Joint Link Prediction in Temporal Networks

Kyle Griswold
kggriswo

Seokho Hong
seokho

Naveen Arivazhagan
naveen67
Stanford University
450 Serra Mall
Stanford, CA 94305

Abstract

Predicting the evolution of a network through time – how a network adds or removes edges – is a challenging task that has had limited exploration. Keeping with the established link prediction problem, we expand upon previous efforts to predict which nodes will gain or lose their edges at a later point in time. We develop more complex metrics to identify how topographical features of a graph can suggest its evolution. Most importantly, we remove the assumption that edges of a graph form independent of each other and attempt to predict links jointly.

1 Introduction

Many network analysis tasks involve dealing with dynamic networks in which new nodes are added and new edges are being formed. In this context an important and interesting problem is that of link prediction. This is the task of predicting which new edges are going to be formed in a given graph. An example of this in the context of social networks, for example in a friendship graph where nodes are people and edges connect friends, is predicting which nodes are likely to become friends in the near future.

However, current methods that attempt to solve this problem have an important drawback. Given the network's evolutionary history, they predict the links that will be formed in the future independent of each other. However, interactions almost never happen in isolation of each other. If a professor sends an email to one student, then she is more likely to send an email to all students. Similarly, if an author is going to collaborate with another author, then this will influence the probability of her collaborating with other authors. However, current approaches for link prediction simply focus on whether nodes u and v will be connected in the next time step without considering what other interactions u and v will also participate in.

In this paper, we try different approaches to break down this independence assumption that links are formed independent of each other. We formulate a few tractable algorithms to jointly predict the interactions of a node with the other nodes in the graph. We then evaluate the performance of these algorithms and report result on graphs such as the Facebook Wall Post Network in which we expect the phenomenon of links being formed in a joint manner to play an important role.

2 Prior Work

Nowell and Kleinberg [2] first formally propose the problem of link prediction in social networks. They use the intuition that similar people or people who have many things in common are more likely to become friends to predict future connections based on the current state of the network. To do this they use various measures of similarity such as Common Neighbors, Adamic Adar, Preferential Attachment, Katz score, Page Rank, Similarity Rank, Low-Rank Approximation and Clustering. These similarity measures are then used to score pairs of nodes, which is used as a measure of how likely they are to connect to each other.

Soundarajan and Hopcraft [3] make use of community information to improve the precision of link prediction by better estimating the similarity between a pair of nodes. They do so by modifying the traditional similarity measures such as Common Neighbors and Resource Allocation to score pairs of nodes that share many communities higher than pairs that do not share communities. Using this community sensitive metric for similarity, they are able to higher performances than the unmodified similarity measures.

Huang and Zeng [1] proposed an approach to detect anomalous emails. In their work, they

model the one to many relationship implicit in emails: a single sender with potentially many recipients. They estimate the likelihood of there being an email interaction between a pair of nodes using traditional similarity metrics. Following this, they estimate the conditional probability of an email being sent to a certain number of recipients as the normalized average of the scores of the pairs formed by the sender and each of the recipients. This is then multiplied by the probability of an email having that number of recipients to get the probability of the email. While this work is very similar to ours in that it attempts to model one to many relationships instead of the naive one to one, it still falls short of our objectives as it still makes the independence assumption when it estimates the joint probably as simply being proportional to the average of the individual scores.

3 Data

3.1 Description

We decided to use the Facebook Wall Post Network (facebook-wosn-wall). It is a publicly available citation network that contains data collected from about 2005 to 2009. [4]

We had a variety of types of datasets to choose from such as gene interactions, collaboration networks, communication networks (email, phone, text messaging) etc. But we chose the Facebook network because it met all of the criteria we needed for the problem, and it was reasonably sized so that we can get meaningful results, but also don't have to wait to long for the computer to run them.

3.2 Modeling

The wall-post network data can be naturally represented as a graph $G = (V, E)$, The set of nodes, V, represent the users. There is an edge $e \in E$ for every wall post such that e is an edge from the person who posted to the person's wall they posted on.

In our paper we also consider the temporal nature of the network, thus for any edge $e \in E$ we use $t(e)$ to denote the time at which the interaction takes place. Define $G[t, t']$ as the graph G consisting of all edges for which $t \leq t(e) < t'$. The final link prediction problem we consider is then formally defined as follows: given $G[t_0, t_0']$ predict the set of edges E in $G[t_1, t_1']$, where $t_0 < t_0' \leq t_1 < t_1'$.

Note that in a given time period $[t_0, t_0']$, not all the nodes of a given graph, G, are connected. We will therefore restrict our link prediction task to among vertices belonging to the largest connected component in $G[t_0, t_0']$. This is a very natural and intuitive thing to do as there is very little information available about how a node will connect in the future if it has zero connections in the past.

We currently do not consider weighted or directed edges, but that can be accomplished in future projects on this topic.

4 Formal Definition of the Problem

The main approach used to measure the likelihood of the formation of a link between two nodes u and v is measuring the similarity or relatedness between them. The intuition is that nodes which are similar are more likely to 'interact with' (link to) each other. Graphical structure has not been fully explored, however, and simple similarity functions leave a lot of the graph topology unused in the prediction task.

The link prediction papers that we have come across have assumed that the formation of each link in $G[t_1, t_1']$ is independent of every other such link formation. This allowed them to calculate $P(e(u, v) \in E(G[t_1, t_1'])|G[t_0, t_0'])$ separately for every $u, v \in V$ in the graph. Our paper is focused on removing a portion of this assumption.

This means that if we label the possible edges $e_1, e_2, ..., e_{\binom{n}{2}} \in E$, then we want to find

$$P(\wedge_{i=1}^{\binom{n}{2}}(e_i \in E(G[t_1, t_1']))|G[t_0, t_0'])$$

Using the chain rule for probability, we get:

$$P(e_1 \in E(G[t_1, t_1'])|G[t_0, t_0'])*$$

$$P(e_2 \in E(G[t_1, t_1'])|e_1 \in E(G[t_1, t_1']), G[t_0, t_0'])*...*$$

$$P(e_{\binom{n}{2}} \in E(G[t_1, t_1'])|\wedge_{i=1}^{\binom{n}{2}-1}(e_i \in E(G[t_1, t_1'])), G[t_0, t_0'])$$

Estimating this joint probability distribution is the goal of our project.

In this project, we make two improvements upon past work. The first is to extend the simple similarity functions to more expressive and complex functions. This part is merely a revision

of the function $f(u, v, G[t_0, t'_0])$ that we will use. We believe this will improve link prediction because previous work has already established that graph topology can improve the prediction task, and adding additional information can only help.

The second is to make each prediction $h(u, v)$ dependent on other predictions made for $G[t_1, t'_1]$. We do this by including the information for $f(u, w, G[t_0, t'_0])$ and $f(w, v, G[t_0, t'_0])$ for third party nodes w in our new predictor for whether (u, v) is in $G[t_1, t'_1]$. Since there are far too many nodes to feed into our model at once, we will approximate the above model by only feeding in the information for k other nodes, where k is a parameter of our algorithm. The process for picking the k nodes to condition on is based on heuristics for which which of these predictions might contribute the most to the current prediction task. We believe this will improve link prediction because links in a graph are usually not independent of each other. Removing this assumption should therefore give more accurate link prediction.

Note that this is a very crude approximation of true joint link prediction, however it is the closest we are able to get without going into much more complex models.

5 Algorithms

Our main algorithm will take a basic feature function $f(u, v, G[t_0, t'_0])$ and a hyper-parameter K. Then for each edge we find K other nodes we think will be useful and include their features when paired with our initial two nodes. This leaves us with several unknowns in our algorithm that we will be experimenting with, what the basic feature function is, what K to use, and how we find the other nodes we want to include.

5.1 Features

For the basic feature function, we have four features that we are working with:

1. **history-feature**: an indicator for whether the edge was in the graph at the previous timestep.

2. **Adamic-Adar**: it is defined as $Adamic - Adar(u, v) = \sum_{w \in \Gamma(u) \cup \Gamma(v)} \frac{1}{\Gamma(w)}$ where $\Gamma(u)$ is the set neighbors of u.

The third and fourth are modifications of the local clustering coefficient.

3. The third feature is $f(u, v) = \frac{1}{|N|} \sum_{i,j \in N} E(i, j)$ where $E(i, j)$ is an indicator function for if there exists an edge between i, j and $N = \{u\} \cup \{v\} \cup (\Gamma(u) \cap \Gamma(v))$.

4. The fourth feature is the same as the third, only with $N = \{u\} \cup \{v\} \cup \Gamma(u) \cup \Gamma(v)$ this time.

There are reasons that we chose each of these features. The first is a relatively simple feature to include - you can't get a much simpler feature for two nodes than whether there is an edge between them. The second is a feature we found in [2] which showed that it could be useful. The third and fourth are custom features which we designed to attempt to detect how strong of a community the two nodes are in, similar to the features devised in [3].

5.2 Selection algorithms

For choosing the K nodes to draw information from, we are simply going to experiment with different values. We chose to experiment with $K = 0, 1, 3, 5, 7, 10, 20, 40$ for our algorithms. $K = 0$ includes only the feature set between u, v.

We have several strategies for choosing the K nodes to include, and they are as follows:

1. **Random:** Choose random nodes from the graph as the K other nodes.

2. **Neighbors:** Choose nodes randomly again, but prioritize the neighbors of our edge nodes.

These would be simple methods to serve as a baseline for our more advance selection algorithms:

3. **Adamic Adar:** Sort each node by it's Adamic Adar value (specifically the sum of the Adamic Adar values with the two nodes in our edge) and choose the K nodes with the highest such value. Normally computing Adamic Adar for every pair would be quite expensive, but since Adamic Adar is only non-zero if the distance between the nodes is 2 or less, we just need to consider the nodes that are at most two steps away from the source and target nodes, which is much faster.

4. **Degree:** Sort the nodes by their degree and use the nodes with highest degree. This will give us the same nodes for every edge, but since the features to those nodes will be different for each edge we try to predict, it should still be useful.

6 Experiments

6.1 Evaluation Metrics

Raw predictive performance on link prediction for most graphs is very bad in an absolute sense, regardless of methodology or algorithm. This is because most graphs are very sparsely linked (Our graph has a of 0.00012433 $edges/vertex^2$, or fill). Even a tiny false positive rate results in an enormous problem.

The best method for evaluating performance on an extremely imbalanced classification task is AUC under the ROC curve. However, it is still unfair to compare performance between graphs strictly by AUC as link prediction performance is largely dependent on fill. For example, a method that achieves 0.8 AUC on a fill of 0.1 may (and does in practice) achieve 0.6 AUC on a fill of 0.01. We will, therefore, present both the AUC and fill. Since we do not have a rigorous way of combining the two numbers into one score, will keep them separate.

6.2 Methodology

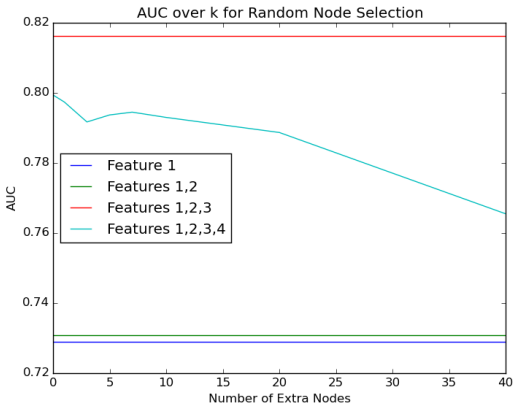
We wanted to experiment with our our choices of the base features, K, and the node selection algorithm, so we decided to try many different possibilities. We first took base feature sets comprised of only feature 1, then features 1 and 2, then features 1,2, and 3, and then finally we used all four features for a total of 4 feature sets. We then paired each one with each choice of K and each of our 4 node selection algorithms and ran every combination of the hyper-parameters.

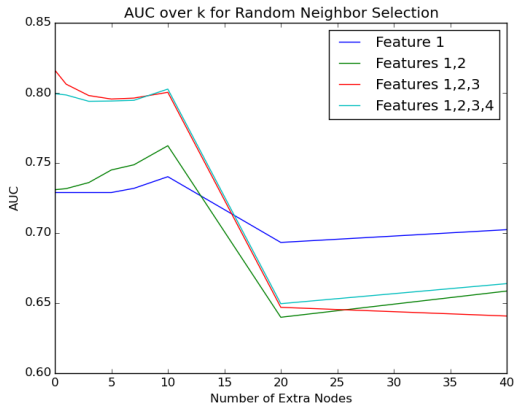
In order to test these combinations though, we need a graph to train on. We got this by using the first two quarters of 2007 as the $G_{train}[t_0, t'_0]$ and $G_{train}[t_1, t'_1]$ to train on, and we used the last two quarters of 2007 as the $G_{test}[t'_0, t'_0]$ and $G_{test}[t_1, t'_1]$ to test on. Since we are not submitting a single model as our best model with an established accuracy, we don't need a full test set, so we only need the train and development sets we describe above.

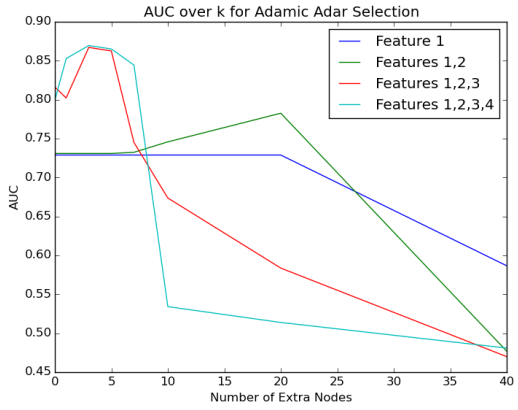
There is a problem with this approach though - our graphs are so sparse that any reasonable classifier would simply say that there were no edges, which while being very accurate, is not very interesting. In order to alleviate this problem, we sampled train and test edges from the graphs in a ratio of 1 positive example to 20 negative examples (the original graph has a ratio of about 1:10000, which is much too sparse for our purposes). We maintain the 1:20 ratio for all experiments conducted in this project. While 1:20 is arbitrary, our goal is to compare the relative performances of different approaches to link prediction, so we fixed the sampling at a level where these differences became evident. At 1:10000, for example, all methods score 0.5 AUC. Sampling at a particular ratio makes our AUC evaluation metric not indicative of performance on the actual task, but it still serves to differentiate the strength of different methods without changing the task at hand. We justify our decision based on the fact that other papers also doing link-prediction either use very dense networks with a positive-to-negative edge ratio less skewed than 1:20 or avoid discussion of how link-prediction tasks can be applied to networks with low fill. [2]

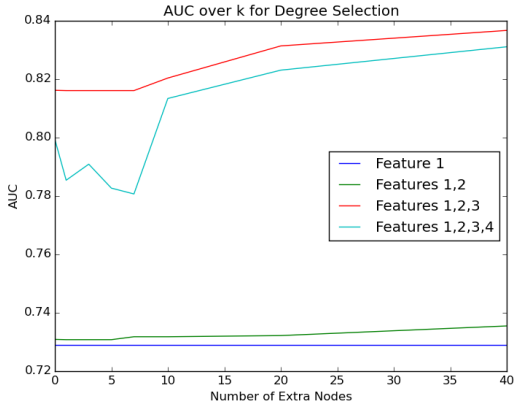
6.3 Results

The graphs show the results of our experiments. Each graph represents a single node selection algorithm, and each line on each graph represents one of our feature sets. The graphs chart how the AUC changes as we modify the number of extra nodes, which we call K.









7 Discussion

Looking at the plots, we see that many of them behave approximately how we would expect. For example, for the random selection algorithm, the AUC is essentially unaffected by changing K until we get to using all 4 features, which means that plugging in random extra nodes doesn't help, which is what we would expect. The 4th feature appears to be minimally meaningful, likely because it is too noisy, and results in a downward trajectory from over-fitting on the training data.

Looking at the graph when we picked random neighbors, we see that even this relatively simple selection procedure allows us to get some gains in performance. These quickly disappear when K increases, but that is about what we would expect : adding more information helps until the classifier starts over-fitting, at which point the performance decreases. Also when K exceeds 20, many pairs of nodes do not have enough neighbors to contribute information.

The Adamic-Adar selection algorithm performs the best. While only using the first or second feature, we get some increase in performance when we get up to $K = 20$, but not much. The main increase comes when we add the third feature though - this gives us a very significant $0.05-0.07$ increase to our AUC. This seems to indicate that only having the third feature for one node doesn't tell us much, but if we get that data for multiple nodes, even if it is just 3 other nodes, it becomes much more useful.

In both the random neighbor selection and Adamic-Adar neighbor selection approaches, we have the characteristic hump curve that appears when adding more information causes us to perform better until we start over-fitting, which is exactly what we expect from this model.

The degree distribution is not what we would expect. We would ordinarily expect to have the characteristic hump curve as above, but we just see the performance increasing. It will most-likely eventually go down if we kept adding more nodes, but it doesn't do so nearly as soon as we would have thought. This seems to indicate that as we include more nodes, we start including nodes that are important to our edge (which becomes even more likely due to the fact that these are the nodes with highest degree), which allows us to predict whether it has an edge better. It is also interesting that we can simply include a constant set of nodes (those with highest degree), and it will still increase the performance. This tells us that some nodes are important to whether or not a significant portion of nodes is connected, which is a very interesting result.

As for which features were helpful, we see that in each graph, the curves for the feature sets $\{1\}$ and $\{1,2\}$ are roughly the same, and similarly the curves for feature sets $\{1,2,3\}$ and $\{1,2,3,4\}$ are very similar. This tells us that adding features 2 and 4 to the feature vector doesn't give us much more information than we already had. On the other hand, since the curve for $\{1,2\}$ is usually very different from the one for $\{1,2,3\}$, and since

the curve for the feature set $\{1\}$ is so much higher than a 50% line (the line with random selection - ie. no features), we can tell that features 1 and 3 tell us a significant amount of information, which is interesting to know. It may be worthwhile to explore why 1 and 3 have such a large impact when 2 and 4 don't (eg. do they just not provide useful information, is the information already included in the other features, or is it something else?), but that will have to wait for a future project.

8 Future Work

One way to improve upon these methods is to use a standard, one-directional recurrent neural network (RNN). This will allows us to compute each probability of a link using the full history for a given node. We start off by ordering the nodes arbitrarily, and then we run through each node in the order we just made with our RNN, using the history from the previous node to compute our current node's probability and the history to pass into the next node. Theoretically, since the history is the result of the computation on all the previous nodes, this should allow us to compute the probabilities we wanted without making any approximations like we did in our first architecture. Of course, in practice, things are not so simple, and there are many issues we would need to address, but that is an issue for another time.

Another direction could be to investigate other approaches to selecting the nodes to condition on. This should also likely be done at a finer time granularity so we can capture precisely what actions lead up to the formation of a given link.

9 Conclusion

Looking at the data we collected, we see that there were many cases where the addition of extra nodes into the feature vector increased the performance of our classifier, which is exactly what we hoped to achieve. We were attempting to show that removing the independence assumption from the edges yields additional information that is useful in predicting whether an edge will be in the graph in the future, and we seem to have done just that. It is important to note as well that these models are merely an initial, relatively crude attempt at making this work. There are so many other ways that this could be expanded (from RNNs to other selection algorithms to other base features to including the timing of the edges in a more meaningful way) that there still remain many avenues in which this idea can be explored. This means that the idea of removing the independence assumption from the edges could lead to much higher gains in performance than we were able to demonstrate here, which is a great possibility to look forward to.

References

[1] Zan Huang and Daniel Dajun Zeng. A link prediction approach to anomalous email detection. In *Smc*, 2006.

[2] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03*, 2003.

[3] Sucheta Soundarajan and John Hopcroft. Using community information to improve the precision of link prediction methods. In *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion*, 2012.

[4] Meeyoung Cha Bimal Viswanath, Alan Mislove and Krishna P. Gummadi. On the evolution of user interaction in facebook.