Soft-clustering dynamic networks with probabilistic tensor factorizations

Nick Foti

May 31, 2011

1 Introduction

Relational data has become abundant, examples of which include social networks, users ratings of items, gene interactions and correlations between financial instruments. This data is often represented as a network where the nodes represent the objects, e.g. users, items, genes, financial instruments, etc. and edges connect nodes that exhibit the relation. The edges may also have an associated weight indicating the strength of the relation. Most current work analyzing such data treats the networks as static objects. However, in most cases both the edges and weights change over time. For example friend relationships on Facebook are added and deleted, users may change their ratings of items over time, genes may interact with different strengths at different stages of the cell cycle and financial instruments exhibit dynamic correlations over time.

An important problem when analyzing relational data between nodes of a single type (e.g. a social network) is community identification (or clustering), finding sets of nodes that are more similar to each other than other nodes in the network. A related problem is co-clustering for networks with two types of nodes (e.g. users rating items), which aims to simultaneously cluster both types of nodes. There has been a lot of work on both clustering and co-clustering for static networks, however algorithms for dynamic networks have received little attention [4, 1].

In this work we develop probabilistic methods based on tensor factorization to cluster dynamic networks [3, 2]. The proposed methods solve both the clustering and co-clustering problems in their respective contexts. We develop inference algorithms and evaluate the performance of the methods on synthetic and real data sets.

2 Data

We consider both synthetic and real data to evaluate the effectiveness of the proposed models. The synthetic data is used to validate the learning algorithms and to provide a

dynamic network with a "ground truth" clustering to compare our learned clusters to. We consider real data from three application areas described below that represent both the clustering and co-clustering problems.

2.1 Synthetic Data

We consider two types of synthetic data. The first type is random data generated from the proposed statistical models. These synthetic data are used strictly to test the convergence of the learning algorithms and are not intended to represent meaningful clustering. The proposed models will be described in more detail below. For each model we generate a sparse third-order tensor with a specified number of non-zero entries (or a specified fraction of nonzero entries) per time. The number of latent factors is also specified a priori. We divide the non-zero entries of the random tensor into a training and validation set. As mentioned previously, we then use this data to test the convergence of our learning algorithms.

The second type of synthetic data we consider is a small hand-made network that evolves over time with known clusters. The data is snapshots of a 16 node network at five time stamps. Figure 1 depicts the network at the different time stamps. There are obvious clusters at the various time points, some of them constant over time, others going in and out of existence.

2.2 Real Data

The first real data set we consider is the World Trade Web network. This network contains 196 contains corresponding to countries and directed edges between countries such that an edge from country i to j indicates the amount of U.S. dollars exported from country i to country j. Each edge has an associated time stamp in the years 1948 to 2000. Note that the network is directed and that nodes are not necessarily present at each year. The goal with this network is to find clusters of countries over time and so represents the clustering problem. Table 1. reports some useful statistics of the world trade web network.

The second real data set we consider is the evolving network of correlations between the monthly returns of equities from the S&P 500 stock index from January 2005 to December 2007. These dates were chosen as it produces a manageable size data set and represents a more or less normal period for the market. The data was constructed by obtaining the daily closing prices of the stocks from the S&P 500 using the DataStream service. The prices were converted to returns using the standard formula:

$$R_{i,t} = \frac{P_{i,t} - P_{i,t-1}}{P_{i,t-1}}$$

where $R_{i,t}$ and $P_{i,t}$ are the return and price of asset i at time t, respectively. The time series of returns are then smoothed with a 20-point moving average filter. A month of

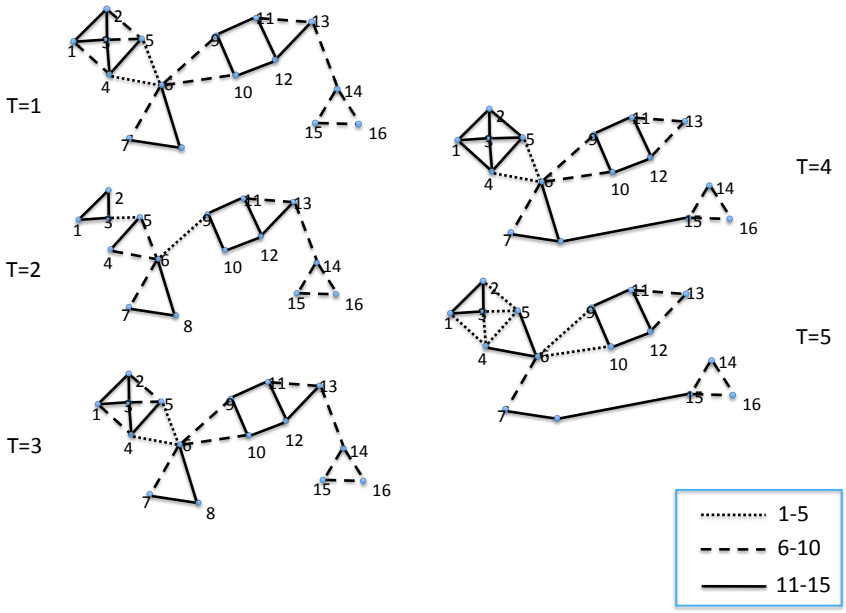


Figure 1: Small synthetic network with known clusters.

trading days is roughly 20 days, thus the choice of a window size of 20. We then compute the correlation matrix between equities for each month using the values of the smoothed series for each day of the current month. The hope is that the smoothing performed will mitigate any edge effects introduced by the finite window size. See Table 1. for some overall statistics. This network represents an instance of the clustering problem as the goal is to cluster stocks over time.

The last real data set we consider consists of authors publishing at ACM conferences over time. The data was mined from DBLP and the resulting data is a third-order tensor where the (i, j, l)'th entry contains the number of papers that author i published in conference j in year l. This tensor represents an evolving bipartite network where the goal is to find clusters of authors and conferences simultaneously and so represents the co-clustering problem. See Table 1. for general statistics. This data set is very large and even modestly sized versions of it were prohibitive to run our algorithms on, we thus leave analyzing this data to future work.

Table 1: General statistics of real data sets.

	# Nodes	**# timestamps**	**# Edges**	**Sparsity**
WTW	196	53	48336	2.37%
S&P 500	474	36	28768	0.36%
DBLP	655551 (auths), 5121 (confs)	52	2294930	1.315e-05%

3 Models and Learning

We develop two statistical models for the evolving network data considered. The first model posits that edges arise from a Normal distribution and is natural for networks where the observed edges may be positive or negative, for instance if modeling the deviation from a baseline, e.g. ratings. The second model posits that edges arise from a Poisson distribution and so is natural for networks with all positive edges that are integers, e.g. counts.

3.1 Gaussian Model

The normal model (NPTF) [7] posits that, given factor matrices $U \in \mathbb{R}^{K\times M}$, $V \in \mathbb{R}^{K\times N}$ and $W \in \mathbb{R}^{K\times L}$, the entries of the third-order tensor $\mathcal{X} = [x_{i,j,l}] \in \mathbb{R}^{M\times N\times L}$ are generated from a Normal distribution,

$$x_{i,j,l} \sim N(\hat{x}_{i,j,l}, \sigma)$$

where $\hat{x}_{i,j,l} = \sum_{k=1}^{K}(U_{ki} \odot V_{kj} \odot W_{kl})$ and σ is a constant variance ($\odot$ means element-wise multiplication). We place Normal priors on the entries of U and V with mean 0 and

variances σ_U and σ_V respectively. To model temporal dependence in the factors we make the columns of W follow an AR-1 process,

$$W_{:,1} \sim N(\mu_W, \sigma_W)$$
$$W_{:,l} \sim N(W_{:,l-1}, \sigma_W), \;\; l \in \{2, ..., L\}$$

Intuitively the U and V matrices indicate the degree to which the factors contribute to the connections that the nodes make. The matrix W indicates when factors are active and inactive. The use of the Normal distribution means that generate edge values may be positive or negative. Most network data is inherently positive so at first glance the Normal model might not seem to be a good idea. One reason to formulate the model is that some network data may be well modeled as a deviation from a baseline. In this case one would first subtract the mean value from all edges and then the Normal distribution is a natural distribution for the deviation (modeling this type of data is not a goal of this work, but it is important to notice). Even in cases where we only observe positive data the Normal model may be useful. Note that there is no constraint the the columns of U, V and W be orthogonal, so when observing all positive data there is nothing keeping the entries of U, V and W from all being positive. In this case we can interpret the learned columns as posterior probabilities of cluster membership and activity.

With the generative model specified we can turn our attention to inference. For simplicity we will resort to MAP estimation of the factor matrices U, V and W with the other parameters held fixed. This amounts to minimizing the following error function:

$$E(U, V, W, \mu_W) = \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{l=1}^{L} I_{ijl}(X_{ijl} - \hat{x}_{ijl})^2 + \frac{\lambda_U}{2} \sum_{i=1}^{M} ||U_i||^2 + \frac{\lambda_V}{2} \tag{1}$$

$$\sum_{j=1}^{N} ||V_j||^2 + \frac{\lambda_W}{2} \sum_{l=2}^{L} ||W_l - W_{l-1}||^2 + \frac{\lambda_W}{2} ||W_1 - \mu_W||^2 \tag{2}$$

We have derived and implemented both a batch alternating least squares (ALS) algorithm as well as a stochastic gradient descent (SGD) algorithm to minimize this error function. In more detail, the batch alternating least squares algorithm works by updating all factors in the U matrix for a fixed index i of the first dimension. The algorithm then updates all factors in the V matrix for a fixed index j of the second dimension and lastly all factors of W for a fixed index l of the third dimension. The algorithm is called ALS because updating the factors for a fixed index is a simple least squares problem on the subset of the data corresponding to the fixed index. The SGD algorithm updates the components of all factors for each observation separately. Figure 2 depicts the convergence of the ALS algorithm on random tensors drawn from the generative model and Figure 3

depicts the same for the SGD algorithm. The alternating least squares method is faster in Matlab for the data we consider and requires less tuning so later results will only consider this algorithm. However, the SGD algorithm could be more useful for very large data sets if implemented in C.

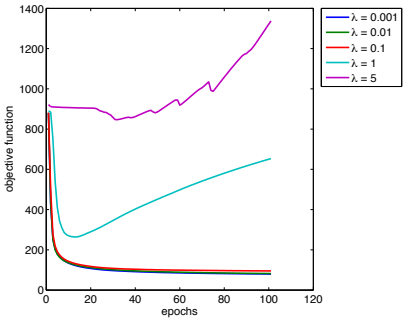
3.2 Poisson Model

The Poisson model (PPTF) [5, 6] assumes that given factor matrices U, V and W, the edge weights of the third-order tensor $\mathcal{X} = [x_{i,j,l}] \in \mathbb{R}^{M \times N \times L}$ are generated from a Poisson distribution with rate (mean) given as in the normal model. Specifically, we have

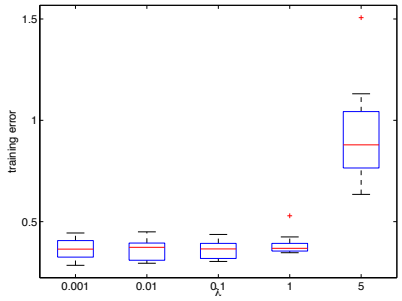
$$\begin{aligned} x_{i,j,l} &\sim \mathcal{P}(\hat{x}_{i,j,l}) \quad \hat{x}_{i,j,l} = \sum_{k=1}^{K} U_{ki} \odot V_{kj} \odot W_{kl} \\ u_{ki} &\sim HN(0, \beta_k^{-1}) \\ v_{kj} &\sim HN(0, \beta_k^{-1}) \\ w_l \,|\, w_{l-1} &\sim N(w_{l-1}, \sigma_w I) 1_{[0,\infty)}(w_l) \\ w_1 &\sim HN(\mu_w, \sigma_w I) \\ \beta_k &\sim \mathcal{G}(a_k, b_k) \end{aligned}$$

where we have placed half-normal priors on the entries of U and V. If y is a scalar random variable with a normal distribution, then $|y|$ has a half-normal distribution. Additionally, the columns of W again follow an AR-1 process, however, it is restricted to be greater than 0. Note that each factor in U and V has a different precision, β_k. We place a sparse Gamma prior on each β_k which has the effect of performing model selection for us as most β_k's will be forced to infinity.

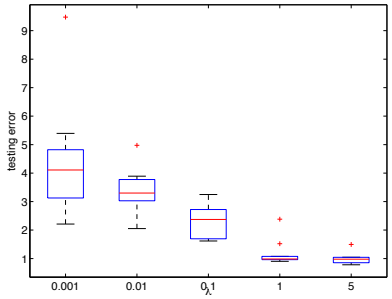
Having specified the generative model we will again resort to MAP estimation of the factor matrices. Our goal is thus to maximize the posterior probability of the factor matrices given the observed tensor. It turns out that the function we are maximizing has close connections with the generalized Kullback-Leibler divergence. We have derived and implemented a stochastic gradient ascent algorithm as well as a multiplicative update gradient algorithm to learn the MAP factor matrices. The SGD algorithm works the same as in the Gaussian case with a different objective function. The multiplicative algorithm is a first-order gradient method where the step-size is chosen to produce multiplicative updates REF. In practice the algorithm converges quickly. The SGD algorithm also is very hard to get to converge to a reasonable solution as it very often eliminates all but one factor on the first iteration and never is able to recover as seen in Figure **??**. Similarly to the learning algorithms for the Gaussian model above we only report results for the multiplicative algorithm as it was quicker to run and required less tuning.



(a)

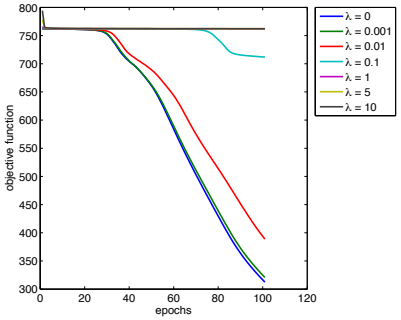


(b)

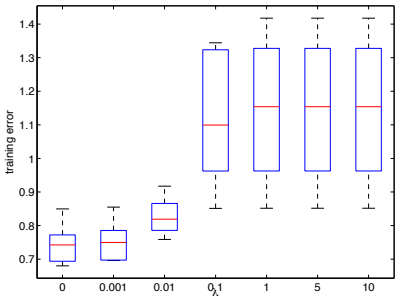


(c)

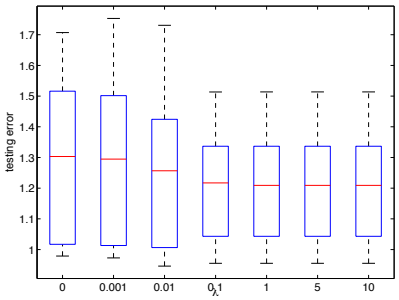
Figure 2: Convergence of ALS algorithm on random tensors. We depict the objective function versus epoch (number of times through all observations) (a), the distribution of the training errors (b) and the distribution of the testing errors (c). We see that the algorithm does converge to a minimum for small values of λ which is probably over-fitting the training data. For larger values of lambda we see the algorithm converge and then the objective function increases indicating that it is starting to over-fit the training data. The plots of training and testing error for various values of λ are what one would expect.



(a)



(b)



(c)

Figure 3: Convergence of SGD algorithm on random tensors. Similarly to the previous figure the objective function is converging, however we probably have not let the algorithm run long enough. Additionally, the training and testing error seem reasonable.

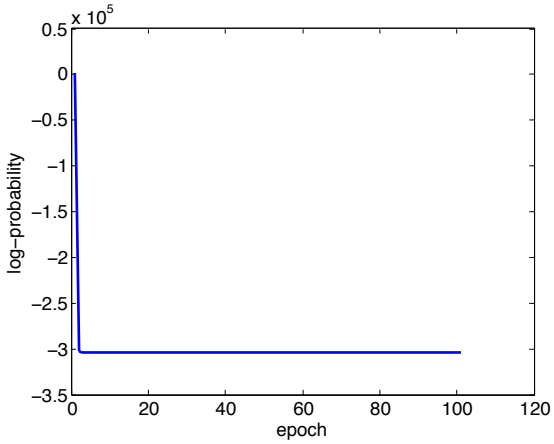


Figure 4: Convergence of objective function for Poisson SGD algorithm. The algorithm eliminates all but one factor and is never able to recover.

In addition to the Poisson tensor model, we also consider a special case of the model where we only observe one snapshot of the network. In this case the Poisson model corresponds to a non-negative matrix factorization REF. We can use the multiplicative gradient algorithm derived for the Poisson tensor model to learn the factors in this model. This special case performs soft-clustering of a static network and provides some hope that these factorization models will work for clustering tensors as well as providing a sanity check that the algorithms are converging.

4 Results

We conducted experiments using all three models described above. As mentioned before the DBLP data considered was too large to run any experiments in a reasonable amount of time. We also discovered that the S&P 500 data was not very amendable to the PPTF model and that the WTW data was not amenable to the NPTF model. The reason for this is that the values in the S&P 500 network are in $[0, 1]$ which a Poisson distribution will have trouble approximating. On the other hand, the WTW data contains very large positive integer values which the Normal distribution does not model well. Therefore, we applied the NPTF model only to the S&P 500 data set, and the PPTF model only to the WTW data. We were able to apply the static PPTF (i.e NMF) model to the toy, S&P 500 and WTW data sets, noting that the results should be taken in the context of the restrictions mentioned above.

Evaluation metrics for soft-clusterings and dynamic clusters are currently non-existent. In the case of soft-clustering what is done is a hard-clustering is formed and standard

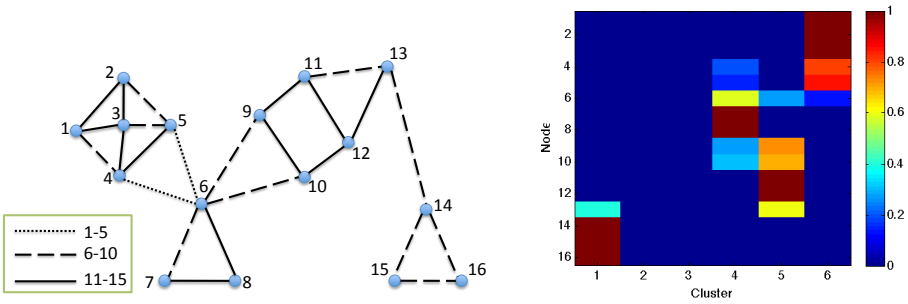
evaluation criteria for hard clusters are used. The problem of dynamic clustering is not widely studied and so there is currently no state of the art for evaluation. In this work we follow the current trend in evaluating our static model and to evaluate dynamic clusters we see how well we predict held out edges of the network.

4.1 NMF Model

We applied the NMF model (PPTF model restricted to one snapshot) to the first snapshot of the toy, S&P 500 and WTW networks. Figure 5 depicts the toy network considered and the resulting clustering. Note that nodes are assigned probabilities of belonging to a given cluster where nodes that are definitively in a cluster (e.g. nodes 1 and 2, or 14, 15 and 16) are given most mass to a single cluster. Nodes that have evidence of belonging to multiple clusters (e.g. nodes 6, 9 or 10) are given different masses proportional to their degree of membership in the given cluster. Additionally the number of clusters in the network was learned as clusters 2 and 3 were not used, i.e. no node had mass assigned to clusters 2 or 3. Table 2 shows some measure of how well the method performs on the toy and real data sets. Specifically it reports the number of clusters learned as well as the *modularity* of the clustering REF. Modularity is a measure of how well a clustering is grouping the nodes in a network and is basically a measure of how different the clustering is from a random clustering of the nodes. To measure the modularity we need a hard-clustering of the nodes which we create from our soft-clustering by assigning a node to the cluster with the highest probability. Typically values greater than 0.3 are deemed good for modularity so the results for the toy and S&P 500 networks are good. The result for the WTW is spurious as it appears that nodes with weak connections are being clustered together. This is probably an artifact of the dynamic range of the edge weights present and will need to be explored further. Regardless, the results of this simplified model are promising for probabilistic tensor factorization clustering models.

	Toy	**WTW**	**S&P 500**
$K_{\mathbf{eff}}$	4	13	11
Modularity	0.5850	-0.1022*	0.5883

Table 2: Number of clusters learned (K_{eff}) and modularity of clustering learned with static PPTF model. The negative value for the WTW seems to be an artifact of the data, however, the results for the toy and S&P 500 networks are very good as modularity values of 0.3 are considered good.



(a) (b)

Figure 5: The first snapshot of our toy network (left) and the resulting clusters learned from the static PPTF model (NMF model). Notice that nodes are given probabilities of cluster membership and that the number of clusters was automatically learned. The color of each cell in the right figure represents the probability that a node belongs to a given cluster.

4.2 NPTF Model

We applied the NPTF model to the S&P 500 network performing learning with the ALS algorithm as our implementation converged faster. First we investigate the behavior of the learning algorithms and how well the models are able to explain the data by looking at the errors. We first divided the data into a training set and testing set with 85% of the data used for training. The model has a few parameters which we collapsed to two parameters, a regularization parameter λ and the number of clusters K (as this model doesn't learn the number of clusters). We performed cross-validation to find *good* values using 5-folds on the training set (this still took about 2 hours). Figure 6 shows the mean cross-validation score (the value of Equation 1) with one standard deviation. We see the usual pattern of the error increasing as the value of regularization parameter. Figure 7 shows the value of the objective function on the test set with a model trained on the entire training set. We see the same behavior as in the cross validation with increasing error with larger values of λ and notice that the values of the objective function (error) is higher for the test data as expected. Figure 8 shows the root-mean-square-error (RMSE) of the test data which decreases to a seemingly good value of 0.25 for larger values of λ. However, more investigation needs to be performed to determine if this is meaningful.

Next, we explore the actual clusters that are learned by the model. Figure 9 shows the factor value of each node for factor 7. We see positive, negative and very small values and we can interpret the positive values as a node belonging to this cluster and small values as a node not belonging to the cluster. Negative values are harder to interpret but a naive interpretation is that the node is opposed to being in this cluster. Further inspection as

to what nodes have large relative positive values shows that the cluster is rather spurious with equities that don't really seem to have anything in common. We can also explore the temporal patterns of the learned clusters as shown in Figure 10. We see some interesting temporal patterns as some of the learned factors are periodic, some are constant and others decrease over time. So we see that the NPTF model is good at reconstructing the data, but our results have not shown meaningful clusters are learned and further work will need to be conducted. For instance, we should compare these results with a baseline method such as predicting the held out edge weights to be the mean edge weight of the training data.

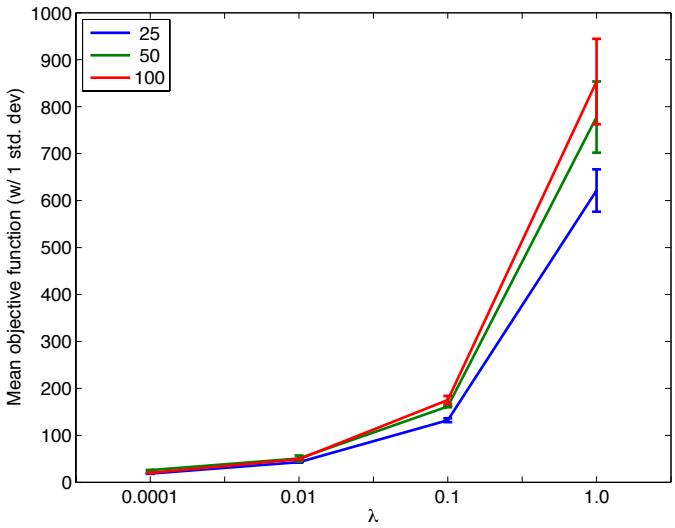


Figure 6: Value of NPTF objective function on validation folds (mean and one standard deviation) for 5-fold cross validation with various values of λ and K.

4.3 PPTF Model

We applied the PPTF model to the WTW network using the multiplicative gradient descent algorithm. As with the NPTF model there are a few parameters in the PPTF model, however, in our experiments we only vary the number of initial clusters, K_0, as the model is pretty robust to the other parameters. As with the NPTF model we perform 5-fold cross validation to determine a good value for the number of initial clusters where we run the multiplicative gradient descent algorithm for 100 iterations. Figure 11 shows boxplots of the distributions of of the objective function (negative log-probability) of the PPTF model. We see that increasing the number of initial clusters noticeably decreases the value of the objective. Setting K_0 forces the algorithm to use too few factors, however choosing

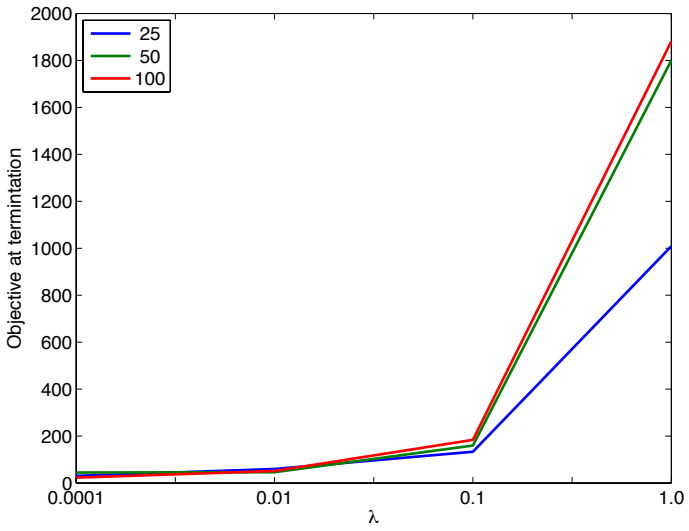


Figure 7: Value of NPTF objective function on validation fold for 5-fold cross validation with various values of λ and K.

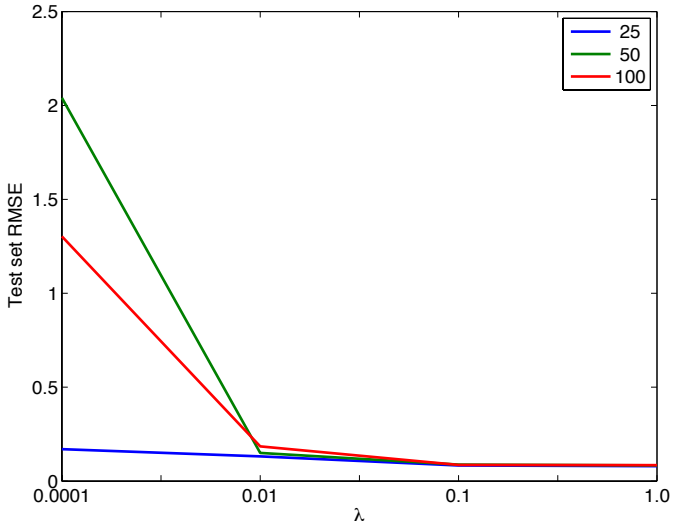


Figure 8: RMSE of test set for various values of λ and K.

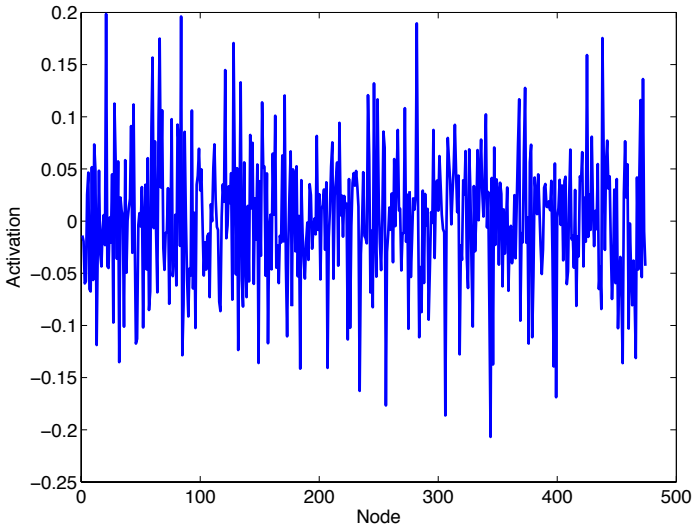


Figure 9: The seventh learned factor for the nodes in the NPTF model on the S&P 500 data. Nodes with large activations are part of this cluster while small or negative activations are not.

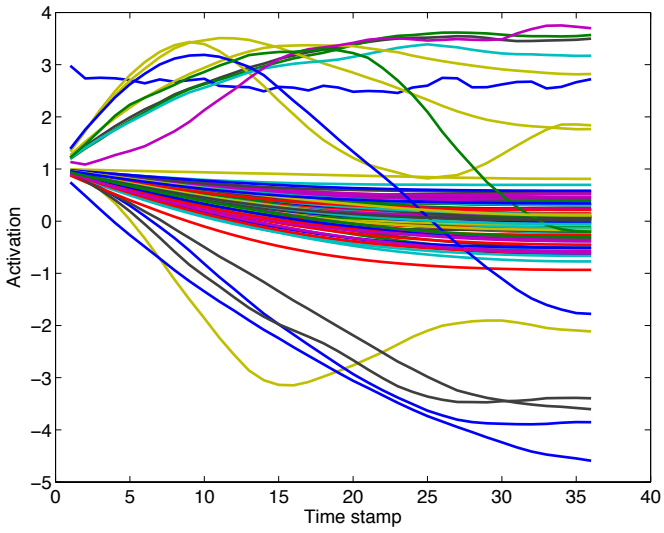


Figure 10: The temporal signatures for all learned factors for the NPTF model on the S&P 500 data. We see some interesting temporal behavior.

K_0 too large can cause the algorithm to overfit the data because it has too many degrees of freedom. We did compute the KL divergence that corresponds to the PPTF model, however the results were inconclusive at this time and further work needs to be performed on the sensitivity to the learned clustering on the starting value of K_0.

We then consider the actual clusters learned by the model. Figure 12 shows the activation of each node for the second cluster. In this case all values are positive and only a few are much larger than the others. Unfortunately, again the nodes with large activations correspond with countries that don't seem to have anything to do with each other. Figure 13 shows the temporal signature of each cluster and in this case we see that all clusters increase their influence over time. This is probably an artifact of the learning algorithm not converging fully to a good solution. So again the results are promising in that the model learns sparse clusters, but the reported results do not indicate that the learned model is explaining the data well nor learning meaningful clusters.

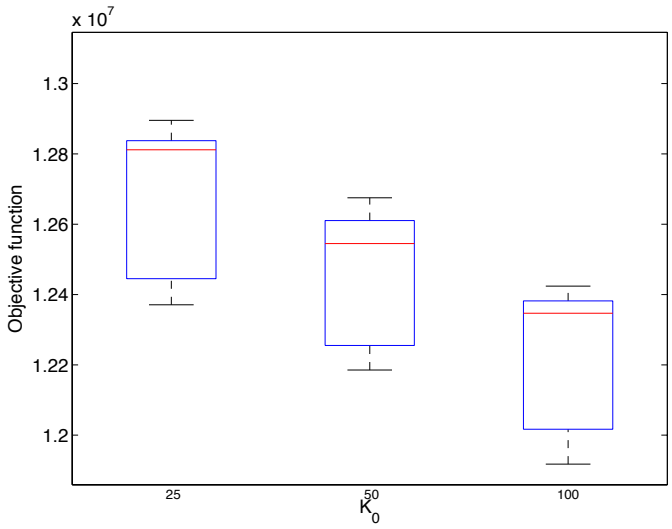


Figure 11: Value of objective function for PPTF model for the WTW data for various values of the initial number of clusters.

5 Conclusions

In this work we proposed two models to perform soft-clustering of time-evolving networks, one that treats the edge weights as Gaussian distributed and the other that treats the edge weights as Poisson distributed. We have implemented learning algorithms for both models and applied them to synthetic data and real evolving networks. We find that the Gaussian model is able to learn latent clusters that explain the data well, but the clusters

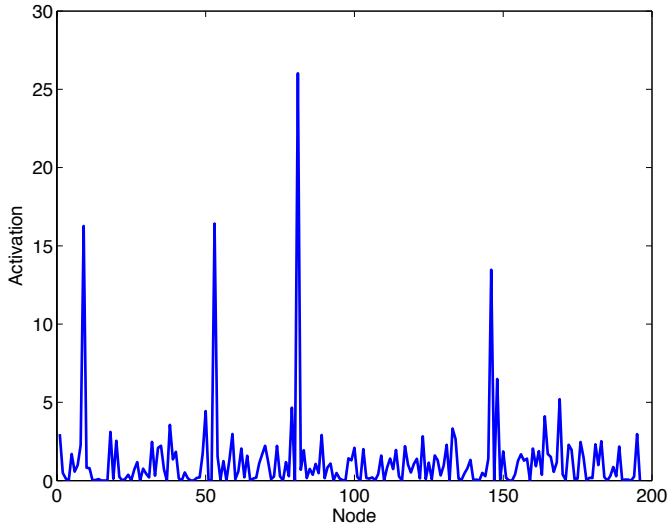


Figure 12: The second learned factor for the nodes in the PPTF model for the WTW data. Nodes with large activations are part of this cluster.

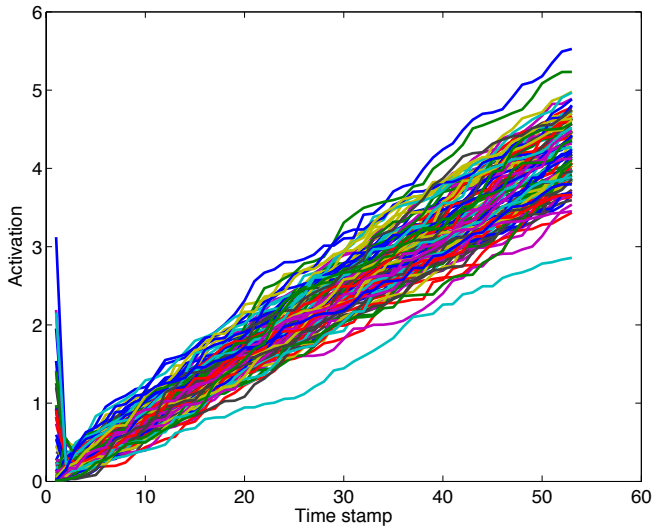


Figure 13: The temporal signatures for all learned factors of the PPTF model on the WTW data. We see some interesting temporal behavior.

at first observation are not meaningful and the cluster indicators are hard to interpret. The Poisson model learns sparse, easy to interpret factors, however, the model did not seem to explain the real data we applied it to very well. Some of these problems may be mitigated by a more sophisticated preprocessing of the raw data because for this work we attempted to keep all preprocessing as simple as possible.

In future work we will augment the NPTF model with explicit positivity constraints to make the learned factors easier to interpret. Sparsity could also be explicitly incorporated into the model to enforce learning small clusters. Additionally, more work needs to be done exploring the convergence of the algorithms and local minima. There is still hope that the Poisson model will work because of the static NMF's success. More work needs to be done on the sensitivity of the algorithm to the initial number of clusters and on the algorithm's convergence in general to local minima. Additionally, though the toy network is very simple with obvious clusters evolving over time, there may just not be enough data to learn the sophisticated latent variable models. Therefore, it is very important to develop a *non-trivial* time-evolving network with known clusters over time that the models can learn. Lastly, a fully Bayesian model should be developed to alleviate any sensitively to parameter values. Learning can be performed in the fully Bayesian setting either by Markov chain Monte Carlo or with variational methods.

In closing, the project has been successful allowing us to explore an idea we have had for a while. Though the models did not deliver all they were supposed to, the project allowed the full models to be fleshed out and logical gaps to be filled as well as some simple learning algorithms to be implemented. However, the most important results of this work were the unanticipated new problems that arose during the work and that were discussed above. We look forward to solving these problems and pushing work in the area of clustering dynamic networks forward.

References

[1] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge Discovery and Data Mining*, pages 269–274, 2001.

[2] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data*, 5(2):Article 10, 27 pages, February 2011.

[3] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.

[4] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2001.

[5] I. Psorakis, S. Robers, and B. Sheldon. Soft partitioning in networks via bayesian non-negative matrix factorization. 2010.

[6] V.Y.F. Tan and C. Fevotte. Automatic relevance determintation in nonnegative matrix factorization. 2009.

[7] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of SIAM Data Mining*, 2010.