CS 224W Final Project

Link Prediction on Crunchbase Investment Network

Yanshu Hong, yanshuh@stanford.edu
Jiayu Wu, jiayuwu@stanford.edu

INTRODUCTION

In this project, we are interested in predicting future VC investments from past investment data. We found a theoretical framework ("link prediction") and a relevant dataset from Crunchbase that fit our needs. We aim to evaluate different link prediction methods surveyed from literature on the CrunchBase dataset, synthesize and hopefully improve upon some of the methods, and also develop a few first-hand insights into the VC-startup ecosystem and its dynamics.

Our motivations behind this project are threefold. First, solving the link prediction problem in the context of investment network is valuable because it helps us better understand investment activities. Second, while in class we learned link prediction on Facebook undirected social network, due to the bipartite nature of investment graph, it is interesting to see how we can apply the link prediction methods to such a graph. Third, we think that the CrunchBase investment dataset is a rare-find and it worths a thorough analysis.

The results we obtained are half positive and half negative. Most predictors outperform the random predictor, indicating a certain correlation between the structure of the investment network and individual investments. However, even our best predictor achieves a F1 score of merely 1%. This, to some extent, confirms the old saying that "investment is an art, not a science".

RELATED WORK

Link prediction problem for social networks is clearly defined by Liben-Nowell and Kleinberg (2007). We would like to adopt their definition as a working foundation for our project. Consider an undirected graph G where all edges are annotated with timestamps (can be interpreted as the time an edge is added). Further suppose we have four timestamps t_0, t_0', t_1, t_1' such that $t_0 < t_0' < t_1 < t_1'$. The link prediction problem is to predict from graph G with only edges created between time $[t_0, t_0']$ to *new* edges that will be created between time $[t_1, t_1']$. For simplicity, we do not consider new nodes that are added through time and restrict the vertex set only to nodes that are present in G at t_0.

Liben-Nowell and Kleinberg discuss various methods for link prediction. The core idea is to devise a scoring function for all node pairs. With this scoring function, we can make predictions by taking, say n, node pairs with the highest scores. Liben-Nowell and Kleinberg survey a wide range of scoring methods and pigeonhole them into three categories:

(1) **methods based on node neighbors**: common neighbors, Jaccard, Adamic/Adar, etc. These methods are "local" in the sense that they only look at the immediate neighbors of some node.
(2) **methods based on the ensemble of all paths**: Katz, hitting time, SimRank, etc. These methods are more "global" as they extract information from the entire graph topology instead of the immediate. For example, the Katz method considers a weighted sum of the number of paths with length l between two points for all l. Weight for larger l is set to be smaller as intuitively a longer link explains less the "proximity" of two nodes.
(3) **meta-approaches**: LSA, unseen bigrams, etc. These methods are meant to further improve efficiency and/or accuracy of the methods in the first two categories.

In the same paper, five co-authorship networks are used for experiment. A random predictor that randomly predicts selects pairs of authors that do not share edges in the training graph is used as the baseline. The results show that although the raw accuracy of the above-mentioned methods is relatively low, they perform significantly better than the random predictor, which indicates that graph topology indeed provides some useful information regarding graph evolution. The best scoring function they find out is Adamic/Adar.

Since our CrunchBase dataset models investment relations as a bipartite graph (companies and investors are nodes while investment relations are edges), the general methods proposed in Liben-Nowell and Kleinberg's paper are not tailored specifically to bipartite graphs. In particular, because two nodes that are not connected by an edge in a bipartite graph share no common neighbors, the whole

category of neighbors-based methods does not work in bipartite graphs.

The paper by Kunegis et al. (2010) and that by Allali et al. (2011) signal two general directions to solve the link prediction problem for bipartite graphs:

(1) **run path-based methods on the bipartite graph**. Kunegis et al. convert most path-based methods into polynomial functions of the adjacency matrix. Suppose A is the adjacency matrix of any graph, $A^i_{(x,y)}$ denotes the number of paths of length i from node x to node y. Since in a bipartite graph, we are only interested in paths of *odd* length, we can constrain the polynomial to consist of only odd powers. This will be one major class of methods that we implement in this paper.

(2) **convert into a regular graph**. We can convert a bipartite graph into an unweighted or weighted regular graph. For example, if we have a bipartite graph between authors and papers, we can turn that into a co-authorship graph. In our case, we can turn the bipartite graph into a co-investment graph (with all nodes as investors). We can also use the neighbor-based scoring methods to add weights onto the edges. Moreover, Allali et al. propose a way to predict *internal links* for bipartite graphs, where a link is internal if its addition does not change the structure of the converted regular graph (called the *projection*). We'll also try this method.

Other papers like Li and Chen's (2013) propose machine learning methods to solve the link prediction problem. We did not find this paper particularly convincing, so we decide to design our own machine learning approach that takes properties of companies and investors as well as properties of the network as features. All methods are described in greater detail in the Models and Methods section.

Data

The investment dataset is provided by crunchbase.com. The dataset contains 168,648 venture capital investment records, ranging from May 1977 to December 2015. Each row in the dataset (as a .csv file) represents one investment. For each row, we are interested in columns such as company name, company category, location/region of the company, investor name, location/region of the investor, funding round, funding time, raised amount and etc. We parse the .csv file into python objects (`class Company`, `class Investor`, `class Investment`) and create references between them. Note that we really wish to know the exact amount of money each investment involves, but such information is not directly provided within the dataset. We know the total investment amount for some investment rounds, but we have no way splitting the round amount into individual ones.

To solve the link prediction problem using this dataset, we built an undirected bipartite graph from the dataset using NetworkX, with all edges going between investors and companies. We have 30,726 companies and 66,368 investors in total. Note that there are 836 companies that are also investors, but we keep them as separate nodes in two clusters of the bipartite graph.

We use investments from 2010-2014 as the training data, and investments from 2015 as the test data. The bipartite graph for training is built only from investors and companies that are present between 2010-2014. We ignore all new nodes (investors and companies likewise) created in 2015. In order to reduce the graph size (so as to make computation tractable) and make the graph less sparse, We filtered out most investors and companies with low degrees. The specific methods we use are described in detail in the Evaluation section.

Preliminary Analysis

Before we proceed to solve the link prediction problem on the dataset, we would like to have some preliminary understanding of the network. Thus, we will draw from concepts that we've learned throughout this course and present some network statistics.

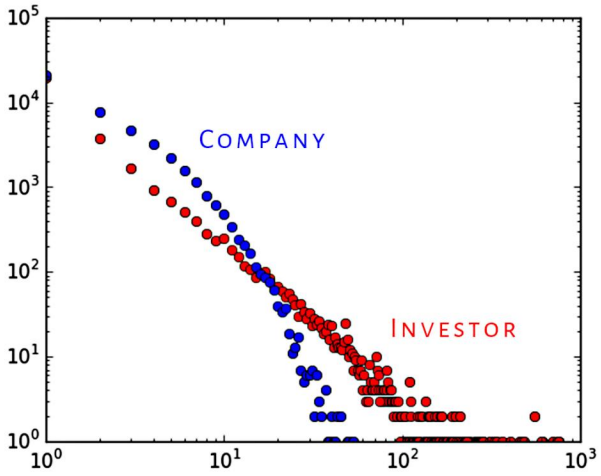

Fig. 1: Degree Distribution for Investors

From Fig. 1, We notice that the degree distribution for investors (size of the portfolio) follows the power-law distribution. A small number of investors have made a large chunk of all investments. Indeed, many start-ups turn to famous venture capital firms to seek seed funding, and thus the brand name of an investor is essential in the venture capital space. In comparison, the degree distribution for companies (number of investors) is slightly more equal than that for investors. When we make predictions, adjusting our models to reflect the two empirical degree distributions might be helpful. The investors and companies with the highest degree are shown in Table 1 and Table 2.

According to Table 1, we see that top investors have made a large number of investments and the number of investments decline exponentially as the investor's ranking lowers. According to Table 2, we see top companies receive a comparable number of investments, and in general, a

company's degree is much lower than that of an investor. Therefore, when we consider whether to predict an edge between an investor and company (and when we decide the ratio of investor nodes to company nodes when we build the training graph), we should take into consideration the imbalance between the average degree of an investor and the average degree of a company.

Investor	Degree
Sequoia Capital	1033
Wayra	936
500 Startups	932
New Enterprise Associates	902
Intel Capital	861
Y Combinator	862
Accel	779
Kleiner Perkins Caufield	727
Start-Up Chile	727
SV Angel	625

TABLE 1: Top 10 investors with highest degree

Company	Degree
Uber	64
DocuSign	61
Fab	61
Pinterest	60
Practice Fusion	59
Mattermark	59
ecomom	59
EndoGastric Solutions	57
Hackers/Founders	57
PTC Therapeutics	57

TABLE 2: Top 10 companies with highest degree

Now, we will look at the category breakdown of investors' portfolio companies.

According to Table 3 and Table 4, we see that two famous venture capital/incubator firms, Y Combinator and 500 Startups, have very similar portfolio structure. The top 10 categories for these two investors almost overlap. However at the same time, we notice that their portfolio companies are extremely diversified among different categories, which suggests that using information about a company's category might be able to improve the prediction accuracy slightly, but too much.

Category	Percentage
Software	5.48%
Curated Web	4.43%
Mobile	3.90%
E-Commerce	3.14%
Enterprise Software	2.71%
Analytics	1.76%
SaaS	1.67%
Social Media	1.52%
Games	1.38%
Advertising	1.33%

TABLE 3: Top 10 Categories, Portfolio Breakdown for Y Combinator

Category	Percentage
E-Commerce	4.28%
Mobile	4.22%
Software	3.39%
Curated Web	3.26%
SaaS	2.52%
Enterprise Software	2.43%
Advertising	2.36%
Marketplaces	2.20%
Social Media	2.11%
Analytics	1.66%

TABLE 4: Top 10 Categories, Portfolio Breakdown for 500 Startups

To understand further the connections between investors and between companies, we transform the original bipartite graph into two undirected graphs, one for investors and the other for companies. In the co-investor network, two investor nodes share an edge if they have invested in the same company; likewise, in the co-company network, two company nodes share an edge if they have been invested by the same investor. We calculate the average clustering coefficient for both projected graphs. Given Table 5 (see next page), we see that both graphs exhibit community structure.

The statistics on the investment network show that apart from the graph structure itself, domain knowledge and information may be incorporated to our model in order to improve prediction. For example, when we are considering a possible investment, we can check whether a company is in the sector an investor has made significant investments in. In the next section, we'll first present prediction methods based only on network structures. THen, we'll detail our design of the machine learning approach that aims to combine contextual information with network properties.

MODELS AND METHODS

Internal Link Method

As mentioned in Related Work section, Allali et. al. notice that there is a special subset of links that when added to the bipartite graph, does not change the graph projection. They propose a method that aims to predict only the internal links. We first implemented this method.

The projected graph in our case is the co-investing graph. The nodes are investors and edges exist between two investors that have invested in at least one common company (two investor nodes have common neighbors in the bipartite graph). We use the Jaccard-coefficients of the two investor to model the edge weights between nodes in the bipartite graph. In particular, the Jaccard-coefficient is defined as the following:

$$\text{Jaccard}(u,v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

where $N(u)$ denotes the set of all neighbors of node u.

It has been shown in the paper that one link in the bipartite graph (I, C) (where I is an investor node and C is a company node) is an internal link if all neighbors of C

	Co-investor Network	Co-company Network
Avg. Clustering Coefficient	0.624	0.480

TABLE 5: Average Clustering Coefficient

are connected to I in the projected graph. The internal link prediction method first thresholds the projected graph on its edge weights by a threshold τ. The resulted graph G_τ only have edges with weight at least τ. The method then considers all missing links (I, C) in the bipartite graph and test if at least one node in $N(C)$ is connected to I in G_τ. If there is, the edge is predicted.

The number of predicted edges is controlled by the threshold τ.

To accelerate the computation, we made a small tweak to the original algorithm. Instead of enumerating over every pair of missing link in the bipartite graph, we enumerate over all existing edges in the training graph. For an existing edge (I, C), we then enumerate over every pair of missing link between $N(I)$ and $N(C)$. By the definition of internal links, it is not hard to see that the tweaked algorithm is still exhaustive. However, since the training graph is very sparse, this tweak could achieve a considerable performance gain.

Node-based Method

The internal link methods are only able to predict links that are internal. Not all new links are necessarily internal, so the internal link method can be defective, by design. The more general framework for the link prediction problem is the scoring-based method as used by Liben-Nowell and Kleinberg. The scoring-based method assigns a score to each missing link (I, C) and predicts the top k links with the highest score or thresholds the list with a score cutoff τ. In our implementation, we normalize all scores with the largest score (of all potential links), to make it easier to choose τ ($\tau \leq 1$).

Since most scoring functions based on node neighbors do not work on a bipartite graph (for example, I and C will have no common neighbors), the only node-based scoring function that we can still use here is the preferential attachment scoring function. In particular, it defines,

$$\text{score}(u, v) = |N(u)| \cdot |N(v)|$$

The preferential attachment method uses this scoring function.

Path-based Method

Another set of scoring functions is the path-based ones. They consider beyond the immediate neighbors of each node and try to incorporate information based on the ensemble of all paths between them. The intuition is that the *more short* paths there are between two nodes, the stronger connected these two nodes are. Therefore, it is more likely that there will be a new link between these two nodes.

One path-based scoring method is the *Katz* scoring function (1953). It is defined over the collection of all paths, exponentially damped by length. In particular, with the damping factor β,

$$\text{Katz}_\beta(u, v) = \sum_{l=1}^{\infty} \beta^l |\text{paths}_l(u, v)|$$

where $\text{paths}_l(u, v)$ is the collection of all paths from u to v with length l. In our case, we are interested in paths between nodes on different sides of the bipartite graph. So the paths can be only of odd length. Therefore, for the link (I, C), its Katz function is defined as,

$$\text{Katz}_\beta(I, C) = \sum_{i=0}^{\infty} \beta^{1+2i} |\text{paths}_{1+2i}(I, C)|$$

Suppose we know the diameter of the graph d. We can calculate the Katz function with i summing up to $(d-1)/2$. Or, according to Kunegis et al., the Katz scoring function can be written in closed form using the adjacency matrix of the graph, A. They call it the Odd von Neumann Pseudokernal. Define,

$$K_\beta(A) = \beta A(I - \beta^2 A^2)^{-1}$$

We have,

$$\text{Katz}_\beta(I, C) = [K_\beta(A)]_{I,C}$$

We will use both the finite formula and the pseudokernal method to calculate the Katz scoring function.

Machine Learning Method

To use any machine learning method, we need to first construct features for each edge.

We divide features into three categories: graph features, investor features and company features. Graph features are information we can obtain from the investment graph itself. For example, we use degree of investor, degree of company, preferential attachment score and Katz score as graph features. Company features are what we can obtain from the CrunchBase dataset other than the investment graph itself. For example, whether the company was founded recently, how many rounds of funding the company has had, and its geological information. Likewise, investor features are information specific to the investor but not directly available in the investment graph. For example, we consider the five most favorite sectors of an investor based on its past investments.

The challenge of formulating the problem as a machine learning problem lies in the the nature of link prediction problem, because we want to predict those edges that we already know do not exist in the training graph. This means that we need to *test* whatever model we train on all missing edges of the training graph to get the predictions. But if

this is the case, we are not left with any negative example to train out model. Since if we also train on the negative edges (with label 0) and positive edges (with label 1) of the training graph, we are training and testing on the same data! Therefore, the naive approach of labeling and training does not work for our link prediction problem. We propose two slightly more sophisticated methods.

The first method is an edge-based prediction method. Just like cross-validation, we split the edges that do not exist in the training graph into two sets, A and B. We use set B (labeled 0) along with all existing edges in the training graph (labeled 1) to construct a balanced training dataset (the size of B is similar to the number of edges). We then train a model on this training set, and finally use the model to predict edges in a different set A. We train a number of models using different splittings such that each of the edges that do not exist in the training graph gets exactly one prediction and the corresponding model is trained with the data that does not include the edge itself.

The second method is a company-based prediction method. The essential idea of the second method is to first learn which companies are going to be popular in the test period, and then try to predict edges between them and investors that are most interested in them. The hypothesis behind this method is that a popular startup may attract attention from a couple of venture capital investors, and thus several venture capital investors may collaborate and invest in this startup together, which results in several new edges connecting to the same company. In this method, we apply the first method to each investor, from which we estimate what probabilities one investor will invest in one of the companies in the test period. Then we aggregate results from all investors, and find out the most popular companies. Now since we have decided on one end of the edges we are going to predict, we need to figure out the other end. For each of these popular companies, we select the investors that we think are most interested in that company, where interest is measured by predicted probability, to finally confirm our predicted edges.

We utilize random forest classifier in both of the methods. Random forest is an ensemble method that utilizes a number of decision trees and bootstrapped samples. It has a low variance because its output is the average of the output of its trees and at each splitting in a tree, it only considers a random subset of features.

Hyperparameter selection and tuning is also a major issue in our machine learning methods. First, for our edge-based prediction method, we need to decide how many models we train, equivalently how to split the possible edges to different set As and Bs. If the number of models we train is too small, then we may have a biased predictor as we utilize too small part of the training data; if the number of models we train is too large, then the running time for this method may be an issue. Furthermore, for each splitting, we also need to carefully control the number of negative examples (label-0 edges) in order to avoid the imbalanced data problem. Second, for our company-based prediction method, we need to decide how many popular companies we want to consider and for each popular company, how many most interested investors we should consider. In addition, there are also some hyperparameters related to the random forest model. For example, we can tune the number of trees and the number of features we intend to use at each tree splitting.

EVALUATION

In an effort to lighten the computation and densify the graph, we select a subset of popular investors and a subset of popular companies based on node degree. Since in the Preliminary Analysis section, we've seen that the top degree for an investor is more than 10 times the top degree for a company, we decide to take 10 times more companies than investors. Therefore, the trimmed training graph consists of around 100 popular investors, around 1,000 popular companies, and all investments between them within the time period of 2010 to 2014. Reaping all nodes with degree zero, we end up with a bipartite graph with 96 investor nodes, 874 company nodes and 2878 investment edges.

There are 24735 investments in total made in 2015, among which only 470 (1.9%) are between investors and companies whose nodes are in the training graph. Excluding multiple investments and re-investments, there are 116 new investing relations made in 2015. In other words, with the same node structure, the bipartite graph constructed from 2010-2014 will have 116 more edges in 2015. These edges are the ones that we wish to predict. We run all methods described in the Models and Methods section and calculate performance statistics (precision, recall, and F1) against the answer set. In particular, since the diameter of our training set is 8, the finite Katz scoring function is essentially,

$$\text{Katz}_\beta = \beta A + (\beta A)^3 + (\beta A)^5 + (\beta A)^7.$$

We will also consider a random predictor as a baseline. A random predictor predicts k links randomly from the set of all possible investor-company links missing from the training graph. Suppose there are M such missing links, and the correct answer contains A links. The expected number of true positives the random predictor will get is kA/M. Thus, the expected precision is A/M and the expected recall is k/M. The F1 score is defined as,

$$\begin{aligned} \text{F1} &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \\ &= 2/M \cdot \frac{Ak}{(A+k)} \end{aligned}$$

Since $k \leq M$, $\text{F1} \leq 2A/(A+M)$. With our data, we have $M = 96 * 874 - 2878 = 81026$ and $A = 116$. Therefore, the random predictor will have an expected precision of 0.14% and a F1 score at most 0.28%. We expect our methods to perform at least better than the random predictor.

METHOD		THRESHOLD	NUM PRED.	T POS.	PRECI.	RECALL	F1
Random					0.14%		0.28%
Internal Link		0.10	52382	12	0.02%	10.34%	0.05%
		0.12	48086	7	0.01%	6.03%	0.03%
		0.16	38005	5	0.01%	4.31%	0.03%
Preferential Attachment		0.15	4883	14	0.29%	12.07%	0.57%
		0.20	2398	4	0.17%	3.45%	0.32%
Katz	$\beta = 0.1$	0.15	2572	11	0.43%	9.48%	0.82%
		0.20	1562	5	0.32%	4.31%	0.60%
	$\beta = 0.05$	0.15	2646	11	0.42%	9.48%	0.80%
		0.20	1621	6	0.37%	5.71%	0.69%
van Neumann Pseudokernal	$\beta = 0.1$	0.15	1033	10	0.97%	8.62%	1.74%
		0.20	600	5	0.83%	4.31%	1.39%
	$\beta = 0.05$	0.15	2581	11	0.43%	9.48%	0.82%
		0.20	1570	6	0.38%	5.71%	0.71%
Machine Learning (edge-based)			1706	5	0.29%	4.31%	0.54%
Machine Learning (company-based)			871	4	0.46%	3.45%	0.81%

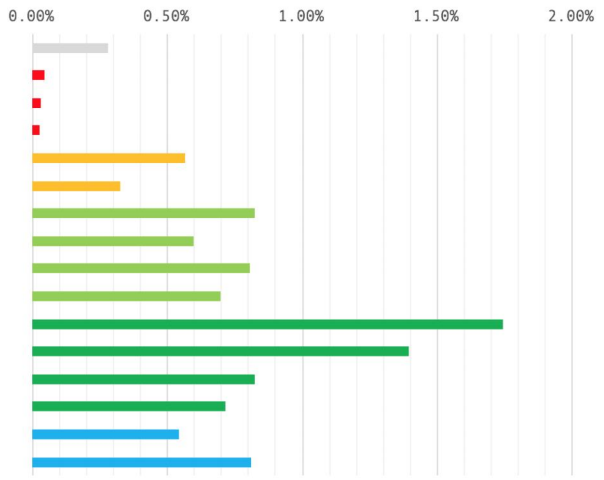

Fig. 2: Evaluation Results

Results and Analysis

We evaluate all five methods described in the Models and Methods section. For each method, we try different thresholds to trade off between precision and recall. For the Katz (finite) and Odd von Neumann Pseudokernal method, we try two different damping factors β, 0.1 and 0.05. We show the results in Fig. 2.

The best F1 score is achieved by Katz scoring method with Odd von Neumann Pseudokenal, damping factor $\beta = 0.1$ and threshold $\tau = 0.15$. It outperforms the random predictor 6.2×. However, none of the methods we tried here achieves satisfactory performance in absolute terms. This shows the inherent difficulty of this problem.

Surprisingly, the internal link predictor performs even worse than the random predictor. Per calculation, out of 116 new links, only 48 are internal. From the sheer number of predictions made by the predictor (in order to get ≥ 1 true positives), we could infer that the predictor has a hard time distinguishing candidate edges. Refer back to the original paper, we see that internal link prediction works much better on graph with 10^6 nodes and 10^6 edges. So, our graph might just be too small for link prediction to work. And investment relations might not exhibit the "triangle-closing" tendency as a file-provider network or user-tag network do.

Since the path-based scoring methods are expected to leverage more information than node-based ones, Katz (finite) scoring function indeed performs better than preferential attachment. Also, setting $\beta = 0.1$ works better in all these cases than setting $\beta = 0.05$, potentially because long-stretched relations (paths with length 3, 5, 7) can be important indicators within the investment network. However, it is not clear why the pseudokernal method performs almost twice better than the Katz (finite).

Despite we used extra contextual information about companies and investors, our machine learning methods only give mediocre results in terms of F1 score. On top of tuning the hyperparameters, we found that the company-based method using 10 most popular companies and 100 most interested investors for each company achieves the best results among the machine learning methods. We notice that although there are only 116 edges actually formed in the test period, some companies, for example, Airbnb, are involved in more than one investment (with more than one investor). We still see the tendency that a minority of companies get a majority of investments. Without saying, the task of finding the most popular companies is essential to get correct predictions. Yet, even the question which investors would invest in these popular companies is not an easy one to answer.

U-test Results

To better understand whether one scoring function *ultimately* works or not, we design a U-test to see whether the score distribution of the correct new edges given by the scoring function is actually special. In other words, if we plot a histogram of the scores of all missing edges in the training graph, will the correct answers have their scores concentrate in one location on the histogram? To test this, we calculate the scores of all 116 correct predictions and put them into set A. Then we randomly choose 3 times more missing edges in the training graph and form their scores into set B. If our prediction framework were to work (choose the top k scores), most, if not all, scores in B should be lower than all scores in A. We employ Mann-Whitney U test to see how different the scores of true edges (set A) are from those of random edges (set B). The perfect case will give us a U-stat of 0. The lower the U-stat is, the better a scoring function distinguishes the correct new edges.

	U-stat	p-value
Pref. A.	14468.5	2.4×10^{-6}
Katz	13233.0	1.4×10^{-8}
van Neumann	20086.5	0.4691
Random	20122.0	0.4804

From the results above, we see that the U-stats of preferential attachment and Katz (finite) are indeed lower than

that of the random predictor. This shows that the former two scoring functions actually help distinguishing the correct new edges. However, the U-stat by Odd van Neumann Pseudokernal is not that different from random. This casts doubt onto the result that this method outperforms all other methods in term of F1 score. It is possible that Odd van Neumann Pseudokernel gets lucky specifically with our data. But given its $2\times$ improvement over Katz (finite), and the high p-value of the U-stat, this claim also needs to be taken with care. Future work is needed.

Last but not least, we would like to give some speculations why all our methods perform poorly on absolute terms. We hope that these speculations could signal directions for future research.

Sparse Graph

The bipartite graph of investors and companies is very sparse. For investments between 2010-2014, we have more than 20,000 investors and more than 20,000 companies, while we only have 50,000 investments. Given that we only have around $1/8000$ number of edges of the complete graph, link prediction is really hard, because we do not have too much positive information to start with and random guessing almost never hits. In this paper, we decide to filter out nodes with low degree in order to have a denser graph. Though the density factor of the trimmed graph increases to $1/29$, it is nevertheless still sparse by normal standard. We could include more nodes into the graph to bring in more information, but it will make the graph even more sparse, and the prediction even harder.

Information Loss

An investment relation may not only depend on the graph structure itself. For example, the graph structure may not capture the preference information of an investor because some investors prefer investing in seed rounds while others prefer investing in pre-IPO rounds.

Furthermore, it is widely believed that networking between people is essential in the venture capital and startup space. Thus a decision on investment may heavily depend on the personal relationship between the founders and the venture capitalists. In addition, there are many early-stage investors who state that they are not investing in a company, but rather the founding team of that company. Therefore, the human capital aspect of a startup may play a major role in attracting investments. Future work on this topic could also consider data from social networks.

Unpredictability of Venture Investments

It takes a long time for an investor to decide a venture investment, from sourcing a deal, due diligence, negotiating the term sheet, to acquiring approval from the investment committee. Each step is difficult to predict, let alone the end result of the process. Many believe that venture capital investment is more of an art than science, for example, many angel investors invest in companies based on their past experience and their connections. Intuitions play an import role in this field, rather than a rigorous scientific model. Therefore, the inherent unpredictability makes the link prediction problem extremely difficult.

Team Work

We did an equal split on the work. Jiayu found the dataset, performed preliminary analysis, designed and implemented the machine learning method. Yanshu proprocessed the dataset, designed data framework, designed and implemented the internal link method, node-based method and path-based method. Both work on the reports.

References

Oussama Allali, Clémence Magnien, and Matthieu Latapy. Link prediction in bipartite graphs using internal links and weighted projection. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 936–941. IEEE, 2011.

Nesserine Benchettara, Rushed Kanawati, and Celine 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.

Jérôme Kunegis, Ernesto W De Luca, and Sahin Albayrak. The link prediction problem in bipartite networks. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 380–389. Springer, 2010.

Xin Li and Hsinchun Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2):880–890, 2013.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.