the max value of 1.
2. While fairness and goodness scores $<$ threshold:
   (a) calculate the goodness score using the last iterations fairness score
   (b) calculate the fairness score using this iterations goodness score

To predict edge weight, the authors rely on the notion that edge weight depends on the fairness and goodness of the two nodes that define the edge. Their first experiment involves using the FGA score nodes in the network and perform Leave One-Out Edge Weight Prediction. They use the goodness score of a node as one predictive value and the product of fairness and goodness as another value. The authors discover that $F * G$ was the best predictor of all the other algorithms they tried. The second experiment they run is building a multiple regression model where they use the outputs of a wide range of algorithms to build a feature set that they then use in the regression. In this experiment, the authors discover that the most important features in their regression was often the $F * G$ feature by a large margin.

3.3 REV2: Fraudulent User Prediction in Ratings Platform

In the paper [1], the authors propose the REV2 algorithm to detect fraudulent users on user opinion platforms. To do so, the authors design an algorithm that assigns a *fairness* score to a user $F(u)$, a *goodness* score to a product $G(p)$, and a *reliability* score to a review $R(u, p)$. These measures are all interrelated, and the authors propose five axioms that describe their interdependence (which they formulate mathematically, as well):

1. Better products get higher ratings
2. Better products get more reliable positive ratings
3. Reliable ratings are closer to goodness scores
4. Reliable ratings are given by more fair users
5. Fairer users give more reliable reviews

The authors then address the matter of the *cold start problem*, referring to the difficulty in assessing the fairness of users who have displayed little activity on the network (and in assessing the quality of products with few ratings). To overcome this, the authors make use of Laplace smoothing with a set of parameters in their calculations. Next, the authors consider the behaviour of users in order to augment their scoring functions. That is, they consider things such as if a user posts many ratings in a short span of time or post ratings at set time intervals. These behavioral measures are then used to calculate a normality score for each user and product. These normality scores, when present, are then used in the initialization the fairness, goodness, and rating scores. Finally, the algorithm iteratively updates these scores using the mathematical formulation of the axioms listed above until convergence.

3.4 Link Weight Prediction with Node Embeddings

In Hou et al.'s recent publication [2], authors experimented with using node embeddings to predict edge weights in a signed and weighted network with a neural network. Their proposed architecture involved (1) a node look up layer where a given node id mapped to a specfic node vector (2) two node vector layers (one for the source node and one for the target node) whose values were updated through backpropagation (3) several fully connected layers (with ReLU activations) (4) an output later that ultimately produce the edge weight prediction. The authors compared their model's accuracy against several baseline stochastic block models including (vanilla SBM, weighted SBM [3], etc.) and found that their approach was consistently more accurate.

3.5 Weight Prediction in Complex Networks Based on Neighbor Set

Zhu et al. [4] attempt to predict edge weights in network by relying on local structural information. Their algorithm relies on their stated assumption "that the formation of link weights is regulated

by local clusterings in which homogenous links tend to have similar weights". In their case they defined a node's local structure to be it's egonet.

Given the task of estimating the weight of an edge between x and y where x and y are in the egonet of a node α:

$$w_{xy|x,y\in\Gamma(a)} = \bar{w} w_{\alpha x} w_{\alpha y}$$
$$\bar{w}_{alpha} = \frac{\sum_{m,n\in\Gamma(a)} w_{mn}\alpha_{mn} + 1}{\sum_{m,n\in\Gamma(a)} w_{\alpha m} w_{\alpha n}\alpha_{mn} + 1}$$

4 Approach

We attempt to generate node embeddings that allow us to improve signed weighted edge link-prediction, with potential applications to node classification. In order to do so, we proceed in two parts. First, we follow the fairness and goodness algorithm above to generate fairness and goodness scores for each node. Second, we modify the skip-gram inspired Signed Network Embeddings from Yuan et al to be weighted in addition to signed by incorporating the fairness scores calculated in the first part of our algorithm. In order to make our embeddings weighted and signed, we modify the embedding equation proposed by Yuan et al.:

$$v_h = \sum_{i=1}^{l} c_i * v_{vi} \tag{1}$$

so as to make c not only either c_+ or c_-, but rather a value in the range [-1,1]. This allows us to capture more subtle notions of trust and distrust between users relative to the all-or-nothing 0 or 1 weighting/signs of their current implementation. The question then becomes, how do we select the appropriate value of c for any given edge? We propose using the fairness and goodness scores generated from the fairness and goodness algorithm. We now first describe our baseline model which incorprates both edge weight and sign into the calculation of c. We then describe our motivation for incorporating fairness into the equation before doing the same for goodness.

For our baseline model, we simply set $c_i = w_{i,i+1}$, where $w_{i,i+1}$ is the weight of the edge between node v_i and v_{i+1}. Since edge weights represent the rating of one user by another, $w_{i,i+1}$ is then a signed, weighted value in the range [-1,1]. We end up with the following baseline embedding calculation:

$$v_h = \sum_{i=1}^{l} w_{i,i+1} * v_{vi} \tag{2}$$

Recall that the fairness of a user u is a measure of how fairly they rate other users. Namely, a fair user is a user who rates trustworthy users as trustworthy and fraudulent users as untrustworthy. Conversely, an unfair user is a user who rates trustworthy users as untrustworthy and fraudulent users as trustworthy. Thus the ratings given by a user with a higher fairness score intuitively should be given more weight than the ratings given by a user with a low fairness score. We accomplish just this when we modify the above algorithm to multiply the product within the sum by the fairness of user i, as we weight the edge in accordance with the fairness of the source node whose rating the edge corresponds to. We end up with the following equation:

$$v_h = \sum_{i=1}^{l} f_{v_i} * w_{i,i+1} * v_{vi} \tag{3}$$

Where f_{v_i} is the fairness of user v_i.

Finally, we incorporate goodness scores into our embedding algorithm. Specifically, we scale the embedding of the target node v_h by its goodness score. We decided upon this strategy after making the assumption: If a source node is fair its rating of the target node will be proportional to the target

node's goodness and if a source node is not fair then its rating of the target node will be *inversely* proportional to the target node's goodness. This gives us our final embedding equation:

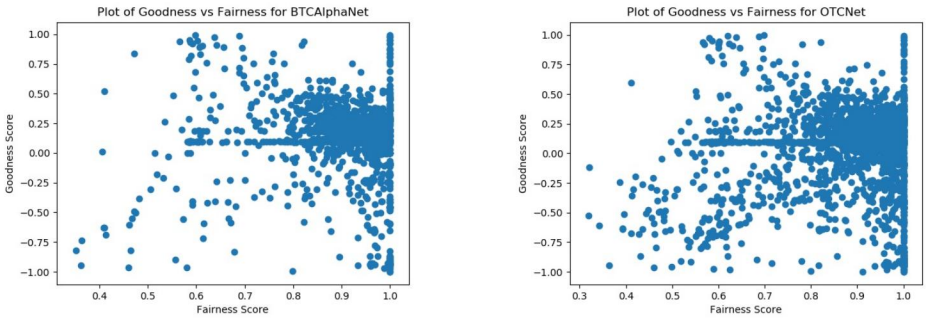
$$\boxed{v_h = g_{v_h} \sum_{i=1}^{l} f_{v_i} * w_{i,i+1} * v_{vi}} \tag{4}$$

Our implementation of the augmented heuristic scales the edge weight to be an integer in the range [0,20]. This gives a much more granular notion of sign and weight than that implemented in the original heuristic above. We then train several softmax classifiers (with the exception of KNN) on these embeddings on a 6-class classification task, where each class is a subrange of the range [-1, 1]. This is our final link-prediction task. The softmax classifiers optimize a cross entropy loss function of the form $L_i = -\log(\frac{e^{-fy_i}}{\sum_j e^{-fy_j}})$ [5]. We additionally train two regression models to predict the real-value edge-weight.

5 Experiments and Results

5.1 Experiments

We began our project by calculating fairness and goodness scores for nodes in the Alpha and OTC bitcoin networks. [6] We were curious if there was a clear relationship between the fairness and goodness scores of nodes, but as Figure 1 shows, there doesn't appear to be a strong correlation between the two scores. [2]

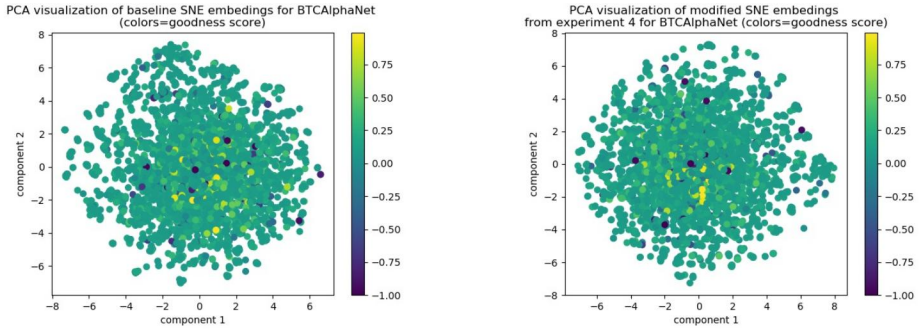


(a) AlphaNet (b) OTCNet

Figure 1: Relationship between Fairness and Goodness

Next, we made the previously discussed modifications to the Signed Network Embedding algorithm [7]. Specifically, we modified this equation: $v_h = \sum_{i=1}^{l} c_i * v_{vi}$. Instead of c_i only capturing the sign of the weighted edge, we use the actual edge weight. We then used random walks over the two networks to learn node embedddings using the modified algorithm.

[2]We modify code from `https://cs.stanford.edu/~srijan/wsn/` to calculate fairness and goodness scores



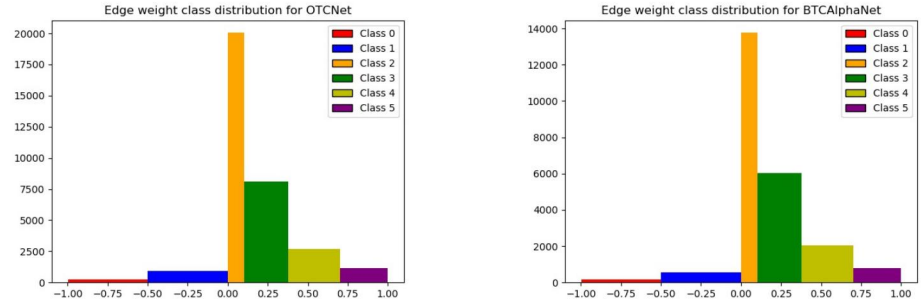
(a) AlphaNet Baseline Embeddings - Goodness

(b) AlphaNet Experiment 4 Embeddings - Goodness

Figure 2: Visualization of AlphaNet Embeddings

Figure 2, uses t-SNE dimensionality reduction to visualize the generated node embeddings for the Alpha bitcoin network. Figure 2 (a) and (b) have each node colored by their Goodness scores. We didn't expect the SNE embeddings to necessarily capture Goodness scores, and the plot generally confirm this. We also experimented with encoding the Fairness and Goodness scores to the modified SNE algorithm as discussed in section 4. Unfortunately, its appears that this particular approach was not effective in encoding goodness scores in the node embeddings.

We then conduct a series of link-prediction experiments on both the OTCNet and AlphaNet bitcoin networks, where each experiment uses one of the four embedding. We perform both a softmax classification task, where each class is an edge weight sub-range on the range [-1, 1], as well as a regression task, where we predict an edge-weight (rather than a class) in the same range. We empirically derive the number of classes as well as the cutoffs for each class by analyzing the distribution of edge weights in each graph. We found that there were six logical classes which were present with almost identical distributions in both networks (see Figure 3).



(a) Class Distribution- OTCNet

(b) Class Distribution - BTCAlphaNet

Figure 3: Graph Edge Weight Distributions

We repeat each classification experiment with each of the following models: Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), and K-Nearest Neighbors (KNN). We repeat each regression experiment using Linear Regression (LR) and Support Vector Machine (SVM) models. We end up with a total of 20 classification experiments for each network (four embeddings, five models) and eight regression experiments (four embeddings, two models). (See Appendix A for model hyperparameters.) Finally, we implement two naive models to establish baseline results for both classification and regression. Specifically, we implement a *mode* classification baseline, which simply predicts the mode of the edge weight classes seen in training for each edge at test time, as well as a *mean* regression baseline, which predicts the mean of the edge weights seen in

training for each edge at test time. Our results are listed below. In order to generate embeddings and evaluate our models, we generate five train-test splits for each network. Specifically, we randomly remove 2% of nodes in a network and generate random walks over this subgraph to learn embeddings. We then test each model by evaluating it on the unseen 2% of removed edges. We repeat this five times for each network and average metrics over each fold in order to generate our final model performance metrics. [3]

5.2 Results

We now present the results of our experiments below. We divide results across three boundaries: Task, Network, and Model. Tables 1 through 15 list softmax-classification results, while tables 16 through 24 list regression results. For each task, we include two tables per model trained on the task, namely one table for each network. In bold are the best values achieved across all models and embeddings for a given task and network for each metric. In grey are the best values achieved on each network for each model. Finally, our embedding notation is as follows:

1. **Baseline:** embeddings generated according to the original formulation of equation 1 by Yuan et al. where $c_i \in c_+, c_-$ (i.e. edge *sign* only)
2. **S+W:** embeddings generated according to equation 1 (edge *sign* and *weight*)
3. **F+S+W:** embeddings generated according to equation 2 (edge sign, weight, and source node fairness)
4. **G+F+S+W:** embeddings generated according to equation 3 (edge sign, weight, source node fairness, and target node goodness)

[3] We remove edges when generating embeddings in order to prevent information leakage

Table 1: **Mode Classification** Baseline results

Table 2: **OTCNet**

Precision	Recall	F1
0.3187	**0.5644**	0.4073

Table 3: **AlphaNet**

Precision	Recall	F1
0.3152	**0.5611**	0.4036

Table 4: **Logistic Regression** Classification Results

Table 5: **OTCNet**

Embedding	Precision	Recall	F1
Baseline	0.4422	0.4721	0.4539
S + W	0.4448	0.4732	0.4573
F + S + W	0.4410	0.4640	0.4512
G + F + S + W	0.4504	0.4780	0.4618

Table 6: **AlphaNet**

Embedding	Precision	Recall	F1
Baseline	0.4210	0.4246	0.4204
S + W	0.4405	0.4410	0.4384
F + S + W	0.4148	0.4280	0.4196
G + F + S + W	0.4275	0.4309	0.4275

Table 7: **SVM** Classification Results

Table 8: **OTCNet**

Embedding	Precision	Recall	F1
Baseline	0.4908	0.5237	0.4973
S + W	0.4726	0.5081	0.4858
F + S + W	0.4663	0.5044	0.4812
G + F + S + W	0.4877	0.5156	0.4890

Table 9: **AlphaNet**

Embedding	Precision	Recall	F1
Baseline	0.4467	0.4624	0.4467
S + W	0.4577	0.4691	0.4561
F + S + W	0.4422	0.4636	0.4475
G + F + S + W	0.4624	0.4708	0.4607

Table 10: **Decision Tree** Classification Results

Table 11: **OTCNet**

Embedding	Precision	Recall	F1
Baseline	0.4410	0.4074	0.4217
S + W	0.4455	0.4097	0.4249
F + S + W	0.4452	0.4092	0.4240
G + F + S + W	0.4311	0.3981	0.4122

Table 12: **AlphaNet**

Embedding	Precision	Recall	F1
Baseline	0.4227	0.3890	0.4025
S + W	0.4279	0.3927	0.4078
F + S + W	0.4188	0.3893	0.4021
G + F + S + W	0.4268	0.3936	0.4080

Table 13: **KNN** Classification Results

Table 14: **OTCNet**

Embedding	Precision	Recall	F1
Baseline	**0.5027**	0.5407	**0.5095**
S + W	0.5001	0.5373	0.5085
F + S + W	0.4968	0.5308	0.5032
G + F + S + W	0.4958	0.5327	0.5017

Table 15: **AlphaNet**

Embedding	Precision	Recall	F1
Baseline	0.4733	0.5057	0.4790
S + W	0.4606	0.4947	0.4686
F + S + W	**0.4820**	0.5216	**0.4915**
G + F + S + W	0.4719	0.50485	0.4757

Table 16: **Mean Regression** Baseline results

Table 17: **OTCNet**

RMSE
0.3564

Table 18: **AlphaNet**

RMSE
0.2819

Table 19: **Linear Regression** Regression Results

Table 20: **OTCNet**

Embedding	RMSE
Baseline	0.3351
S + W	0.3360
F + S + W	0.3325
G + F + S + W	0.3399

Table 21: **AlphaNet**

Embedding	RMSE
Baseline	0.2820
S + W	0.2864
F + S + W	0.2890
G + F + S + W	0.2831

Table 22: **SVM** Regression Results

Table 23: **OTCNet**

Embedding	RMSE
Baseline	**0.3051**
S + W	0.3073
F + S + W	0.3062
G + F + S + W	0.3110

Table 24: **AlphaNet**

Embedding	RMSE
Baseline	**0.2705**
S + W	0.2723
F + S + W	0.2745
G + F + S + W	0.2708

5.3 Analysis

5.3.1 Classification

Having run a fairly large set of experiments across multiple models and networks, we can analyze both the performance of embeddings within a single model for a microscopic view of embedding performance, as well as the performance of embeddings across models and networks for a more macroscopic understanding of both the performance of embedding *and* models. While comparing across models allows us to appreciate the extent to which a *model* can affect the relative performance of our different embeddings, comparing *within* a model allows us to directly compare the performance of the four embedding techniques.

We first note that the highest recall on each network was achieved by our mode baseline, which simply predicts the mode of the classes seen in training. This points to the fact that over half of the edges seen in training belong to *one* of the six classes. While none of our models, including our baseline, outperform the mode model on recall, we do outperform the naive model substantially on precision and F1 scores. Looking at the highlighted cells, we can see that our baseline model performed relatively well on the OTC Network relative to the Alpha Network. Specifically, in the OTC Network, looking at Table 14, we see that our baseline model records the highest precision and F1 scores across all models when run through the KNN model. Additionally, the baseline model has the highest precision, recall, and F1 scores on the OTC Network in our experiments with both the SVM model and KNN model.

Looking across the Alpha Network, we see that at least one, and typically most, of our three augmented embedding methods outperformed the baseline model on each metric. Embeddings incorporating goodness, fairness, edge-sign, and edge-weight in accordance with equation 3 performed particularly well, scoring the highest in our SVM model all three metrics and scoring the highest on leverage and recall and nearly precision in our Decision Tree model (see Table 9, 12). Finally, we see that fairness, edge-sign, and edge-weight embeddings (equation 2) achieve the highest precision and F1 scores across all Alpha Network models (see Table 15).

5.3.2 Regression

We first note that the naive mean model, which predicts the mean edge-weight seen in training at test time, outperforms all embeddings for the Alpha Network when used to train our Linear Regression model. However, we again see a disparity in embedding and model performance across networks, as our RMSE values decrease a non-trivial amount on both models for each embedding on the OTC Network compared to the naive mean baseline. However, we are not able to outperform the baseline embeddings on either network on either model, save for F+S+W embeddings on the OTC Network and Linear Regression model (Table 20).

6 Discussion & Future Work

Our work can be extended in a variety of ways, both in terms of embedding techniques, model selection, and dataset selection.

As previously mentioned, the OTC exchange and Alpha exchange have weights that are highly clustered around 0, with the majority of edges belonging to one of six classes. In order to more rigorously each of the proposed embedding techniques, we propose repeating our experiments on networks with a less skewed edge-weight distribution. Additionally, while edge-weight in the Bitcoin networks analyzed here corresponds to notions of trust, our work can be applied to any weighted and signed network. Thus, we propose applying the above experiments to networks in which edge weight and sign correlate to notions beyond just trust, as well as to *undirected* networks.

When analyzing goodness and fairness scores across both Bitcoin networks, we found that a high percentage of goodness scores were clustered very close to 0. As a result, scaling the final embedding vectors by the goodness score of the target node has a high likelihood of scaling down the magnitude of the embedding vectors significantly. In future work, we are interested in both scaling goodness scores to prevent such down-scaling, as well as normalizing the magnitude of our embeddings. This may make our embedding techniques more robust to highly skewed data.

Finally, in future work we would like to combine the multiple classification models we experiment using ensemble methods in order to potentially outperform any single model. Additionally, we would like to experiment more rigorously with hyper-parameter tuning for each of our models, as we were constrained by time in our hyper-parameter searches.

7 Individual Contributions

While the work was very evenly split between the two of us, and we worked jointly on nearly every part of the assignment, primary ownership of each portion was roughly split as follows:

1. **Sam**: Coming up with/implementing embedding algorithms 1 and 2, running and formatting classification/regression experiments, model search/selection.
2. **Jacob**: Coming up with/implementing embedding algorithm 3, implementing code to generate network train/test splits, PCA analysis and network visualizations, modifying starter code to generate random walks/fairness goodness scores.

A Model Hyperparameters

A.A Classification

Logistic Regression(**penalty** : $l2$, **maxiter** : 100)
SVM(**kernel** : rbf, **gamma** : nsamples*X.std(), **decision function** = one-v-rest)
Decision Tree(**split criterion** : gini, **max depth** : ∞)
KNN(**num neighbors** : 3)

A.B Regression

Linear Regression(**normalize=True**)
SVM(**kernel** : rbf, **gamma** : nsamples*X.std(), **penalty** = 0.1)

References

[1] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. Subrahmanian, "Rev2: Fraudulent user prediction in rating platforms," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 333–341, ACM, 2018.

[2] Y. Hou and L. B. Holder, "Deep learning approach to link weight prediction," in *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 1855–1862, IEEE, 2017.

[3] C. Aicher, A. Z. Jacobs, and A. Clauset, "Learning latent block structure in weighted networks," *Journal of Complex Networks*, vol. 3, no. 2, pp. 221–248, 2014.

[4] B. Zhu, Y. Xia, and X.-J. Zhang, "Weight prediction in complex networks based on neighbor set," *Scientific reports*, vol. 6, p. 38080, 2016.

[5] CS231N, "Linear classification."

[6] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks.," in *ICDM*, pp. 221–230, 2016.

[7] S. Yuan, X. Wu, and Y. Xiang, "Sne: signed network embedding," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 183–195, Springer, 2017.