Modeling Network Data with Probabilistic Programming and Probabilistic Graphical Models

Peter Racioppo, Tianyu Zhang, Jiapeng Wan, Andrey Storozhenko

June 17, 2020

Abstract

In this paper, we investigate the use of probabilistic graphical models and probabilistic programming to model social network data from Facebook. We use a Markov Chain Monte Carlo algorithm to sample from an Ising model on a graph, demonstrate some algorithms for predicting the labels of vertices given information about their neighbors, and implement a maximum likelihood estimation of Ising model parameters. We compare these results with a greedy combinatorial algorithm and inference using the probabilistic programming language Problog, and we discuss the implementation of Problog.

Keywords: *Probabilistic Graphical Modeling, Social Network Analysis, Bayesian Networks, Markov Random Fields, Markov Chain Monte Carlo.*

1 Introduction

Many real-world situations involving interaction between discrete agents can be effectively modeled with graphical models. Network problems commonly arise in fields as diverse as biology, sociology, economics, and computer science. The advent of the internet and the widespread use of social media has made social network modeling an important big data problem. Probabilistic graphical models (PGMs) such as Markov Random Fields and Bayesian Networks, which use a graph structure to model conditional dependence between random variables, are among the most promising approaches to modeling the enormous and complex data sets characteristic of modern social media. Markov Chain Monte Carlo sampling techniques have made sampling from complex and high-dimensional probability distributions on PGMs increasingly tenable. Probabilistic programming leverages these sampling techniques to provide a framework for performing inference on complex graphical models. [1]

In this paper, we attempt to solve the following problem: Suppose we are given an adjacency matrix and a set of vertex labels, such that each vertex has exactly one label. Now, suppose we are given the labels of some subset of the vertices, but the labels of some other subset are unknown. Can we predict the labels of these unknown vertices? For instance, suppose Facebook users are divided between Democrats and Republicans. Can we predict whether a particular set of users is most likely to be majority Democrat from the party affiliation of the users with whom they are friends?

2 Background

Markov Random Fields
Markov Random Fields (MRFs) are undirected graphs, with node connections satisfying the Markov property, where any random variable is conditionally independent of its neighbors, given all other variables. Formally, given an undirected graph $G(V, E)$, a set of random variables

$X = (X_v)_{v \in V}$, indexed by the vertices V, form an MRF in the edges E if for any $u, v \in V$ such that $u \neq N(v)$ we have $\Pr(X_u \mid X_v,\ X_{V \setminus \{u,v\}}) = \Pr(X_u \mid X_{V \setminus \{u,v\}})$, where $N(v)$ denotes the set of vertices adjacent to vertex v. MRFs are equivalent to Gibbs distributions, as the Hammersley-Clifford theorem establishes, and have applications in statistical physics and computer vision. The Ising model, a pair potential MRF from statistical physics, has probability distribution over a graph I given by $p(I) = \frac{1}{Z}\exp(\sum_s \alpha_s I_s + \sum_{t,s} \beta_{ts} I_t I_s)$, where I_s is the binary state of vertex s (spin up or spin down), Z is a partition function, $\beta_{-t} = \beta_t$ and $\beta_{t-s} = 0$ unless t and s are neighbors. Let I_s denote the state of vertex s and let I_{-s} denote the state of all other vertices. The conditional distribution for an individual vertex is $p(I_s | I_{-s}) = p(I_s | I(N_s)) \propto \exp(\sum_s \alpha_s I_s + \sum_{t,s} \beta_{ts} I_t I_s)$. Here, α represents an external field and β represents the interaction between adjacent vertices. [2]

Bayesian Networks

Bayesian networks, in contrast, are directed acyclic graphs (DAGs), which can be used to represent conditional dependencies in a hierarchical manner. In particular, let $G(V, E)$, be a DAG, with random variables $X = (X_v)_{v \in V}$, indexed by the vertices V. We say that G is a Bayesian network if the probability density function of any $v \in V$ is conditionally dependent only on its parents (this is a local Markov property). Thus, the joint probability distribution of a subset of n random variables $X_1, ..., X_n$, which is given by the product rule as $P(X_1 = x_1, ..., X_n = x_n) = \prod_{v=1}^{n} P(X_v = x_v \mid X_i = x_i,\ \forall i \neq v)$, reduces to $\prod_{v=1}^{n} P(X_v = x_v \mid X_i = x_i,\ \forall X_i \in \mathrm{Pa}(X_v))$, where $\mathrm{Pa}(X_v)$ denotes the set of parents of X_v. [1, 3]

Markov Chain Monte Carlo

Markov Chain Monte Carlo algorithms are commonly-used tools for sampling from hard-to-sample probability distributions. Given a target distribution $\pi(x)$, and a starting state X_0, we form a Markov chain of approximate samples $X_0, X_1, ..., X_T$ from $\pi(x)$. In practice, we cannot draw X_0 from the target distribution, and it may be randomly set. However, under mild constraints, the distribution of the Markov chain will approach the target distribution as the number of samples goes to infinity. It may be difficult to determine how quickly the Markov chain will converge. Typically, a burn in length B is set such that the first B samples from the Markov chain are discarded. If we wish, for example, to compute the integral $I = E_\pi[h(x)] = \int_X h(x)\pi(x)$ we can estimate I by $I \approx \frac{1}{N}\sum_{i=1}^{N} h(X_{B+i})$. $[4-7]$

PageRank

Consider the graph $G = \langle V, E \rangle$. Given a vertex x, denote the set of vertices with a directed edge from x as out(x) = $\{w \in V | x \to w\}$, and the set of vertices with a directed edge to x as $\mathrm{in}(x) = \{y \in V | y \to x\}$. Let $\pi(x) = \sum_{y \in \mathrm{in}(x)} \pi(y)/|\mathrm{out}(y)|$. This recursive definition of π states that a node will have a higher value of π if the nodes with edges directed into it have higher values of π and if these edges have few outgoing edges. Thus, π can be thought of as a measure of the total flow into a node, or the node's "influence." We can sample from this distribution by defining a Markov chain transition probability $K(y, x) = 1/|\mathrm{out}(y)|$, since in this case $\sum_{y \in V} \pi(y)K(y, x) = \pi(x)$. If any node has no edges, we must modify this transition probability in order to ensure that the MC is ergodic. We can do this by adding a small probability of jumping between unconnected nodes. The probability of visiting a node in a Markov chain with this transition probability converges to the stationary distribution of the Markov chain as the length of the walk goes to infinity. The stationary distribution can in fact be cheaply computed with a simple power iteration method. [5]

Convergence of Markov Chains

The convergence of an irreducible, aperiodic Markov chain is bounded by the second largest eigenvalue modulus λ_{slem} of its kernel according to the Diaconis-Hanlon bound: $||\nu K^n - \pi||_{TV} \leq \sqrt{\frac{1-\pi(x_0}{4\pi(x_0}}\lambda_{slem}^n$. We can in turn bound λ_{slem} using some measures of graph connectivity. Let E be the set of edges in graph G, K the corresponding Markov chain kernel, and

Γ_{xy} a path between vertices x and y containing each edge at most once. The bottleneck of G, the edge in the graph with maximum stochastic flow, is defined as $\kappa = \max_{e \in E} \sum_{\Gamma_{xy} \ni e} \gamma_{xy} \pi(x)\pi(y)$, where $\gamma_{xy} = \sum_{<s,t> \in \Gamma_{xy}} \frac{1}{\pi(s)K(s,t)}$. By Poincare's inequality, $\lambda_{slem} \leq 1 - \kappa^{-1}$. Another bound is given by the conductance, which is a measure of how "well-knit" the graph is. Suppose the vertex state space is partitioned into two disjoint subspaces, $\Omega = S \cup S^c$. Let $\pi(S) = \sum_{x \in S} \pi(x)$, $K(S, S^c) = \sum_{x \in S} \sum_{y \in S^c} K(x, y)$, and $Q(S, S^c) = \sum_{x \in S, y \in S^c} \pi(x) K(x, y)$. The conductance is then defined as $h = \min_{S:\pi(S) \leq \frac{1}{2}} \frac{Q(S,S^c)}{\pi(S)}$. Cheeger's inequalities bound λ_{slem} in terms of the conductance: $1 - 2h \leq \lambda_{slem} \leq 1 - h^2/2$. [5] For diagnostic purposes, we implemented functions to compute the bottleneck and conductance of an arbitrary graph.

Markov Clustering

We also implemented the Markov Clustering (MCL) algorithm, a fast, unsupervised clustering algorithm which simulates stochastic flow through a graph using two operations called expansion and inflation. In the expansion step, we take the power of a stochastic matrix, simulating random walks of higher length through the graph. In the inflation step, we take the power of the matrix element-wise and renormalize the columns. The effect of this operation is to weaken weak connections and strengthen strong ones, which tends to break the graph into separate star graphs. [8]

3 Technical Contribution

Preprocessing the Data

We used anonymized data collected from survey participants on Facebook and published online by the Snap Laboratory at Stanford University [9]. This dataset encodes 4,039 user profiles as vertices in an adjacency matrix, with 88,234 edges between "friend" pages. This is a very sparse matrix, with only about 0.5% of the possible edges appearing in the adjacency matrix.

To assign labels to the data, we simulated an Ising model on the adjacency matrix using Gibbs sampling, with $\alpha = 0$. (Note that this process can be generalized to more than two states with a Potts model.) Gibbs sampling proceeds as follows: given a sample distribution x of vertex labels, we produce a proposal distribution x' by randomly choosing a vertex and flipping its state. We then compute an acceptance probability $\alpha(x'|x) = \min(1, \frac{\pi(x')}{\pi(x)})$, accepting the proposal distribution with probability α and keeping the old state otherwise. The distribution generated by this process will converge to the Ising distribution as the number of iterations goes to infinity. Starting from a white noise distribution, we set a burn-in on the order of the number of vertices in the graph, to ensure our sample is sufficiently close to the target. The result of simulating an Ising model with 4,039 sampling iterations on the Facebook graph, with $\beta = 0.1$, is shown in Fig. 1.

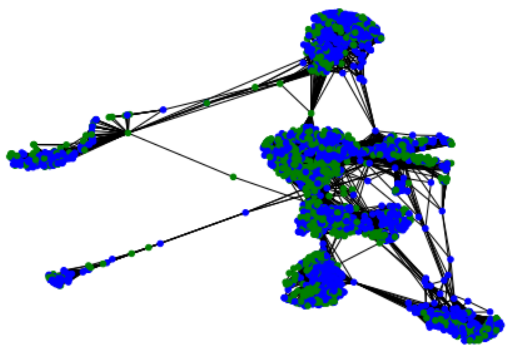

Figure 1: Facebook Data with labelings (blue and green) generated by an Ising model with $\beta = 0.1$.

Inference on a Graph Ising Model

Now, if we assume that we know the parameters of the Ising model that generated a distribution of labels, we can predict the labels of an unknown subset of the graph, using the labels of the vertices in their neighborhood. The simplest version of this problem is to predict the label of a single vertex. In this case, the most probable state of a vertex is simply the most common state among its neighbors. In 10,000 trials, this algorithm predicted the correct label in 85.5% of trials for $\beta = 0.5$, 79.5% for $\beta = 0.2$, 56.5% for $\beta = 0.1$, and 55.6% for $\beta = 0.001$. As expected, as the temperature increases, the prediction accuracy approaches the 50% baseline for a random guess. That it remains above 50% can probably be attributed to the non-random structure of the graph.

Inference on a Subset

Now suppose, more generally, that for some subset S of the graph, we are given the labels of all the neighbors of the vertices in S, denoted $N(S)$, but none of the labels in S itself. Because of the Markov blanket property, we can ignore the states of all vertices other than those in S. If the number of vertices in S is small, we can simply enumerate the $2^{|S|}$ possible states and add their Hamiltonians to compute the partition function (in practice, we approximated the partition function with a Monte Carlo sample). If we are interested in comparing probabilities (for example, to determine which label is more common in $|S|$) there is no need to compute the partition function, as Z cancels in the log-likelihood. We considered the problem of predicting which label in $|S|$ is more common. The exact MLE solution requires $O(2^{|S|})$ computations, so we approximated the solution by instead comparing the probabilities of *all* states in S being spin up or spin down. This simple method works surprisingly well on small subsets, when the interactions between S and $N(S)$ tend to be more significant than those within S. In 100 trials with $|S| = 3$, the predicted label was correct in 85.0% of trials for $\beta = 0.5$, 83.0% for $\beta = 0.2$, 73.0% for $\beta = 0.1$, and 62.0% for $\beta = 0.001$. We compared this method with a greedy combinatorial algorithm which iteratively finds the $q \in S$ with the minimum possible pair potential, updates the state of q, and adds q to $N(S)$. The accuracy of this algorithm appears to be comparable.

Inference with an Incomplete Neighborhood

Now suppose that in the previous example we are given only a subset of the labels in $N(S)$, which we will call $N_k(S)$. In this case, our Markov blanket assumption no longer holds, and the inference problem becomes much more difficult. We therefore tested the following Monte Carlo approach to predict the majority label in S: use the Gibbs sampler to simulate an Ising model, but at each step only flipping the states of the vertices outside of $N_k(S)$. After a burn-in period, we count the number of worlds in which $N(S)$ has a majority label of 1 or -1. After some large amount of iterations, we can the number of worlds in which each state occurred as an estimate of their respective probabilities. In 100 trials with $|S| = 5$, and the labels known for only 75% of the $N(S)$, the predicted label was correct in 87.0% of trials for $\beta = 0.5$, 61.0% for $\beta = 0.2$, 57.0% for $\beta = 0.1$, and 52.0% for $\beta = 0.001$. (Note that these numbers are quite variable, as they depend on the randomly chosen S.

Maximum Likelihood Estimation of Ising Model Parameters

In the previous section, we assumed that we knew the parameters of the Ising model and tried to predict vetex labels. Now we consider the inverse problem. Given the set of training data (vertex labels) for a subset S of the network, we compute the maximum likelihood estimate (MLE) of the Ising model parameters. Consider a probability density function $f_\theta(x) = \frac{h_\theta(x)}{z(\theta)}$. The log-likelihood ratio is $L(\theta) = \log(\frac{f_\theta(x)}{f_\phi(x)}) = \log(\frac{h_\theta(x)}{h_\phi(x)}) - \log(\frac{z(\theta)}{z(\phi)})$. In (Geyer, 1991), the last term is approximated as $\frac{z(\theta)}{z(\phi)} \approx \log[\frac{1}{n}\sum_i(\frac{h_\theta(X_i)}{h_\phi(X_i)})]$, where X_i is an MCMC sample from the Ising model [10, 11]. Using this approximation and our Gibbs sampler, we wrote a function to compute the approximate log-likelihood ratio of any two parameters β_1 and β_2. We can set β_2 to some fixed value and perform a line search on β_1 to maximize the log-likelihood. We can then set β_2 to β_1 and repeat the search, centered on this improved estimate. A drawback of this algorithm is that it is very computationally expensive, because it requires that we sample from a separate Ising model for each possible value of β. Running the algorithm many times to determine its accuracy was therefore not feasible in this paper. However, individual trials seemed to show promising results (see, for example, Fig. 2).

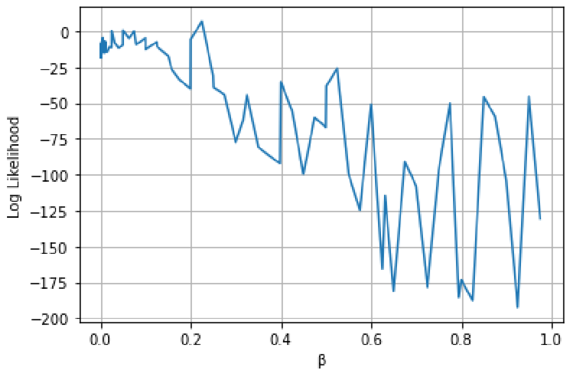

Figure 2: Single trial of maximum likelihood estimation of β in Ising model. In this example, $\beta = 0.3$ and $\beta_{MLE} = 0.22$, where we start from the estimate $\phi = 0.1$.

Inference in Problog

Intuitively, it may be possible to predict unknown vertex labels using only the most important vertices in a neighborhood. In this section, we use Problog to predict the label of a vertex $L(v)$, given the labels of its three highest-PageRank neighbors $L(N_3(v))$. We assume that the influence of each neighbor decreases exponentially as their PageRank decreases. We can set the predicted $L(v)$ to be the most common label among the neighbors of v by ensuring that the conditional probabilities of the vertex given each neighbor sum to 1. Thus, we solve a geometric series $\sum_{n=0}^{N-1} ar^n = \frac{a(r^N-1)}{r-1} = 1$ for r, indexed from most to least important neighbor, where N is the total number of neighbors and a is a hyper-parameter. We set $N = 9$, since the computation doesn't change much for larger N. Given a value of a, we can compute the conditional probability of $L(v)$ given each combination of $L(N_3(v))$. This fully defines a Bayesian network. We can perform a line search on a, with $0.5 \leq a \leq 1$, to find the value which performs best on training data. Given observations of vertex labels, we can now perform Bayesian inference. Let $L(n_i)$ denote the label of the ith highest-PageRank neighbor of v. Then, for example, when $a = 0.5$ and we observe that $L(v) = 1$, we have that $\Pr(L(n_1) = 1) = 0.75$, $\Pr(L(n_2) = 1) = 0.62$, and $\Pr(L(n_3) = 1) = 0.55$. If we observe that $L(v) = -1$, we have that $\Pr(L(n_1) = 1) = 0.28$, $\Pr(L(n_2) = 1) = 0.40$, and $\Pr(L(n_3) = 1) = 0.45$.

Analysis of Problog Implementation

The Problog program contains a set of facts and a set of clauses, where the facts are labeled with corresponding probabilities and clauses define background knowledge. There are two different types of inference in Problog, exact inference and approximative inference. For the exact inference, one possible way to calculate the success probability of queries is to sum up probabilities of all independent proofs. But this method is not scalable for large programs since it is impossible to check a large number of subprograms. Instead, we can reconstruct all proofs by forming a DNF formula and then compute its probability. Similarly, Problog engine will first use SLD-resolution to obtain all independent proofs. Suppose f_i is the ground fact, e is the evidence or proof, c is the clause and S is the set of proofs of the query. Then we can get the set of subprograms containing proofs by calculating $\vee_{e \in S} \wedge_{c \in e} f_i$. Detailed derivation of the formula can be found in [14]. Thus we can calculate the success probability by computing probability of the following DNF formula.

$$P(q) = P(\vee_{e \in S} \wedge_{c \in e} f_i)$$

However, for the probability of a DNF formula, we cannot simply sum up different proofs as a fact can be used multiple times and repetition may occur. A natural idea to solve the problem is to disjoin repetitive facts in a different proof, which can be achieved by adding negated terms. But this method is also infeasible when the program gets more complex. In fact, this kind of disjoint sum problem is #P-complete [15]. Therefore, Problog uses Binary Decision Diagrams to represent the boolean function in graphs, which is feasible for even thousands of proofs.

When the formula gets more complicated, exact inference is too expensive to perform. Therefore, Problog provides some approximative inference

methods to cut down the computation.
Monte-Carlo is a classical method used for estimation. In Problog, a logic program is sampled first, and then the existence of proofs is checked. Finally, the probability of query is estimated by calculating the fraction of samples where the proof is valid. This process would be repeated until convergence.
Another approximative inference method is called K-Best. Since the original DNF contains a large number of possible proofs with the increase of program size, we could ignore part of proofs to reduce the complexity. This method estimates the probability by only considering the k most likely proofs. And the formula is slightly different from original one, where S_k is the set of proofs with probability greater than the k-th large one, so the order of all proofs by probability is needed.

$$P_k(q) = P(\vee_{e \in S_k} \wedge_{c \in e} f_i)$$

For most algorithms discussed above, the key challenge is how to store proofs, compute the probability of individual proof and sets of proofs. First, the Problog facts are transformed into different numbers or identifiers. Then during manipulation of proofs, all proofs can be represented as queues of identifiers in the order when the fact is used. This technique greatly reduces the space complexity for a single Problog proof. For set of proofs, in order to deal with prefixes and suffixes shared by different proofs, Problog stores sets of proofs by using tries. A specific distinction from original trie is when the number of sibling nodes exceeds a threshold, then the nodes are stored in a hash table to reduce the time complexity of search to $O(1)$. Since all proofs with the same prefixes are stored only once, this method can spare much space.

4 Conclusion

The methods examined in this paper provide a general framework for inference in graphical models. Numerical experiments confirmed the effectiveness of these methods when the underlying generative model is known to be Ising. While the Ising model is a reasonable model for data generation, inferring its parameters from training data remains a challenging problem. The maximum likelihood estimation of the Ising model parameters was slow and noisy, prone to overflow errors, and sensitive to initialization. These difficulties reflect both the difficulty of this estimation problem and the need, at minimum, for a more sophisticated optimization procedure. The simplest inference algorithms we examined are equivalent to combinatorial algorithms which minimize conflict among neighboring vertices. This observation suggests that heuristic combinatorial algorithms are a promising, and certainly cheaper, alternative to the probabilistic algorithms implemented in this paper.

Feedback
The need in Problog to specify the probability of every possible world limited its use for the inference problems we considered. The problems considered in this paper require a means of automatically specifying the probability of each combination of variables.

5 Contributions

Peter: PageRank, MCL, Bottleneck and Conductance, Gibbs Sampling, Inference using Ising model, MLE of Ising parameters, Background.
Tianyu: Bayesian Network, Inference and implementation in Problog.
Andrey: Inference with incomplete neighborhood. Combinatorial algorithm.
Jiapeng: Analysis of Problog implementation.

6 References

[1] A, Farasat, A. Nikolaev, S. Srihari, R. Blair. *Probabilistic Graphical Models in Modern Social Network Analysis.*
[2] S. Zhu and Wu, Y. Wu. *Computer Vision: Statistical Models for Marr's Paradigm.* 2020.
[3] Gelman et al. *Bayesian Data Analysis.* (pg. 290-291). Chapman & Hall, 2004.

[4] Y. Wu. *A Note on Monte Carlo Methods. UCLA Statistics*. STATS 102C, 2017.

[5] A. Barbu and S. Zhu. *Monte Carlo Methods*. Springer Nature Singapore Pte Ltd. 2020.

[6] A. Sinclair. *CS294 Markov Chain Monte Carlo: Foundations & Applications*. 2009. (*https://people.eecs.berkeley.edu/ sinclair/cs294/n2.pdf*).

[7] C. Robert. *The Metropolis–Hastings algorithm*. Universite Paris-Dauphine, University of Warwick, and CREST. 2016. (*https://arxiv.org/pdf/1504.01896.pdf*)

[8] *MCL - a cluster algorithm for graphs.* https://micans.org/mcl/

[9] *SNAP: Network datasets: Social circles*. Stanford University. https://snap.stanford.edu/data/egonets-Facebook.html.

[10] C. Geyer. *Markov Chain Monte Carlo Maximum Likelihood.* 1991. Computing Science and Statistics: Proc. 23rd Symp. Interface, 156–163.

[11] C. Geyer. *Reweighting Monte Carlo Mixtures.* 1991.

[12] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt. *Inference and learning in probabilistic logic programs using weighted Boolean formulas*.

[13] A. Kimmig, B. Demoen, L. De Raedt, V. Costa, and R. Rocha. *Theory and Practice of Logic Programming*. 2015.

[14] A. Kimmig, B. Demoen, L. De Raedt. *On the implementation of the probabilistic logic programming language ProbLog*. Theory and Practice of Logic Programming, 11(2-3), 235-262.

[15] Valiant, Leslie G.. *The Complexity of Enumeration and Reliability Problems.* SIAM J. Comput. 8 (1979): 410-421.

[16] De Raedt, Luc Kimmig, Angelika Toivonen, Hannu. (2007). *ProbLog: A Probabilistic Prolog and Its Application in Link Discovery*. IJCAI. 2462-2467.