Mach Learn (2015) 100:477–507
DOI 10.1007/s10994-015-5506-z

Poisson Dependency Networks: Gradient Boosted Models for Multivariate Count Data

Fabian Hadiji[1] · **Alejandro Molina**[1] · **Sriraam Natarajan**[2] · **Kristian Kersting**[1]

Received: 18 January 2015 / Accepted: 19 May 2015 / Published online: 11 July 2015

Abstract Although count data are increasingly ubiquitous, surprisingly little work has employed probabilistic graphical models for modeling count data. Indeed the univariate case has been well studied, however, in many situations counts influence each other and should not be considered independently. Standard graphical models such as multinomial or Gaussian ones are also often ill-suited, too, since they disregard either the infinite range over the natural numbers or the potentially asymmetric shape of the distribution of count variables. Existing classes of Poisson graphical models can only model negative conditional dependencies or neglect the prediction of counts or do not scale well. To ease the modeling of multivariate count data, we therefore introduce a novel family of Poisson graphical models, called Poisson Dependency Networks (PDNs). A PDN consists of a set of local conditional Poisson distributions, each representing the probability of a single count variable given the others, that naturally facilitates a simple Gibbs sampling inference. In contrast to existing Poisson graphical models, PDNs are non-parametric and trained using functional gradient ascent, i.e., boosting. The particularly simple form of the Poisson distribution allows us to develop the first multiplicative boosting approach: starting from an initial constant value, alternatively a log-linear Poisson model, or a Poisson regression tree, a PDN is represented as products of regression models grown in a stage-wise optimization. We demonstrate on several real world

Editors: João Gama, Indre Žliobaite, Alípio M. Jorge, and Concha Bielza.

✉ Kristian Kersting
kristian.kersting@cs.tu-dortmund.de

Fabian Hadiji
fabian.hadiji@cs.tu-dortmund.de

Alejandro Molina
alejandro.molina@cs.tu-dortmund.de

Sriraam Natarajan
natarasr@indiana.edu

1 LS VIII, TU Dortmund University, Dortmund, Germany

2 School of Informatics and Computing, Indiana University, Bloomington, IN, USA

datasets that PDNs can model positive and negative dependencies and scale well while often outperforming state-of-the-art, in particular when using multiplicative updates.

Keywords Graphical models · Dependency networks · Poisson distribution · Learning · MAP inference

1 Introduction

The world contains an unimaginably vast amount of information, and much of the data consists of counts, i.e., observations that can take only non-negative integer values. Without counting, we cannot know how many people are born and have died; how many men and women still live in poverty; how many children need education, and how many teachers to train or schools to build; the prevalence and incidence of diseases; how many customers arrive in a shop daily and whether demands for products are expanding, and how many teams did a team win in the football world championship. Our scientific and digital lives also thrive on counts. Example data are publication and citation counts, bag-of-X representations of, e.g., collections of images or text documents, genomic sequencing data, user-ratings data, spatial incidence data, climate studies, and site visits, among others. Behavioral data of users visiting web sites for example are tracked on a large scale, and visits or log-ins are counted and then used to enrich the user experience and to increase revenue. Or consider computational social science, a field that leverages the capacity to collect and analyze data at a scale that may reveal patterns of individual and group behaviors. Here, for instance, the number of people living in a region and the number of people migrating from one city to another one, among others, are of great interest to politicians and also influence decisions in education and research. Finally, counts also play an essential role in statistics. Consider for example an election analyst interested in the association of gender and voting intentions. Standing on a street corner for an hour recording data from anyone willing to talk to them, they build a contingency table, say, with the gender in the rows and and the voting intentions in the columns. In contrast to interviewing a fixed number of n individuals, there are no constraints on the row and column totals, and hence the cell entries follow a count distribution.

All these examples share a common attribute in that they require a distribution over counts, a potentially skewed, discrete distribution over the natural numbers. Indeed, there are several count distributions. Here we focus on one of the most widely used ones, namely the Poisson distribution. Poisson distributions are observed in sports data (Karlis and Ntzoufras 2003), and in natural sciences (Feller 1968). Clarke (1946) even observed that the points of impact of bombs in flying-bomb attacks are Poisson distributed. However, these classical studies have only considered the uni- and bivariate cases even though in many situations count variables may directly influence each other and should not be considered independently. For instance, if "Neural" appears often in a document then it is also likely that "Network" appears in the same document. Indeed, one may consider to employ a probabilistic graphical model widely used for modeling distributions among several random variables. Unfortunately, research has so far mainly focused on graphical models over binary, multinomial and Gaussian random variables only. These standard distributions, however, are often ill-suited for modeling count data[1], since they disregard either the infinite range over the natural numbers or the poten-

[1] If the mean λ of a Poisson distribution $p(x) = (\lambda^x e^{-\lambda})/(x!)$ is small, say less than 5, then its probability histogram is markedly asymmetrical, making a Normal-approximation ill-suited. Using Stirling's formula for $x!$, i.e., $x! \approx \sqrt{2\pi x}e^{-x}x^x$, as $x \to \infty$, one can see that its probability histogram is essentially symmetric and

tially asymmetric shape of the distribution of count variables; counts are neither binary nor continuous, they are discrete with a typically right skewed distribution over an infinite range.

Therefore, it is not surprising that extensions of them to the Poisson case have been proposed, see e.g., Besag (1974) and Yang et al. (2012, 2013). However, all existing extensions are restricted in that they model only negative conditional dependencies or neglect the prediction of counts or do not scale well. In particular negative dependencies limit the expressiveness of the models because one can only represent relationships where the mean of a variable decreases with increasing neighbors.

To ease the modeling of multivariate count data, we therefore propose a novel family of Poisson graphical models, called Poisson Dependency Networks (PDNs). A PDN consists of a set of local conditional Poisson distributions—each representing the probability of a single count variable given the others—that naturally facilities a Gibbs sampling inference procedure. Moreover, the family admits simple training procedures induced by a functional gradient view on training. Specifically, triggered by the recent successes of functional gradient ascent for representing multinomial dependency networks, our first technical contribution is to show that PDNs can be represented as sums of regression models grown in a stage-wise optimization starting from an initial constant value, or alternatively a log-linear model, or a Poisson regression tree. In fact, this first functional gradient approach to modeling multivariate Poisson distributions—as we will show empirically on several real world datasets—scales well and can model positive and negative dependencies among count variables while often outperforming state-of-the-art approaches. However, our main technical contribution is the development of the first multiplicative functional gradient ascent, which is demonstrated empirically to be able to boost the performance even further, since it essentially implements an automated step size selection without incurring in computational overhead.

We proceed as follows. We start off by discussing related work in more detail. We then introduce *Poisson Dependency Networks (PDNs)* in Sect. 3 and in Sect. 4 we show how to learn them, in particular, using the multiplicative functional gradient approach. In Sect. 5 we explain how to perform inference in PDNs and before concluding, Sect. 6 presents our exhaustive experimental evaluation on different synthetic and real-world datasets demonstrating the effectiveness of PDNs.

2 Related Work and a First Empirical Illustration

Most of the existing machine learning and data mining literature on graphical models—we refer to Koller and Friedman (2009) for a general introduction to graphical models—is dedicated to binary, multinominal, or certain classes of continuous (e.g., Gaussian) random variables. Undirected models, a.k.a. *Markov Random Fields (MRFs)*, such as Ising (binary random variables) and Potts (multinomial random variables) models have found a lot of applications in various fields such as robotics, computer vision and statistical physics, among others. Whereas MRFs allow for cycles in the graphical structure, directed models, a.k.a *Bayesian Networks (BNs)*, require acyclic directed relationships among the random variables. They have also been used in a number of applications such as planning, NLP and and information retrieval, among others. *Dependency Networks (DNs)*, the focus of the present paper, combine concepts from directed and undirected worlds and are due to Heckerman et al. (2000).

Footnote 1 continued
bell-shaped. More precisely, we rewrite $p(x) \approx (\sqrt{2\pi\lambda}x^{x+0.5})^{-1}e^{x-\lambda}\lambda^{x+0.5}$. In the limit of large λ, this approaches the Normal distribution $p(x) \approx (2\pi\lambda)^{-0.5}e^{-(x-\lambda)^2/2\lambda}$. This fact can be traced back to A. De Moivre, *Approximato ad Summam Terminorum Binmoo* $(a+b)^n$ *in Seriem Expansi* (London, 1733).

Specifically, like BNs, DNs have directed arcs but they allow for networks with cycles and bi-directional arcs, akin to MRFs. This makes DNs quite appealing because, if the data are fully observed, learning is done locally on the level of the conditional probability distributions for each variable mixing, directed and indirected as needed. Based on these local distributions, samples from the joint distribution are obtained via Gibbs sampling (Heckerman et al. 2000; Bengio et al. 2014). Except for few cases that we will discuss below, however, surprisingly little attention has been paid to (graphical) models of multivariate count data within the machine learning and data mining communities.

Indeed, when only one count variable is considered, the literature is vast. A lot of this can essentially be treated as a *Generalized Linear Model (GLM)*, see McCullagh and Nelder (1989) for example. In GLMs, the response variable is related to a link function applied to a linear model. One specific instantiation of GLMs is the Poisson regression case where the link function is the logarithm, i.e., the mean of the Poisson distribution is defined by a log-linear model. Compared to ordinary least squares regression, an advantage of the GLM framework is the fact that non-Gaussian error structures are possible. When considering jointly two or more count variables, however, things become more complicated and there exist much less work.

For instance, one can define a multivariate Poisson distribution by modeling node variables as sums of independent Poisson variables, see e.g., Karlis (2003) and Ghitany et al. (2012). Since this is again a Poisson, the marginals are Poisson as well. The resulting joint distribution, however, can only model positive correlations. There are also bivariate extension for specific models, e.g., Karlis and Ntzoufras (2003). In general, however, even calculating probabilities for these multivariate Poisson distributions is computationally challenging and hence their usage is often limited (Tsiamyrtzis and Karlis 2004). Hoff (2003) proposed to use *Generalized Linear Mixed Model (GLMMs)* using Poisson regression and modeling the dependencies between variables using random effects. Training the resulting GLMM, however, is computationally demanding because it requires the estimation of the unobserved mixed effects; nevertheless, GLMMs are often used, for example in studies in ecology and evolution (Bolker et al. 2009). Using just GLMs, Yang et al. (2012) have recently proposed an undirected Poisson model, called *GLM graphical model*, where—close in spirit to PDNs—each conditional node distribution is assumed to be from the exponential family with the Poisson distribution as one particular instantiation.

In contrast to the non-parametric functional gradient approach of PDNs, where interactions among variables are introduced only as needed and hence the learner does not explicitly consider the potentially immense parameter space, they employ a sparsity constrained, parametrized conditional MLE approach along the lines of Meinshausen and Bühlmann (2006). It must be mentioned that Yang et al.'s GLM graphical models—like PDNs—extend the seminal work by Besag (1974). More precisely, Besag's *Auto-Poisson model* can be seen as an instantiation of GLM graphical models as the latter ones allow for higher-order cliques. In general, this line of work models the dependencies between the variables directly, instead of adding mixed effects. That is, the mean of each variable follows a GLM where all neighboring variables are used as explanatory variables. However, to guarantee a consistent joint probability distribution known in closed form, the parameters are required to be negative, i.e., competitive relationships. In the case of arbitrary positive dependencies, the joint distribution is not guaranteed to be normalizable anymore because the normalization constant becomes infinite. In contrast, PDNs drop the guarantee of consistency and stay local, allowing for negative and positive parameters, i.e., competitive and attractive relationships. Alternatively, Kaiser and Cressie (1997) suggested the use of Winsorized Poisson distributions to remove the drawback of negative dependencies only. The Winsorized Poisson distribution is the univariate distribution obtained by truncating the inter-valued Poisson variables at a finite

Table 1 A comparison of existing Poisson graphical models most similar to PDNs

	Auto-Poisson (Besag 1974)	TPGMs (Yang et al. 2013)	SPGMs (Yang et al. 2013)	LPGMs (Allen and Liu 2013)	PDNs
Consistent joint distribution	✓	✓	✓	(✓)	(✓)
Arbitrary parameters	–	✓	✓	✓	✓
Unbounded range	✓	–	–	✓	✓
Covers learning	✓	✓	✓	✓	✓
Higher-order potentials	–	✓	✓	✓	✓
Tree-structured potentials	–	–	–	–	✓
Functional gradient	–	–	–	–	✓
Investigates inference	–	–	–	–	✓

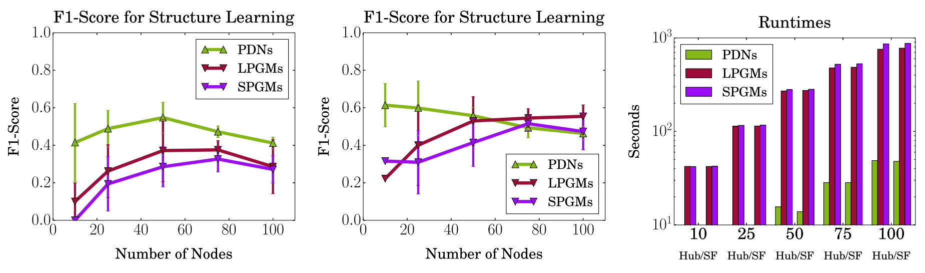
As one can see, PDNs cover all important aspects of modeling count data while facing almost no restrictions
The "(✓)" denotes that this is the case given that an unordered Gibbs sampler on the model converges, see main text for more details

constant. Doing so, however, makes estimation considerably harder than for PDNs that are naturally equipped with a simple learning approach.

Probably closest in spirit to PDNs are the recent approaches due to Allen and Liu (2013) and due to Yang et al. (2013). More precisely, Allen and Liu's *Local Poisson Graphical Models* (LPGMs) are also in the tradition of Besag's Auto-Poisson models and related to GLM graphical models. They assume that each variable conditioned on all other variables in the network follows a Poisson distribution. Similar to PDNs, they do not focus on a consistent joint distribution but consider local conditional probability models only. In contrast to PDNs, which employ a well scaling functional gradient ascent, the structure of LPGMs is learned with a computationally more demanding ℓ_1-norm penalized log-linear regression for neighborhood selection. Assuming the data is fully observed, the neighborhood for every node is learned separately in the spirit of Gaussian MRFs. Yang et al. (2013) introduced *Truncated Poisson Graphical Models* (TPGMs), *Quadratic PGM* (QPGMs), and *Sub-linear PGMs* (SPGMs). TPGMs are a modification of Kaiser and Cressie's truncated models. QPGMs are a generalization of the previously introduced Poisson graphical models with quadratic base measure that features both positive and negative parameters under the restriction that the pairwise parameter matrix is negative-definite. Instead of changing the base measure, SPGMs use sub-linear sufficient statistics to ensure normalizability. All this makes learning harder compared to PDNs since parametric ℓ_1-norm penalized log-linear regression or proximal gradient ascent are used to find a single best model. Instead, PDNs are triggered by the intuition that finding many rough rules of thumb of how count variables interact can be a lot easier than finding a single, highly accurate local Poisson model.

We have summarized[2] the properties of the existing Poisson graphical models that are most similar to our work and and how they compare to PDNs in Table 1. The "(✓)" denotes

[2] The table lists the TPGMs from Yang et al. (2013) instead of the Winsorized Poisson model from Kaiser and Cressie (1997) because TPGMs provide a consistent joint distribution and present the state-of-the-art approach for truncated Poisson graphical models.



(a) Scale-free networks. **(b)** Hub networks. **(c)** Runtimes in log-scale.

Fig. 1 Comparison of different count model approaches on synthetic data. Local models (LPGM and PDN) outperform global models (SPGM). Moreover, PDNs can be an order of magnitude faster. **a** Scale-free networks. **b** Hub networks. **c** Runtimes in log-scale (Color figure online)

that this feature does not generally hold but—as already mentioned—Bengio et al.'s (2014, Proposition 2) recent result suggests the existence of a consistent distribution. More precisely, Bengio et al. showed that an unordered Gibbs sampler over a DN equipped with evidence induces a so-called *Generative Stochastic Network (GSN)* Markov chain. As long as this GSN Markov chain has a stationary distribution, i.e., the Gibbs sampler converges, the DN defines a joint distribution, which, however, does not have to be known in closed form. Now, since PDNs (and hence LPGMs) are Dependency Networks (with local Poisson distributions), this argument should carry over, but a formal analysis is left for future work.

Our, empirical results support this as illustrated in Fig. 1. It summarizes the results of using the most recent approaches from Table 1 as well as PDNs for a network discovery task. More precisely, following Yang et al. (2013), two network families were considered that are commonly used throughout genomics: the hub and scale-free graph structures. As one can see, the local approaches, i.e., LPGMs and PDNs have competitive performances compared to the guaranteed consistent SPGMs[3]. The plots show the F1-scores for structure recovery for varying problem sizes averaged over five runs; we sampled 1000 graphs per run and problem size. Moreover, the training of PDNs is considerably faster compared to the state-of-the-art approaches, often an order of magnitude.

The speedup is likely due to the non-parametric nature of PDNs. In particular, we train PDNs using functional gradients. That is, we train them in a stage-wise manner at low and scalable costs following Friedman's *Gradient Tree Boosting* (GTB) (Friedman 2001). This boosting approach has been proven successful in a number of cases, see e.g., Ridgeway (2006), Kersting and Driessens (2008), Dietterich et al. (2008), Elith et al. (2008), Natarajan et al. (2012, 2014b, 2013) and Weiss et al. (2012), and since it estimates the parameters and the structure jointly, it is generally related to structure learning of graphical models, in particular to approaches that use the local neighborhood of each variable to construct the entire graph. For example, covariance selection is used in Gaussian models where edges are added to the graph until a stopping criterion is met. Despite being greedy, the method is not practical for multivariate distributions with a large number of variables. A popular alternative is neighborhood selection via Lasso (Meinshausen and Bühlmann 2006). This has also been used for learning the structure of binary Ising models, see e.g., Ravikumar et al. (2010). In the case of DNs, Heckerman et al. (2000) originally did neighborhood selection implicitly by learning probabilistic decision trees for each variable. Also, undirected probabilistic relational

[3] We did not compare to TPGMs since they were compared to SPGMs already in Yang et al. (2013).

models such as Markov Logic Networks were learned with the help of (boosted) decision trees in Khot et al. (2011), Lowd and Davis (2014) and Natarajan et al. (2014a).

3 Poisson Dependency Networks (PDNs)

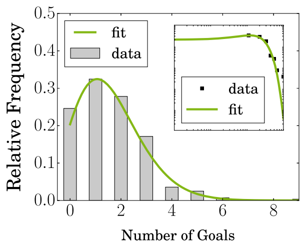
The review of related work revealed that modeling multivariate Poisson distributions faces the following main challenges:

Unrestricted parameters Some approaches only handle negative dependencies which is a serious limitation because positive dependences are often present as well.
Joint distribution Given local Poisson probability distributions, formulate a joint probability distribution.
Learning Recovery of the dependencies in the present data, so that an easy interpretation of the data is possible. A learned model should also quantify the dependences properly to enable accurate inference on unlabeled instances.
Inference Given a partially observed case, the goal of MAP inference is to predict the most likely assignment of the unobserved variables. Other probabilistic queries, such as marginal probabilities, are of great interest as well.

With this in mind, we now develop *Poisson Dependency Networks (PDNs)* that are different from existing Poisson graphical models in several ways:

Positive and negative dependencies PDNs are capable of modeling positive and negative dependencies. Furthermore, we do not have to restrict the weights to be symmetric or limited in any way.
Non-parametric PDN is the first non-parametric Poisson graphical model, both at the level of local models as well as at the global level of the overall model. The use of (Poisson) regression trees for the (initial) conditional distributions are more flexible than using parametric log-linear models for the mean in the Poisson distribution. Moreover, finding many rough rules of thumb of how count variables interact can potentially be a lot easier than finding a single, highly accurate local Poisson model.
Multiplicative GTB We present the very first multiplicative functional gradient ascent. This implements an automated step size selection and hence avoids an expensive line search while improving convergence.
Flexible structure learning The learning estimates the structure and parameters simultaneously and scales well. PDNs do not require fixing the missing symmetries in the structure to do inference. The structure learning of PDNs is competitive with consistent models, has much lower computational costs, and is readily parallelizable.
Predictions PDNs naturally facilitate a simple Gibbs sampling inference. More precisely, we use a Gibbs sampler that provides us with samples from a joint probability distribution. This joint distribution allows us to predict counts of unobserved instances easily instead of solely focusing on structure recovery as done for existing Poisson graphical models.

To introduce PDNs, we use X to denote a random variable and x as its instantiation. Sets of random variables are written as $\mathbf{X}$ and correspondingly their instantiations as $\mathbf{x}$. Given a set of random variables $\mathbf{X} = (X_1, \ldots, X_n)$ where each variable is defined over the natural numbers, including 0, then a Poisson Dependency Network (PDN) is a pair (G, P). Here, $G = (V, E)$ is a directed, possibly cyclic, graph with V being a set of nodes where each node corresponds to a random variable in $\mathbf{X}$. Hence, we can use nodes in G and the random variables in $\mathbf{X}$ interchangeably. $E \subseteq V \times V$ is a set of directed edges where each edge models



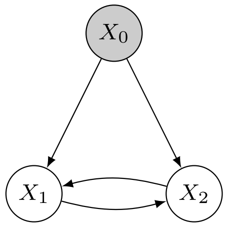


Fig. 2 Illustration of a Poisson Dependency Network (PDN). (*Left*) Plot of a fitted Poisson distribution. The number of goals scored in football games is Poisson distributed. The *plot* shows the distribution of home goals in the season 2012/13 of the German Bundesliga by the home team. Here, $\lambda = 1.59$, stating that the home team scored on average 1.59 goals per game. (*Right*) Example PDN. The conditional distribution of each count variable given its neighbors is a Poisson distribution. Observed variables are denoted by *gray shades*. Similar to a Bayesian network a PDN is directed, however, it also contains cycles (Color figure online)

a dependency between variables, i.e., if there is no edge between two variables X_i and X_j, the variables are conditionally independent given the other variables $\mathbf{X}_{\setminus i,j}$ in the network. Here, $\mathbf{X}_{\setminus i,j}$ is shorthand for $\mathbf{X} \setminus \{X_i, X_j\}$. We refer to the nodes that have an edge pointing to X_i as the parents of X_i, denoted as $\mathbf{pa}_i$. P is a set of conditional probability distributions for every variable in $\mathbf{X}$. For now, we assume that each variable X_i given its parents $\mathbf{pa}_i$ is Poisson distributed, i.e.,

$$p(x_i|\mathbf{pa}_i) = p(x_i|\mathbf{x}_{\setminus i}) = \frac{\lambda_i(\mathbf{x}_{\setminus i})^{x_i}}{x_i!} e^{-\lambda_i(\mathbf{x}_{\setminus i})}. \tag{1}$$

Here, $\lambda_i(\mathbf{x}_{\setminus i})$ highlights the fact that the mean can have a functional form that is dependent on X_i's neighbors. Often, we will refer to it simply as λ_i. An example of such a local Poisson conditional probability distribution is illustrated in Fig. 2 (left). The construction of the local conditional probability distribution is similar to the (multinomial) Bayesian network case, however, in the case of PDNs, the graph is not necessarily acyclic and $p(x_i|\mathbf{x}_{\setminus i})$ has an infinite range, and hence cannot be represented using a finite table of probability values. Finally, the full joint distribution is simply defined as the product of local distributions:

$$p(\mathbf{x}) := \prod_{x_i \in \mathbf{x}} p(x_i|\mathbf{x}_{\setminus i}) = \prod_{x_i \in \mathbf{x}} \frac{\lambda_i^{x_i}}{x_i!} e^{-\lambda_i}. \tag{2}$$

Note, however, that doing so does not guarantee the existence of a consistent joint distribution, i.e., a joint distribution of which they are the conditionals. We will come back to this issue later.

An example of a PDN with three variables is depicted in Fig. 2 (right). The gray shaded variable X_0 can be seen as an observational vector. In the spirit of other graphical models, for example Conditional Random Fields (Lafferty et al. 2001), we can define a set of observed variables in advance. It will not be necessary to learn a local model for these variables as they are always observed. Additionally, we have fewer constraints on these variables. They do not have to be count variables but instead can be arbitrary features. This allows us to easily incorporate features that are for example normal distributed, hence, paving the way for *hybrid models* with local models of all kind of local distributions. We call this new formalism *Poisson Dependency Networks* because they generalize *Dependency Networks (DNs)* (Heckerman et al. 2000) for multinomial distributions to the Poisson case.

One crucial part of PDNs that we have not touched upon yet is the encoding of $p(X_i|\mathbf{X}_{\setminus i})$ and of λ_i in particular. There are two sensible ways. First, one can follow a parametric approach as it has been done before in the literature. Second, as an alternative, we propose a non-parametric encoding.

More precisely, we show that a PDN can be represented as sums resp. products of regression models grown in a stage-wise optimization starting from an initial constant value, or alternatively a log-linear Poisson model, or a Poisson regression tree. Before doing so, let us touch upon the issue of consistency again. Our non-parametric approach to model the means can essentially be seen as linear functions with no restrictions on the parameters. In turn, we will generally not be able to formulate the underlying joint probability distribution of the PDN in closed-form by means of a standard (pairwise) Poisson graphical model. However, as we will show later, we can use sampling techniques to draw samples from the joint distribution as we have access to the conditional probabilities. If this sampling converges, there exists a consistent distribution, which however does not have to be known in closed form. We will discuss the question of consistency in greater detail in this work.

4 Learning Poisson Dependency Networks

Learning[4] PDNs amounts to determining the conditional probability distributions from a given set of m trainings instances over n count variables X_i

$$\left[\mathbf{x}^{(\mathbf{j})} \in \mathbb{N}^{\mathbf{n}}\right]_{\mathbf{j}=\mathbf{1},\mathbf{2},\ldots,\mathbf{m}},$$

which in turn is equal to learning λ_i for each variable X_i because the Poisson distribution is completely determined by the mean. However, λ_i will possibly depend on all other variables in the network, and these dependences define the structure of the network. We will now develop a functional gradient ascent approach to learning PDNs, i.e., learning the λ_is. We will do so by starting from parametrized maximum likelihood estimation, extending it to additive local Poisson models grown in a stage-wise manner, and finally turning this into a multiplicative update.

Before going into details, let us summarize the resulting high-level approach in Algorithm 1, since it covers all the approaches. The mean λ_i of each variable is assumed to consist of a set of local models grown in a stage-wise manner. More precisely, initially the set is a singleton learned in Line 9. Here, the formula $X_i \sim \sum \mathbf{X}_{\setminus i}$ denotes that the mean for X_i can have all other variables potentially as features. Please note that the models used may include hyper-parameters such as step sizes and pruning parameters (if e.g., using regression trees)—we will touch upon them along with each approach—that might be estimated using a grid search and a validation set. After the initial learning, a pre-defined number of gradient steps (T) are made to further improve the model (line 11). Thus, the main computational task is the induction of regression models. More precisely, one regression model per random variable X_i and per iteration in the stage-wise optimization (lines 9–11). For simplicity, let us assume that the regression models are trees. Tree-based regression models are known for their simplicity and efficiency when dealing with domains with large numbers of variables and cases. With careful implementations, inducing a single regression tree can be realized with practically linear time complexity, see e.g., Dobra and Gehrke (2002). Hence, the running time is, practically speaking, linearly dependent on $n \cdot T$, i.e., on the number of count variables times the number of stage-wise optimization iterations.

[4] For the sake of simplicity we assume the data is fully observed. Together with the Gibbs sampler discussed later, everything can in principle be extended to the partially observed case.

Algorithm 1 Gradient Boosting Poisson Dependency Networks (PDNs).

```
1: function LearnPdn(data)
2:    score ← − inf
3:    optimalPDN ← none
4:    for each $\alpha_i$ in HyperparameterSpace do
5:       parameterScore ← 0
6:       for each train, test in CrossValidationSplit(data) do
7:          P ← []
8:          for each $X_i$ in X do
9:             P[$i$] ← LearnInitialCondProb($\alpha_i$, $X_i \sim \sum \mathbf{X}_{\setminus i}$)
10:            for each $t$ in $[1, T]$ do
11:               GradientBoosting(P[$i$])
12:         parameterScore ←parameterScore + ScoreFunction(test, P)
13:      if parameterScore > score then
14:         optimalPDN ← P
15:         score ← parameterScore
     return OptimalPDN
```

Depending on how we encode the initial models and how we grow the local models per variable—additive or multiplicative updates—different learning approaches are realized, which we will now discuss.

4.1 Poisson Log Linear Models

To grasp the idea of functional gradient ascent for training PDNs, it is helpful to first describe the learning a single parametric Poisson model. Most commonly, as mentioned in the related work, the mean is modeled using a log-linear model. In its most naive form, without further regularization, this presents a fully-connected PDN. More precisely, using a log-linear model, the mean of each variable is:

$$E[X_i] = \lambda_i = \exp\left(\beta_i + \sum_{j \neq i} \beta_{ij} \cdot x_j\right) \tag{3}$$

In the univariate case, this approach is often referred to as *Poisson Regression*. The parameters β_i and β_{ij} determine the mean—$\beta_{ij} > 0$ models a positive dependency on X_j whereas $\beta_{ij} < 0$ models a negative dependency—and can be learned via maximum likelihood estimation (MLE). That is, we assume that the training examples are i.i.d. and we seek for parameters that maximize the conditional log-likelihood

$$\mathrm{cll}(\beta_i, \boldsymbol{\beta}_{ij}) = \log \prod_{k=1}^{m} p\left(x_i^{(k)} | \mathbf{x}_{\setminus i}^{(k)}\right) = \sum_{k=1}^{m} \log p\left(x_i^{(k)} | \mathbf{x}_{\setminus i}^{(k)}\right).$$

So, we obtain the partial derivatives

$$\frac{\partial\, \mathrm{cll}}{\partial \beta_i} = \frac{\partial}{\partial \beta_i} \sum_{k=1}^{m} x_i^{(k)} \log\left(\lambda_i^{(k)}\right) - \log\left(x_i^{(k)}!\right) - \lambda_i^{(k)} = \sum_{k=1}^{m} x_i^{(k)} - \lambda_i^{(k)},$$
$$\frac{\partial\, \mathrm{cll}}{\partial \beta_{ij}} = \frac{\partial}{\partial \beta_{ij}} \sum_{k=1}^{m} x_i^{(k)} \log\left(\lambda_i^{(k)}\right) - \log\left(x_i^{(k)}!\right) - \lambda_i^{(k)} = \sum_{k=1}^{m} \left(x_i^{(k)} - \lambda_i^{(k)}\right) \cdot x_j^{(k)}.$$

Since there is no closed form solution for solving this problem, typically, we employ an iterative approach to improve the conditional log-likelihood. For instance, using a simple gradient ascent, we can take a step in the direction of the gradient in each iteration until convergence:

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^{t} + \eta \cdot \nabla \, \mathrm{cll}$$

where ∇ cll denotes the gradient consisting of all partial derivatives. The speed of convergence is influenced by the choice of the step size η which denotes one of the possible hyper-parameters in Algorithm 1. Besides the simple gradient ascent, algorithms such as (L-)BFGS are common for solving such optimization problems. However, in the case of Poisson Regression, iteratively-re-weighted least squares (IRLS) is probably the most popular technique. We refer to McCullagh and Nelder (1989) for further details on IRLS.

Before moving to functional gradient ascent, let us make a further note on the consistency of PDNs. In general, we may not be consistent, however, using log-linear models, we can fit our PDNs with log-linear means into the setting of Besag (1974) (and its follow up work) by restricting the parameters and structure. As Besag showed for the pairwise lattice case, there exists an undirected graphical model with Poisson local conditional probability distributions if the parameters $\boldsymbol{\beta}$ are non-positive. Further it is assumed that $\beta_{ij} = \beta_{ji}$. This was further generalized for the non-pairwise case in Yang et al. (2012). If our parameters satisfy these conditions, we can specify the joint distribution for our model and hence we do have a consistent Poisson MRF as well.

However, we are interested in models with as few limitations as possible on the structure and the parameters. Hence we do not rely on the existence of a closed form equation for the joint distribution or even insist on the existence of a consistent distribution. Instead, we introduce a more general class of Poisson graphical models. This is mainly motivated by computational concerns. For domains with a large number of count variables and many dependencies among them, learning a consistent Poisson MRF might be too costly. Moreover, not being able to model both positive and negative influences may hurt the performance and limits the expressiveness. Finally, as the results of our first experimental comparison in Fig. 1 already showed, approaches that are guaranteed to provide an explicit joint distribution are not necessarily better. Specifically, we follow Heckerman et al. (2000) and Bengio et al. (2014) and employ the machinery of Markov chains to generate pseudo samples. The generated samples are used to answer complex probabilistic count queries. For more details on Gibbs sampling and a justification for its usage we refer to Sect. 5.

4.2 Non-parametric Poisson Models Via Gradient Boosting

Local log-linear models tend to estimate fully connected networks and overfit. Consequently, they typically do not provide major insights into the structure of the underlying data generation process. Hence, regularization or post-processing such as thresholding (Allen and Liu 2013; Yang et al. 2013) have to be employed to extract the true nature of the network. Moreover, as demonstrated in our experiments, using log-linear models easily results in PDNs that overshoot at prediction time leading to overflows and, hence, prediction is not possible at all.

Therefore, we propose an alternative: instead of a single local log-linear model for λ_i, we grow a set of local models in stage-wise manner using gradient information (Line 11 of Algorithm 1). This way we avoid considering the large parameter space explicitly and include interactions among count variables only as needed. We here employ regression trees for this boosting approach and, hence, call the resulting approach *Gradient Tree Boosting* (GTB). Using trees has two significant advantages over standard parametric and non-parametric regression approaches:

- By allowing the tree-structure to handle much of the overall model complexity, the models in each leaf can be kept at a low order and, hence, are more easily interpreted.
- Interactions among features are directly conveyed by the structure of the regression tree. As a result, interactions can be understood and interpreted more easily in qualitative terms; we will touch upon this later again.

To develop gradient tree boosting, recall that the parameters of a parametrized log-linear model can be viewed as a sum of gradients. Gradient boosting lifts this intuition to the function space. That is, using the log link function, the mean $\lambda_i(\mathbf{x}_{\setminus i})$ of a Poisson variable X_i is viewed as $\lambda_i(\mathbf{x}_{\setminus i}) = \exp\left(\psi_i(\mathbf{x}_{\setminus i})\right)$, i.e., as a function of some (feature-)function ψ_i. Now, we perform a gradient ascent in (log) function space for some iterations T in order to estimate the feature functions $\psi_i(\mathbf{x}_{\setminus i})$ for the count variables x_i. This corresponds to representing the local functions of a PDN as a weighted sum:

$$\lambda_i^t(\mathbf{x}_{\setminus \mathbf{i}}) = \exp\left(\psi_i^t(\mathbf{x}_{\setminus i})\right) = \exp\left(\psi_i^{t-1}(\mathbf{x}_{\setminus i}) + \eta \cdot \nabla_i^t\right) \tag{4}$$

The initial $\psi_i^0(\mathbf{x}_{\setminus i})$ can be, e.g., a constant, the logarithm of the empirical mean of X_i, or a more complex model. The ∇_i^ts are functional gradients:

$$\nabla_i^t = E_{\mathbf{x}_{\setminus i}, x_i}\left[\frac{\partial}{\partial \psi_i^{t-1}(\mathbf{x}_{\setminus i})} \log p\left(x_i | \mathbf{x}_{\setminus i}; \psi_i^{t-1}(\mathbf{x}_{\setminus i})\right)\right] \tag{5}$$

indicating intuitively how we would like the mean model to change in order to increase the log-likelihood. The expectation $E[\cdot]$ in (5), however, requires access to the joint distribution, which we do not have.

Fortunately, we can approximate it with the help of the instances in our training dataset, which are sampled from the true joint distribution anyhow:

$$\nabla_i^t \approx \frac{1}{m}\sum_j \frac{\partial}{\partial \psi_i^{t-1}(\mathbf{x}_{\setminus i}^{(j)})} \log p\left(x_i^{(j)} | \mathbf{x}_{\setminus i}^{(j)}; \psi_i^{t-1}(\mathbf{x}_{\setminus i}^{(j)})\right) = \frac{1}{m}\sum_j \nabla_i^t(x_i^{(j)}, \mathbf{x}_{\setminus i}^{(j)})$$

Intuitively, as long as we can estimate $\nabla_i^t(x_i^{(j)}, \mathbf{x}_{\setminus i}^{(j)})$, they give us point-wise regression examples

$$\left(\mathbf{x}_{\setminus i}^{(j)}, \nabla_i^t(x_i^{(j)}, \mathbf{x}_{\setminus i}^{(j)})\right)$$

of the functional gradient. Hence, it is a sensible idea to train a regression model using them approximating the true functional gradient; following Dietterich et al. (2008), we minimize the following objective

$$\sum_j \left[h_i^t(\mathbf{x}_{\setminus i}^{(j)}) - \nabla_i^t(x_i^{(j)}, \mathbf{x}_{\setminus i}^{(j)})\right]^2 \tag{6}$$

using a regression tree h_i^t (Breiman et al. 1984). Here, j iterates over all instances in our training dataset, and $\mathbf{x}^{(j)}$ is the jth training example. Although fitting a regression model to (6) is not exactly the same as the desired ∇_i^t, it will point into the same direction if we have enough training examples. So, ascent in the direction of the regression model h_i^t—replacing ∇_i^t by h_i^t in (4)—will approximate the true functional gradient ascent (4) and was empirically proven to be quite successful for training many probabilistic models. How do the point-wise gradient regression examples look like for PDNs? For the log link function $\lambda_i = \exp(\psi_i(\mathbf{x}_{\setminus i}))$ with some feature function ψ_i, where we have now omitted the iteration index t for the sake of simplicity, we get:

Theorem 1 (see also e.g., Ridgeway (2006)) *The point-wise gradients for the log link function* $\exp(\psi_i(\mathbf{x}_{\setminus i}))$ *can be generated using*

$$\frac{\partial \log p(x_i|\mathbf{x}_{\setminus i};\, \psi_i(\mathbf{x}_{\setminus i}))}{\partial \psi_i(\mathbf{x}_{\setminus i})} = x_i - \lambda_i(\mathbf{x}_{\setminus i}).$$

Proof One can verify the correctness of this gradient as follows:

$$\begin{aligned}\frac{\partial \log p(x_i|\mathbf{x}_{\setminus i};\, \psi_i(\mathbf{x}_{\setminus i}))}{\partial \psi_i(\mathbf{x}_{\setminus i})} &= \frac{\partial}{\partial \psi_i(\mathbf{x}_{\setminus i})} x \cdot \psi_i(\mathbf{x}_{\setminus i}) - \log(x_i!) - \exp(\psi_i(\mathbf{x}_{\setminus i}))\\ &= x_i - \exp(\psi_i(\mathbf{x}_{\setminus i})) = x_i - \lambda_i(\mathbf{x}_{\setminus i})\end{aligned}$$

□

These point-wise gradients are quite intuitive. We want to make the predicted means $\lambda_i(\mathbf{x}_{\setminus i})$ as similar to the observed count x_i as possible.

However, we are still left with the step size parameter η to compute the updated mean model in (4). For other probabilistic models such as CRFs, performing a line search was reported to be too expensive (Dietterich et al. 2008). Hence, it was suggested to rely on the "self-correcting" property of tree boosting to correct any overshoot or undershoot on the next iteration, i.e., to use a step size of $\eta = 1$. Unfortunately, we have observed in our experiments that using a fixed step size of $\eta = 1$ can lead to very slow convergence or failures due to overflows for large counts. Consequently, we now develop our main technical contribution, *a multiplicative gradient boosting approach* that avoids this without incurring any computational overhead.

4.3 Multiplicative Gradient Boosting

For non-negative optimization problems, multiplicative update rules for the parameters have been shown to have much better convergence rates, see e.g., Lee and Seung (2000), Saul and Lee (2001), Sha et al. (2003) and Yang and Laaksonen (2007), and we confirm this for PDNs in our experimental section. More importantly, since the tree-structure of the induced regression trees handle much of the overall PDN complexity, faster convergence implies sparser PDNs.

To derive a multiplicative update, we consider a simpler functional dependency between $\lambda_i(\mathbf{x}_{\setminus \mathbf{i}})$ and the feature function ψ_i. More precisely, triggered by Chen et al. (2009), who have proven the use of the identity link function to be beneficial when using parametrized univariate Poisson distribution for large-scale behavioral targeting, we assume $\lambda_i = \psi_i$ (again omitting the iteration index t). For this link function, the point-wise gradients are the following:

Theorem 2 *The point-wise gradients of the log-probability assuming the identity link function are*

$$\frac{\partial \log p(x_i|\mathbf{x}_{\setminus i};\, \psi_i(\mathbf{x}_{\setminus i}))}{\partial \psi_i(\mathbf{x}_{\setminus i})} = \frac{x_i}{\lambda_i(\mathbf{x}_{\setminus i})} - 1,$$

and can be used to realize an additive functional gradient ascent.

Proof The correctness of the point-wise gradient can be seen as follows:

$$\begin{aligned}\frac{\partial \log p(x_i|\mathbf{x}_{\setminus i};\, \psi_i(\mathbf{x}_{\setminus i}))}{\partial \psi_i(\mathbf{x}_{\setminus i})} &= \frac{\partial}{\partial \psi_i(\mathbf{x}_{\setminus i})} x_i \log \psi_i(\mathbf{x}_{\setminus i}) - \log(x_i!) - \psi_i(\mathbf{x}_{\setminus i})\\ &= \frac{x_i}{\psi_i(\mathbf{x}_{\setminus i})} - 1 = \frac{x_i}{\lambda_i(\mathbf{x}_{\setminus i})} - 1\end{aligned}$$

□

The multiplicative update now follows from Theorem 2 together with a functional step size.

Corollary 1 *The multiplicative functional gradient ascent using the identity link function is*

$$\psi^{t+1}(\mathbf{x}_{\backslash i}) = \psi^{t}(\mathbf{x}_{\backslash i}) \cdot E_{\mathbf{x}_{\backslash \mathbf{i}}, x_i}\left[\frac{x_i}{\lambda_i(\mathbf{x}_{\backslash i})}\right], \tag{7}$$

and the training instances can be generated using

$$\left(\mathbf{x}_{\backslash i}^{(j)}, \frac{x_i}{\lambda_i(\mathbf{x}_{\backslash i})}\right). \tag{8}$$

Proof From Theorem 2 it follows that the point-wise gradients can be split into a negative part, namely 1, and a positive part, namely $x_i \lambda_i^{-1}(\mathbf{x}_{\backslash i})$. Since the functional gradient ∇_i^t in (5) is an expectation, ∇_i^t itself can be split into a negative part and a positive part, $\nabla_i^{t-} = E_{\mathbf{x}_{\backslash \mathbf{i}}, x_i}[1] = 1$ respectively $\nabla_i^{t+} = E_{\mathbf{x}_{\backslash \mathbf{i}}, x_i}\left[\frac{x_i}{\psi_i(\mathbf{x}_{\backslash i})}\right]$, with $\nabla_i^t = \nabla_i^{t+} - \nabla_i^{t-}$. Setting now the step size to

$$\eta = \frac{\psi_i^t(\mathbf{x}_{\backslash i})}{\nabla_i^{t-}},$$

the additive gradient boosting (4) can be rewritten as

$$\begin{aligned}
\psi^{t+1}(\mathbf{x}_{\backslash i}) &= \psi_i^t(\mathbf{x}_{\backslash i}) + \eta \cdot \nabla_i^t = \psi^t(\mathbf{x}_{\backslash i}) + \eta \cdot (\nabla_i^{t+} - \nabla_i^{t-}) \\
&= \psi_i^t(\mathbf{x}_{\backslash i}) + \frac{\psi_i^t(\mathbf{x}_{\backslash i})}{\nabla_i^{t-}} \cdot (\nabla_i^{t+} - \nabla_i^{t-}) = \psi^t(\mathbf{x}_{\backslash i}) \cdot \frac{\nabla_i^{t+}}{\nabla_i^{t-}} = \psi^t(\mathbf{x}_{\backslash i}) \cdot \frac{\nabla_i^{t+}}{1} \\
&= \psi^t(\mathbf{x}_{\backslash i}) \cdot E_{\mathbf{x}_{\backslash i}, x_i}\left[\frac{x_i}{\psi_i(\mathbf{x}_{\backslash i})}\right] = \psi^t(\mathbf{x}_{\backslash i}) \cdot E_{\mathbf{x}_{\backslash i}, x_i}\left[\frac{x_i}{\lambda_i(\mathbf{x}_{\backslash i})}\right]
\end{aligned}$$

This proves the correctness of the multiplicative functional gradient (7). Moreover, following now the same steps as for the additive functional gradient ascent proves the correctness of point-wise gradients (8). That is, as for the additive update, following the multiplicative functional gradient (7), approximated using regression trees trained using (8), will increase the log-likelihood assuming there are enough training examples. □

The point-wise multiplicative updates are quite intuitive. Instead of making the differences as small as possible, we want to make the *ratio* of observed counts x_i and predicted means $\lambda_i(\mathbf{x}_{\backslash i})$ as close to 1 as possible.

Finally, we comment on the choice of the identity link function. Using it, we make the assumption that the expected means are greater than zero. This is sensible, since a count variable should be observable. Second, positive and negative dependencies can still be expressed in single regression models induced over the iterations. Third, one can naturally realize Laplace smoothing by adding constants α_i and β_i to the point-wise gradients:

$$\frac{x_i + \alpha_i}{\lambda_i(\mathbf{x}_{\backslash i}) + \beta_i}. \tag{9}$$

That is, α_i / β_i gives the default Poisson mean of new configurations not encountered during training. Finally, the identity and log link functions are weakly connected. Following the same steps as in the proof of Corollary 1 but now using Theorem 1, we arrive at the following multiplicative functional gradient ascent for the log link function

$$\psi^{t+1}(\mathbf{x}_{\backslash i}) = \psi^t(\mathbf{x}_{\backslash i}) \cdot \frac{E_{\mathbf{x}_{\backslash i}, x_i}[x_i]}{E_{\mathbf{x}_{\backslash i}, x_i}\left[\lambda_i(\mathbf{x}_{\backslash i})\right]} = \psi^t(\mathbf{x}_{\backslash i}) \cdot E_{\mathbf{x}_{\backslash i}, x_i}\left[\frac{\bar{x}_i}{\lambda_i(\mathbf{x}_{\backslash i})}\right], \tag{10}$$

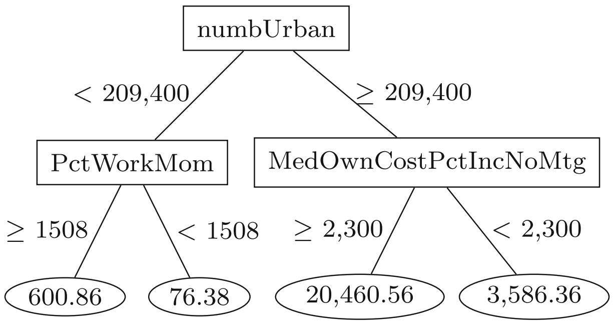


Fig. 3 Example of a Poisson regression tree for the Communities and Crime data set from our experimental evaluation

since $E_{\mathbf{x}_{\setminus i},x_i}[x_i]$ is the empirical mean $\bar{x}_i$ of x_i. This multiplicative update, however, is not valid for all cases. For example, in cases where the empirical mean of a variable is smaller one, we cannot initialize with this mean because we have $\psi_i^0 < 0$ due to $\lambda_i^0 = \exp(\psi_i^0)$. In such a setting, the direction of the gradient step is not correct anymore. Nevertheless, since $\bar{x}_i$ is a constant for all iterations, we can move the expectation out of the ratio. Approximating the expectation using the training samples now yields $E_{\mathbf{x}_{\setminus \mathbf{i}},x_i}[\bar{x}_i \cdot \lambda_i^{-1}(\mathbf{x}_{\setminus i})]$

$$\approx \frac{1}{m}\sum_j \left[\frac{\bar{x}}{\lambda_i(\mathbf{x}_{\setminus i}^{(j)})}\right] = \frac{1}{m^2}\sum_j \left[\frac{\sum_l x_i^{(l)}}{\lambda_i(\mathbf{x}_{\setminus i}^{(j)})}\right] \geq \frac{1}{m^2}\sum_j \left[\frac{x_i^{(j)}}{\lambda_i(\mathbf{x}_{\setminus i}^{(j)})}\right] \tag{11}$$

because $\bar{x}_i$ is the empirical mean of non-negative values and, hence, it is always larger than any particular x_i. Since the number of training examples m is a constant, this proves that the multiplicative point-wise gradients for the identity link function establish a lower bound for a multiplicative update for the log link function.

4.4 Model Compression, Initial Count Models, and Dependency Recovery

So far, we have specifically assumed two natural candidates for the initial mean model $\lambda_i^0(\mathbf{x}_{\setminus i})$, namely a constant or the empirical mean of X_i. Both models might not be very precise and, in turn, may require many iterations of optimization, even when using adaptive step sizes via multiplicative updates, resulting in a large number of regression trees and a tendency to overfit.

One additional way to avoid overfitting next to Laplace estimates is model compression (Bucila et al. 2006). Here, we collapse the trained additive resp. multiplicative model into a single model. To do so, we evaluate it on the training set and learn a single Poisson model per count variable. Since we are dealing with count data, we should use e.g., a Poisson regression tree (Chaudhuri et al. 1995) as shown in Fig. 3; that is, the compressed PDN model consists of a set of local Poisson regression trees, one for each variable X_i, where we train tree$_i(X_i|\mathbf{X}_{\setminus i})$ and evaluate $\lambda_i^0(\mathbf{x}_{\setminus i}) = \text{tree}_i(\mathbf{x}_{\setminus i})$.

More precisely, a Poisson regression tree partitions the training examples in the space of the dependent variables $\mathbf{X}_{\setminus i}$ in order to best fit the response variable X_i. It is a binary tree whose leaves represent the λ_i of that given partition, all other nodes represent the splitting criterion on a variable $X_j \in \mathbf{X}_{\setminus i}$. We use the Poisson regression tree implementation of rpart (Therneau et al. 2011) where the splitting criterion is given by the likelihood ratio test for two Poisson groups $D_{\text{parent}} - (D_{\text{leftson}} + D_{\text{rightson}})$ with the deviance given by

$D = \sum \left[x_i \log \left(\frac{x_i}{\hat{\lambda}} \right) - (x_i - \hat{\lambda}) \right]$, where $\hat{\lambda}$ is the sample mean[5] of count variable X_i. This splitting is recursively applied until each subgroup reaches a minimum size or there are no improvements. To avoid overfitting, the depth of a tree is typically limited a priori, alternatively post-pruning can be used after the tree has been learned. In the rpart-implementation the pre-pruning is controlled by a complexity parameter `cp`. This parameter (which we consider to be part of the hyper-parameter space of our algorithm) controls the size of the initial tree. A secondary step is executed where the tree is pruned back using cross validation after the initial tree has been learned. The height of the final tree will be the one that reduces the cross validation error and in our experiments it generalizes well.

Compression is not the only advantage of Poisson regression trees. Obviously, we can also use them as initial models, providing us with a potential head start. Moreover, interactions among count variables are directly conveyed by the structure of the compressed PDN and, as a result, interactions can be understood and interpreted more easily in qualitative terms. More precisely, since after compression there is only one local tree model per count variable, we simply look at the count variables used in the inner split nodes of the trees; they indicate relevant features of the PDN. It is important to note that one tree can use a variable X_k multiple times with different splitting criteria. This indicates that the variable is important in the PDN. Also, count variables closer to the root of a tree are more important than a variable further down the tree. This is e.g., captured by Breiman et al.'s (1984) notation of *relative influence* saying how important the value of x_v is for predicting the value for x_u:

$$I^2(u|v; \lambda_u) = \sum_{l=1}^{L-1} i_l^2 \cdot \delta\Big(v(l) = u\Big),$$

where l iterates over the levels of the Poisson tree λ_u of u, the value i_l^2 is the maximal estimated improvement over a constant fit over the entire region of the current node $v(l)$, and δ is an indicator function selecting all splits involving x_v.

To summarize, in contrast to other Poisson graphical models, compressed PDNs return local models that are likely to be sparse and therefore easier to interpret.

5 Making Predictions Using Poisson Dependency Networks

In many applications, we obtain only partially observed instances and we want to use probabilistic inference to predict the values of the missing variables. To be more precise, assume that $\mathbf{X} = \mathbf{Y} \cup \mathbf{E}$, where $\mathbf{Y}$ and $\mathbf{E}$ are disjoint. $\mathbf{Y}$ amounts to the unobserved variables and $\mathbf{E}$ describes the evidence. Then we want to answer queries of the form $p(\mathbf{y}|\mathbf{e})$, $p(y_i|\mathbf{e})$, or $\arg\max_{\mathbf{y}} p(\mathbf{Y} = \mathbf{y}|\mathbf{e})$. The latter query corresponds to MAP inference and finds the most likely assignment to the unobserved variables. In an univariate Poisson model, MAP inference consists of just reading off the mode of the Poisson distribution which is equal to $\lfloor \lambda_i \rfloor$. Other marginal probabilities, e.g., $p(X_i = k)$, can also be read off the distribution because the Poisson distribution is completely defined based on the mean. The same holds for a variable X_i in a PDN if all neighbors of this variable are observed. However, PDNs with unobserved variables require an inference machinery to account for the dependencies. We here resort to Gibbs sampling (Geman and Geman 1984) to do so.

Since we do not know explicitly the underlying joint distribution, the Gibbs sampler provides us only with pseudo samples, and hence it is called *Pseudo Gibbs* sampler (Heckerman

5 For stability reasons the implementation uses a revised Bayes estimate of $\hat{\lambda}$.

et al. 2000) due to the potentially inconsistencies arising from conflicting local and joint distributions. Specifically, the Pseudo Gibbs sampler starts with an arbitrary initialization of the unobserved variables and then iterates over each variable for a previously defined number of sweeps. Each sweep produces a new sample by first calculating the conditional probability distribution for every variable X_i. That is calculating the mean λ_i based on the current states of its parent variables $\mathbf{pa}_i$. Based on the Poisson distribution parameterized by λ_i, a new state for X_i is sampled. This procedure will sample from the joint distribution after an adequate burn-in phase which removes early samples because they are heavily biased by the initial values. In terms of computations, this algorithm does not distinguish itself from a standard Gibbs sampler. However, in the case of an inconsistent set of local probability distributions, it is referred to as a "Pseudo" Gibbs sampler to stress the fact that it may not produce samples from a joint distribution consistent with the PDN. Its run time is identical to the one a standard Gibbs sampler. Besides the number of variables in the PDN and the number samples, the run time also depends on the number of trees used in the stage-wise optimization. However, evaluating regression trees is done rapidly and hence, calculating λ_i can be done efficiently.

From the collected samples we can then compute an approximate marginal distribution or MAP assignment. If we are interested in the MAP assignment, we simply select the configuration occurring most often in our samples. Due to the nature of count variables, the number of different configurations can be fairly large in several cases. We can then also approximate the MAP assignment by looking at the marginal probabilities and picking the most probable state for each variable individually.

The order of the Gibbs, however, actually does matter in the case of PDNs due to conflicting β_{ij} and β_{ji}. Consequently, we follow Bengio et al. (2014) and use an unordered Pseudo Gibbs sampler in which in each step one randomly chooses a X_i. Bengio et al. showed that this unordered Pseudo Gibbs sampler induces a so-called Generative Stochastic Network (GSN) Markov chain. As long as this GSN Markov chain has a stationary distribution, the DN defines a joint distribution, which, however, also does not have to be known in closed form. This is for example the case, if the chain is ergodic.

From this perspective, PDNs aim to estimate the generating distribution of multivariate count data indirectly, by boosting the transition operator of a Markov chain rather directly. Since all means are positive, and hence $p(\mathbf{x}) > 0$, the chain is ergodic and we will arrive at a consistent estimator of the associated joint distribution.

6 Experimental Evaluation

Our intention here is to investigate the usefulness of PDNs and to show its benefits in comparison to other Poisson graphical models. To this aim, we investigated the following questions:

Q1 Can PDNs learn both positive and negative dependences?
Q2 Can Gibbs sampling predict good counts?
Q3 Can PDNs outperform existing Poisson graphical models?
Q4 Are PDNs easy to interpret?
Q5 Is the strength of the learned dependencies robust to data perturbations?
Q6 Can gradient tree boosting improve the quality of an initial model?
Q7 Can multiplicative updates speed up training, without sacrificing performance?

If all questions can be answered affirmatively, PDNs have the potential to be a valid alternative to existing Poisson graphical models.

We implemented PDNs using a combination of Python and R, and conducted several experiments on synthetic and real-world datasets on a standard single-core Linux machine with 32 GB RAM. To measure the quality of the trained models, we used several measures. Given a training dataset $(\mathbf{x}^{(k)})_{k=1,2,\ldots,m}$, the log-likelihood score on the training data is defined as follows:

$$\text{ll\_score}(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}|\boldsymbol{\theta}) = -(m\cdot n)^{-1}\sum_{j=1}^{m}\sum_{i=1}^{n}\ln p(x_i^{(j)}|\mathbf{x}^{(j)}{}_{\setminus i};\theta_i). \tag{12}$$

Here, θ_i amounts to the set of parameters that define the conditional probability distributions. This can be the $\boldsymbol{\beta}$ in the case of log-linear models or the set of learned trees in the case of boosted models. This also subsumes the initial model, for example the initializing Poisson regression tree. The score on the test data can be calculated accordingly. However, sometimes it is more informative to consider the scores separately for each count variable x_j, i.e., $\text{ll\_score}_j(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}|\boldsymbol{\theta}) = -m^{-1}\sum_{i=1}^{m}\ln p(x_j^{(i)}|\mathbf{x}^{(i)}{}_{\setminus j};\theta_j)$. Both measures can also be used to measure the performance of the predicted counts from the Gibbs sampler. Assuming that $\hat{x}_i^{(j)}$ denotes the predicted value of variable X_i for the data case j and $x_i^{(j)}$ is the true observation of variable X_i, we can calculate the predictive log-likelihood score: $\text{ll\_score}(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)};\hat{\mathbf{x}}^{(1)},\ldots,\hat{\mathbf{x}}^{(m)}|\boldsymbol{\theta})$

$$= -(m\cdot n)^{-1}\sum_{j=1}^{m}\sum_{i=1}^{n}\ln p(x_i^{(j)}|\hat{\mathbf{x}}_{\setminus i}^{(j)};\theta_i).$$

Following convention, we also consider a performance score based on the average rooted-mean-square error

$$\text{RMSE} = \left((m\cdot n)^{-1}\sum_{j=1}^{m}\sum_{i=1}^{n}\left(\hat{x}_i^{(j)} - x_i^{(j)}\right)^2\right)^{\frac{1}{2}},$$

where $\hat{x}_i^{(j)}$ again denotes the predicted value of variable X_i for the data case j and $x_i^{(j)}$ is the true observation of variable X_i. However, especially in the case of Poisson distributed variables with small values, the RMSE is not a good measurement because it assumes a symmetrically shaped error. Moreover, it takes only point predictions into account. A slightly better loss for our task is the normalized RMSE:

$$\text{NRMSE} = n^{-1}\sum_{i}^{n}\frac{\left(m^{-1}\sum_j^m(\hat{x}_i^{(j)} - x_i^{(j)})^2\right)^{\frac{1}{2}}}{x_i^{\max} - x_i^{\min}}.$$

6.1 Network Discovery from Simulated Data (Q1, Q2, Q3)

In order to investigate whether PDNs can model positive and negative dependences we used the simple PDN that was shown already earlier in Fig. 2. This PDN consists of three variables where one variable, namely X_0, only acts as a dummy variable for the constant parameter. The means of X_1 and X_2 are described by log-linear models, e.g., $\lambda_1 = \exp(\beta_1 + \beta_{12}X_2)$. Because X_0 corresponds to the constant feature with value 1, we can omit it from the sum. In turn, this PDN is an LPGM as introduced by Allen and Liu (2013). We chose $\beta_i = \log(10)$, which means that we expect an average value of 10 in the completely independent model. For different choices of the pairwise parameters, we use this PDN to sample 200 instances from

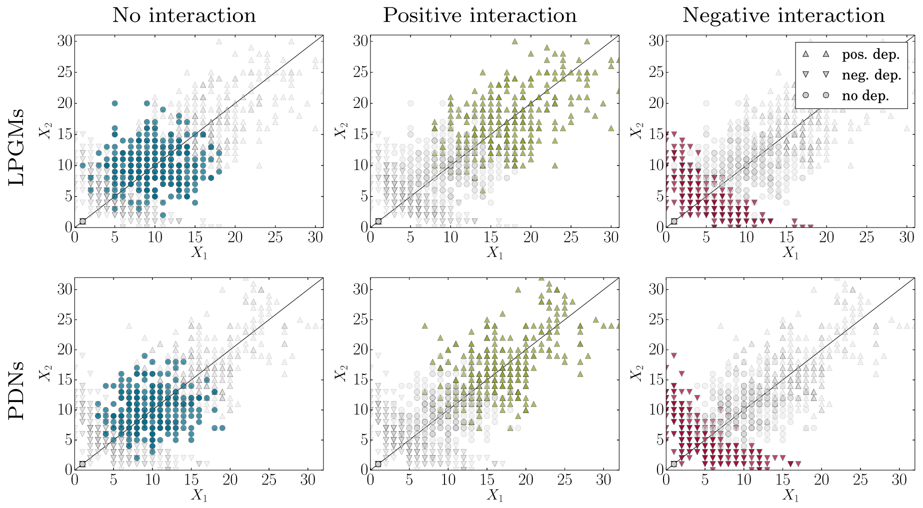


Fig. 4 (*Top*) A log-linear Poisson model. **a** Model without dependencies between X_1 and X_2. The data points (*blue circles*) are close to $\beta_0 = \log(10)$. **b** Model with positive dependencies. The data points (*green triangles*) are now shifted towards the upper right corner. **c** Model with negative dependencies. The data points (*red triangles*) are pushed to the lower corner. (*Bottom*) A boosted PDN. We re-learned a boosted PDN from the samples generated by the log-linear model. One can see that all three groups are well separated by the tree-based model as well (Color figure online)

the joint distribution after a burn-in phase. We consider three different parameter choices of the PDN in total: (i) the independent case (blue circles), (ii) cooperative interactions (green triangles) (iii) competitive interactions (red triangles). For (ii) and (iii) we assumed $\beta_{12} = \beta_{21}$. The obtained samples are plotted in the top row of Fig. 4 (top). As can be observed, the samples drawn with the different parameter settings are well separated and all three types of dependences are captured well. Now we estimated PDNs on this data using additive updates. We then used this model to generate 200 new samples. The results are summarized in the bottom row of Fig 4. Again, it can be observed that the three groups are well separated, highlighting that boosted PDNs can model both positive and negative dependencies. This clearly answers **Q1** affirmatively. In addition, this also highlights the fact that the (pseudo) Gibbs sampler does generate good samples for boosted PDN models. This already provides an affirmative answer to question **Q2**, but we will additionally use the experiments in the next subsection to answer this question using real world datasets.

Before doing so, we will return to the experiments already shown in Sect. 2. For this structure recovery task, we used the code provided by Yang et al. (2013) for both, the data generation and the algorithmic comparison. We employed the "PGM"[6] R-package. The code contains several implementations of the models described in Yang et al. (2013), including SPGMs and LPGMs, and additionally also provides code to sample instances from Winsorized PGMs.

The datasets were generated as follows. First, with the help of the R-package "huge"[7] a graph with $n = 10, 25, 50, 75, 100$ nodes was generated. This data generation process is based on multivariate normal distributions for different graph structures; we created "hub" and "scale-free" graphs. The adjacency matrix of this true graph was used to construct the

6 http://github.com/zhandong/XFam/tree/master/PGMs/Package_Trial/T2.

7 http://cran.r-project.org/web/packages/huge/.

Table 2 Comparison of network discovery performance based on F1-scores (the higher, the better)

	Hub					Scale-free				
	10	25	50	75	100	10	25	50	75	100
PDNs	**0.614**	**0.599**	**0.558**	0.493	0.463	**0.415**	**0.489**	**0.548**	**0.472**	**0.412**
LPGMs	(0.222)	0.400	0.530	**0.545**	**0.554**	(0.100)	0.262	0.372	0.376	0.286
SPGMs	(0.316)	0.310	0.415	0.515	0.473	(0.000)	0.194	0.286	0.327	0.271

Results are averaged over five graphs for each size. LPGMs are the locally learned models of Allen and Liu (2013) and SPGMs are the consistent Poisson PGMs introduced in Yang et al. (2013)
The F1-scores in brackets "(·)" indicate that SPGMs and LPGMs failed to converge on several graphs and hence did not return results for every graph
Bold values indicate best performance

neighborhoods in the PGM model. Together with an unary ($\beta_i = 0$) and a pairwise ($\beta_{ij} = 0.1$) parameter, we then obtained the PGM model. We used the Winsorized PGM Gibbs Simulator contained in the "PGM" package to generate $m = 1000$ samples. This resulted in an $m \times n$ matrix with count values used to learn a model for the original neighborhood graph. More precisely, we used the relative influence to construct an adjacency matrix from our learned models by setting a cell to 1 whenever $I^2(u|v; \lambda_u) > 0$. Similarly, an adjacency matrix can be read off from the parameter matrix of LPGMs and SPGMs. The results in terms of F1-scores are shown in Table 2 and correspond to Fig. 1. As can be seen, PDNs achieve competitive results on the hub graphs and outperform LPGMs and SPGMs on the scale-free networks; the latter ones may even not converge. Moreover, as already shown in Fig. 1c, PDNs can be an order of a magnitude faster.

Because of this, we only compared PDNs to LPGMs in the remaining experiments. In fact, we did ran SPGMs, but on some of the more complex datasets they were not only an order of magnitude slower but did not terminate after days of running; while learning a PDN took less than 30 min. Moreover, we note that in the following LPGMs refer to local Poisson models with GLM mean models learned without regularization. To cover regularization, we also present results for PDNs with Poisson regression trees as mean models, which can be seen as a regularized model.

6.2 Cell Counts and Bibliography Data (Q2, Q3, Q6)

To investigate the quality of predicted counts returned by the Gibbs sampler, we used two different datasets, namely cell counts and counts of publications.

Cell Counts The analysis of microscope images has become a popular application combining biology and computer vision. One particular application is counting the number of cells in an image. To do so, we used the SIMCEP-tool[8], which is described in Lehmussola et al. (2007), to generate 100 microscope images showing cells. An example image is depicted in Fig. 5a. We split each image into 25 patches of equal size, and the cells within each patch were counted (see Fig. 5b). This gives us a dataset with 25 count values per image. We employed tenfold cross validation. More precisely, we used 90 images to train PDNs in various ways over these counts and used the remaining 10 images to do predictions based on the learned model. We are primarily interested in constructing a simple use case for our PDNs and showing that reasonable predictions are possible. Therefore, we randomly removed the counts of five patches in each image for prediction, as depicted in Fig. 5c, and used the trained PDNs to

[8] www.cs.tut.fi/sgn/csb/simcep/tool.html.

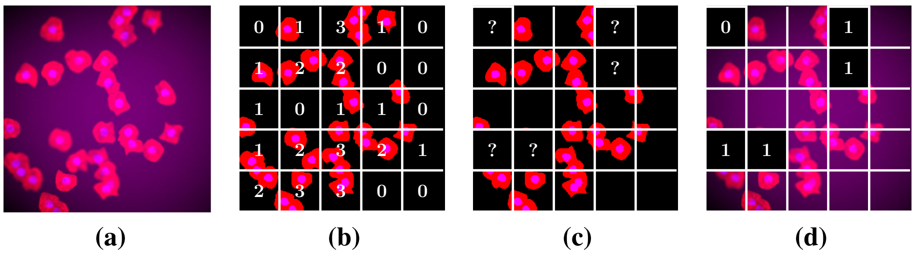


Fig. 5 Prediction of cell counts. We partitioned the original image **a** into 25 patches of equal size. We then counted the number of cells in each patch as depicted in (**b**). To measure the predictive power of our PDNs, we removed the counts of five random patches in the images (**c**) and ran the Gibbs sampler of the partially observed image. We then read off the most likely counts of each missing patch as shown in (**d**) (Color figure online)

Table 3 Tenfold cross validation comparison of the log-scores (the lower, the better) on training and test data for the cell count images (top) and publication data (bottom)

	Boost. Iters.	Train LL	LOO LL	
PDN- $A^{\text{const,log}}$ (w/ $\eta = 1$)	1.91	1.026	1.541	Cells
PDN- $A^{\text{const,log}}$ (w/ selected η)	5.60	1.046	1.428$^{\bullet}$	
PDN- $M^{\text{const,id}}$	1.26	1.167	1.547	
PDN- $M^{\text{const,id}}$ (Laplace)	1.75	1.074	1.511	
PDN- $M^{\text{tree,id}}$	**0.85**	1.134	1.655	
PDN- $M^{\text{tree,id}}$ (Laplace)	1.18	1.049	1.590	
PDN^{tree}	–	1.219	1.525	
LPGM (Allen and Liu 2013)	–	**0.963**	**1.117**$^{\circ}$	
PDN- $A^{\text{const,log}}$ (w/ $\eta = 1$)	x	x	x	Publications
PDN- $A^{\text{const,log}}$ (w/ PARAMILS η)	31.50	0.980	**1.182***	
PDN- $M^{\text{const,id}}$	1.95	1.069	1.248	
PDN- $M^{\text{const,id}}$ (Laplace)	2.68	0.994	1.228	
PDN- $M^{\text{tree,id}}$	**0.23**	0.998	1.232	
PDN- $M^{\text{tree,id}}$ (Laplace)	0.83	**0.955**	1.235	
PDN^{tree}	–	1.004	1.212	
LPGM (Allen and Liu 2013)	–	1.137	1.233	

The “x” indicates that training did not finish properly due to oscillation or overflows. For LOO LL, a “$\bullet$” denotes that boosted PDNs are significantly better then non-boosted PDNs, a “$\circ$” that LPGMs are significantly better than PDNs, and a “$\star$” that PDNs are significantly better than LPGM
Bold values indicate best performance

infer the cell counts for the missing patches. We ran the Gibbs sampler for 50,000 iterations, with an initial burn-in of 5000 iterations which amounts to 10 % of the generated samples, and determined the MAP configuration of the missing patches from the samples produced, cf. Fig. 5d.

The results are summarized in Tables 3 (top rows) and 4 (top rows). As can be seen, additively boosted PDNs—denoted by “A”—produced the lowest NRMSE, followed by multiplicatively boosted PDNs—denoted by “M”. Due to the structure of the problem, an identical

Table 4 Tenfold cross validation comparison of collective prediction performances (the lower, the better) made by the Gibbs sampler on cell count images (top) and publication data (bottom)

	Pred. LL	NRMSE	
PDN- $A^{\text{const,log}}$ (w/ selected η)	**1.429**$^\bullet$	**0.396**$^\bullet$	Cells
PDN- $M^{\text{const,id}}$	1.570	0.408	
PDN- $M^{\text{const,id}}$ (Laplace)	1.503	0.410	
PDN- $M^{\text{tree,id}}$	1.619	0.455	
PDN- $M^{\text{tree,id}}$ (Laplace)	1.576	0.448	
PDN^{tree}	1.511	0.439	
LPGM (Allen and Liu 2013)	–	–	
PDN- $A^{\text{const,log}}$ (w/ ParamILS η)	**0.446**$^\bullet$	0.262	Publications
PDN- $M^{\text{const,id}}$	0.609	0.320	
PDN- $M^{\text{const,id}}$ (Laplace)	0.610	0.268	
PDN- $M^{\text{tree,id}}$	0.525	0.256	
PDN- $M^{\text{tree,id}}$ (Laplace)	0.550	0.254	
PDN^{tree}	0.508	**0.249**	
LPGM (Allen and Liu 2013)	–	–	

A "–" indicates that PDNs with log-linear mean were not able to predict the counts for all instances due to overflows
A "$^\bullet$" denotes that boosted PDN are significantly better than non-boosted PDNs
Bold values indicate best performance

step size was chosen for all variables in PDN- A. It must be mentioned that, we determined a step size of 0.25 by systematic search. Nevertheless we also present experimental results for a step size of 1.0. It can be observed that simply setting $\eta = 1.0$ leads to worse performance and the learning algorithms is not capable of improving over more than two iterations as too large steps are taken. We also tested Laplace smoothing with $\alpha_i = 0.1$ and $\beta_i = 0.2$ in the cases of multiplicative updates. It can be seen that this smoothing improves the quality of the predictions in terms of the predictive log-score. Although PDNs with a log-linear mean model, i.e., LPGMs, produced the best log-score on the test data (LOO LL), they were not able to make meaningful predictions due to overflows. This is mainly due to the fact that patches are often correlated with strong positive weights. Using the Gibbs sampler quickly produces (too) large counts which are meaningless. In comparison, using Poisson regression trees—PDN^{tree}—helps accommodating for highly varying counts and in turn to limit the predicted counts.

Publication Data It has been shown in Hadiji et al. (2013) that migration data based on bibliographic entries exhibits interesting phenomena. Here, we used the AAN (Radev et al. 2009) corpus instead of DBLP and moved from descriptive to predictive models. The goal is to predict the number of publications of a researcher in future years. The AAN corpus at hand contains 19,410 publications written by 15,397 authors from the NLP community. We observe the first 6 years of a researchers' publication record and predict the number of publications for the following 4 years. We take only active researchers into account, i.e., researchers who had publications in the first 3 years of their career. For this experiment, we ran the Gibbs sampler for 40,000 iterations, with an initial burn-in of 4000 iterations (10 %), to obtain predictions on the number of publications of an author.

The tenfold cross-validated results for the training and test likelihoods are summarized in Table 3. Here, boosted PDNs with additive updates achieve the best results. However, one should note that these results were obtained by optimizing the step size for each class separately. Instead of a simple grid search, we used ParamILS (Hutter et al. 2009) to find the best step sizes. In our setting, we obtained step sizes of 0.5, 0.075, 0.05, and 0.075 for the 4 years to predict. Although Allen et al.'s LPGMs—PDNs with log-linear mean-models—do a full maximum likelihood optimization for the parameters, they are not able to outperform the boosted approaches. Most importantly, with only a few iterations of optimization and no ParamILS, multiplicative updates achieved a better train likelihood than LPGMs, without sacrificing the test likelihood much.

Even more interesting are the collective prediction performances as summarized in Table 4. As one can clearly see, PDN$^{\text{tree}}$, i.e., PDNs with PRTs, do the best job. On first sight, it is striking that boosted PDNs with initial tree models and multiplicative updates perform worse than standard PDN$^{\text{tree}}$. We attribute this to the rather small validation set used during boosting and in turn to overfitting.

This is also confirmed by the boosting results using Laplace smoothing with $\alpha_i = 0.1$ and $\beta_i = 0.2$. It significantly reduced the error for PDN- M$^{\text{const,id}}$. In any case, LPGMs, the only model not developed in the present paper, are not capable of predicting all tests instances due to overflows in the Gibbs sampling.

To summarize, both experiments answer question **Q2** affirmatively. Moreover, they already indicate that **Q7** may also be answered affirmatively.

6.3 Bag-of-Word PDNs (Q4, Q5)

To investigate whether PDNs are easy to interpreted, we trained a PDN on a text corpus. Specifically, we used the NIPS bag-of-words data set from the UCI repository[9] containing 1500 documents with a vocabulary above 12k words. We considered the 100 most frequent words in the corpus only and trained an additively boosted PDN on this data, i.e., PDN- A$^{\text{const,log}}$. Fig. 6 shows the dependency structure among the words extracted from the trained PDN using the relative influence I^2. The strength of an edge indicates the value of the relative influence, and the size of a node is relative to the occurrence of the word in the corpus. As one can see, the word dependencies are rather sparse and reflect natural groupings of words. Words such as "neural", "net", "network", "weight", "input", and "unit" respectively "learning", "algorithm","loss" and "function" are inter-related as indicated by the strength of the edges. Generally, the interactions are asymmetric. Even if X_j is important for X_i, i.e., we have an arc from X_j to X_i, there must not necessarily be an arc in the opposite direction. This is due to the fact that the trees for X_j can possibly ignore X_i. For instance, the frequency of "training" has an impact on how often we read "error". However, words such as "model" and "data" are connected by edges of similar weights in both directions. This suggest that they often co-occur.

To investigate the robustness of the relative influence among words extracted from the PDN (**Q5**), we shuffled the NIPS dataset randomly ten times and for each reordering we removed 5, 10 and 15 % of documents from the end. We learned PDNs for each of the subsets of reduced size, and considered the mean and standard deviation of the normalized relative influences among the words induced by the PDNs. The results are depicted in Fig. 7. Here, the strength of the edges represent the mean, and the gray shade is the standard deviation of the relative influence with darker tones indicating small values. As one can see, strong dependencies

[9] https://archive.ics.uci.edu/ml/datasets/Bag+of+Words.

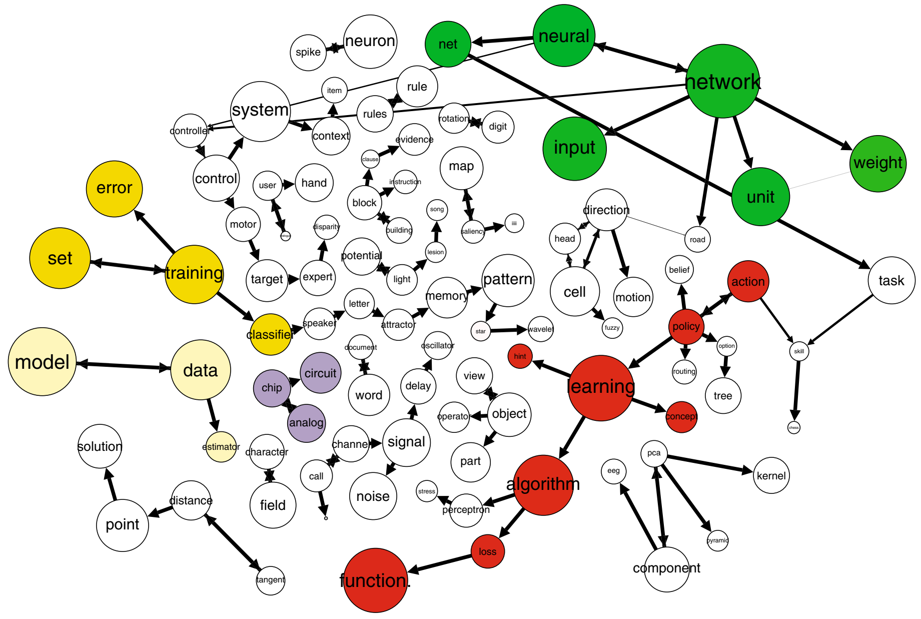


Fig. 6 Dependencies among words of the NIPS corpus induced by the relative influence extracted from the learned PDN. The network reflects natural groupings of words as illustrated for some cases by the *colors* (Color figure online)

are less affected by removing documents from the corpus but the variance indeed increases, mostly by deviation on the strong edges or new small ones.

Overall, the extracted dependency structure in terms of relative influence shows that PDNs can be easily interpreted and used to gain non-trivial insights; an affirmative answer to **Q4**. Furthermore, we can also see that the most important dependencies extracted are rather robust even when removing 15 % of the documents; this answers **Q5** affirmatively.

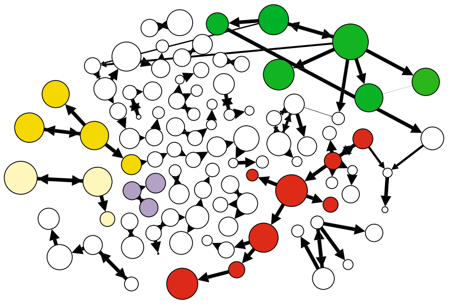
6.4 Communities and Crime and Click-Stream Data (Q6, Q7)

Finally, to investigate the overall performance of PDNs as well as to compare additive and multiplicative updates, we used two real-world datasets, namely the Communities and Crime dataset and a click-stream dataset.

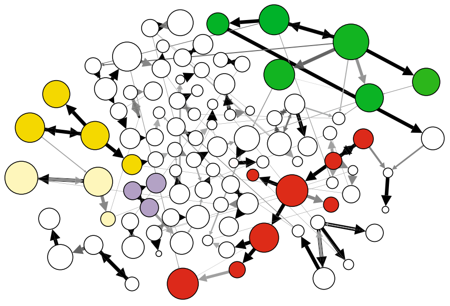
Communities and Crime This dataset from the UCI repository[10] was obtained from 2215 communities in the United States, reporting different crime statistics. It contains 125 observational features presenting different demographics such as the population of a community but also statistics such as the unemployment rate. There are also eight different target statistics per community and we focus on the count values specifying crimes such as the number of robberies, burglaries, and others. The dataset contains missing values for some of the features and for some target variables. For our evaluation we removed communities with incomplete data as well as features that are not available for all communities[11]. Our cleaned data dataset contains 1902 communities with 101 features.

[10] http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized.

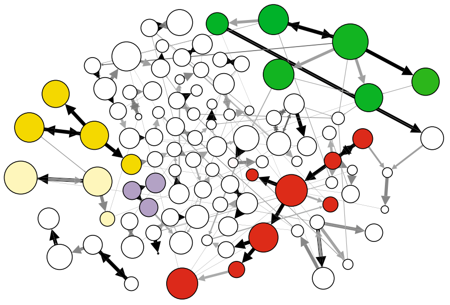
[11] As one further exception, we also removed New York City from the data as it presents an extreme outlier in terms of size.



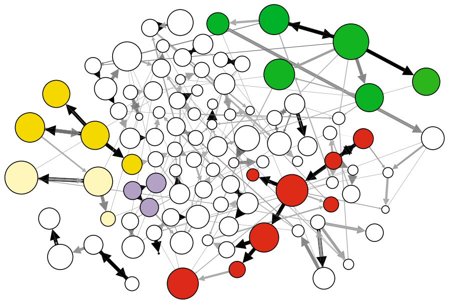
(a) PDN on NIPS corpus.



(b) PDNs on 95% of the corpus.



(c) PDNs on 90% of the corpus.



(d) PDNs on 85% of the corpus.

Fig. 7 Relative influences among words extracted from PDNs learned on subsets of the NIPS corpus of reduced sizes. For ease of comparison, **a** shows the influence structure of the full model as shown in Fig. 6 but without labels. The size of the edges in **b**, **c** and **d** reflect the mean relative influence, and the gray scale is the standard deviation with darker tones indicating lower values. As one can see, the learned influences only vary little as more documents are removed. **a** PDN on NIPS corpus. **b** PDNs on 95 % of the corpus. **c** PDNs on 90 % of the corpus. **d** PDNs on 85 % of the corpus (Color figure online)

For a comparison of the additive and multiplicative updates for PDNs, we used a tenfold cross validation based on the folds defined in the dataset. The bottom plots in Fig. 8 shows the learning curves for the eight different crimes. The plots are averaged over the tenfolds using a maximum of 50 iterations and measure the effectiveness of the learning rate in terms of the log-score per class. For the additive models, PDN- A, we used a step size of $\eta = 0.01$ in case of the identity link function and the log link case used $\eta = 1^{-5}$. Both step sizes were found based on a systematic search. We started with $\eta = 1.0$ and decreased the step size until no more oscillation was observed during the training. In particular, when using a constant mean-model for initialization, one can see that the multiplicative update outperforms the additive ones: it is significantly faster and achieves better test performance. It must be mentioned that this was achieved without a time consuming selection of an adequate step size at all. In case of the additive updates, the log link function learns faster in cases of crimes with high counts such as “larcenies” with data mean around 1999 per community. For means with small values such as “murders” having an empirical mean of around 6.4, however, we see that additive updates using the identity link function learned faster. Moreover, the additive updates always required the maximum of 50 iterations to obtain the best value. We determined the optimal iteration based on a validation set during the learning. On the other hand, multiplicative updates required only a fraction of the iterations, while not sacrificing predictive performance. Averaged over all experiments and classes, multiplicative updates

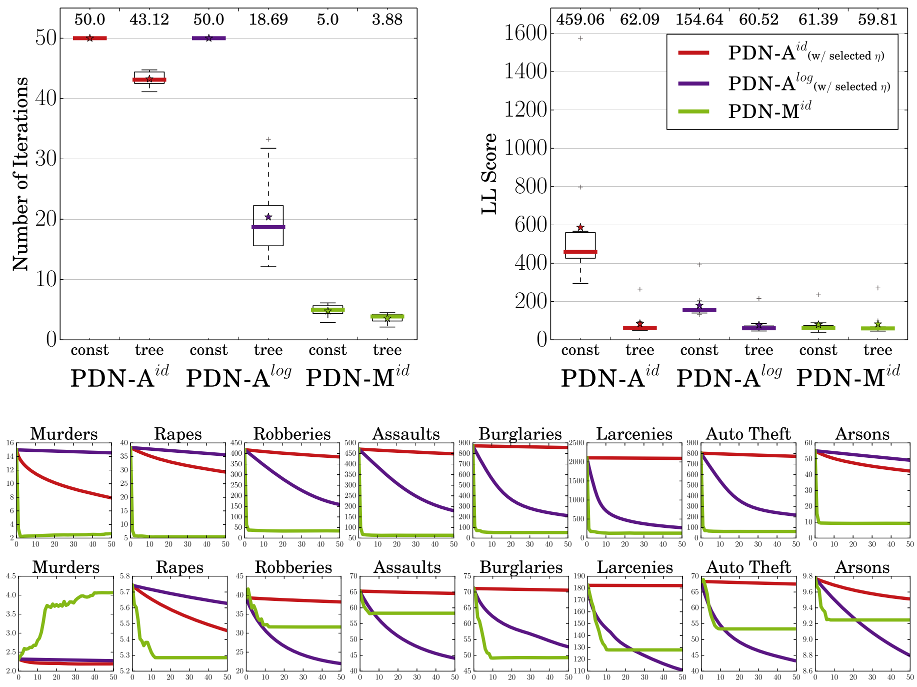


Fig. 8 Comparison of gradient tree boosting PDNs on the CnC-dataset with initial mean or Poisson regression tree models. (*top, left*) The *boxplot* shows the average best iteration found by cross validation. For easier comparison, each *column* shows the median in the *top area* of the plot. (*top, right*) The *boxplot* shows the average ll_score of the best iteration on the test dataset. The results show that the multiplicative updates, PDN- M, require only very few iterations to achieve comparable, or even better, results than the additive updates, PDN- A. This holds for both types of initializations, mean and tree models, even though the step sizes were carefully chosen for PDN- A. (*bottom*) The *two bottom plots* show the learning *curves* in terms of the ll_score$_i$ for the eight different crime types in the dataset. The *first eight plots* show the *curves* for constant initial models. The *curves* highlight that multiplicative updates learn much faster than the additive updates. The *second eight plots* show the learning *curves* for initial regression tree models and indicate that the trees can give an head start (Color figure online)

require far less than ten iterations and the log likelihood score on the test data is significantly better than for the additive case with mean initialization. This clearly shows the advantage of multiplicative updates.

What about using Poisson regression trees for initialization? Looking at the learning curves for PDNs with Poisson regression trees (second set of eight plots in Fig. 8, bottom), we first see that the initial value of the log-score is drastically lower compared to initializing with a constant mean. This is not surprising as the tree can represent the data initially already much better than a single mean value. The number of iterations required drops as well. This is also emphasized in the boxplot that compares the number of iterations (Fig. 8, top, left). On average, the multiplicative updates, PDN- M, add only less than four trees to the initial tree and achieve comparable performance.

Finally, we investigated the robustness of the learned PDNs w.r.t. the numbers of iterations of the stepwise optimization. To this end, we computed the gain in normalized relative influence over the iterations for each fold in the CnC-dataset. The summarized results in Fig. 9 show that the gain in influence decreases with later iterations. That is, the learner focuses on important dependencies first. Influences added at later iterations are less important. The

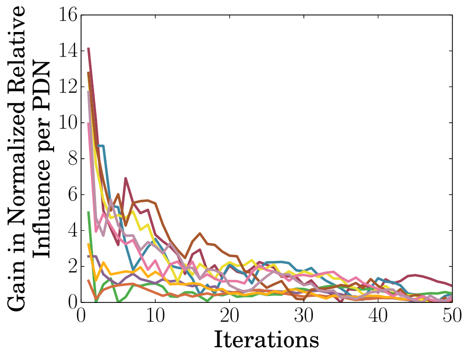


Fig. 9 The gain in normalized relative influence for PDNs over the number of learning iterations. *Each line* represents onefold in the CnC-dataset. The decrease in gain indicates that later iterations have a decreasing influence on the overall model (Color figure online)

important influences are robust w.r.t. to the number of iterations set. This agrees with the robustness results obtained in Sect. 6.3.

To summarize, PDNs outperform their univariate counterpart, boosting improves the initial model, and multiplicative updates outperform additive updates. These are affirmative answers to our initial questions (**Q6, Q7**).

Click-Stream-Data. To reaffirm the results of the last experiment, we created a count dataset based on the *MSNBC.com* dataset from the UCI repository[12]. The data gives sequences corresponding to a user's page views for an entire day, which are grouped into 17 categories. Instead of the original dataset, we used the post-processed version from the *SMPF* library[13], which removed very short click sequences. In total, this dataset contains information of about 31,790 users. We discard the sequence information and instead analyzed solely the frequencies of the visited categories. In contrast to the CnC dataset, the means of the 17 categories have all low mean values. To be more precise, the category "frontpage" has the highest mean (3.62) and the category "msn-news" has the lowest mean (0.03). We also note that the variance in the data is much lower than in the CnC-dataset.

First, we considered boosted PDNs with the empirical mean as initial model again. Since the learning curves were qualitatively identical to the CnC-experiment, we omit them here. Due to the low means and variance of the categories, the simple initial models themselves achieve already a low average log-score of 1.22. Still, boosting was able to improve the model with both, additive and multiplicative updates. Additive updates achieved log-scores of 1.14 (PDN- $\text{A}^{\text{const,id}}$) and 1.08 (PDN- $\text{A}^{\text{const,log}}$), however, requiring more than 40 iterations. With less than four iterations on average, multiplicative updates achieved an average log-score of 1.09. That is, again, multiplicative updates can be an order of magnitude faster, without sacrificing predictive performance. Initialization using Poisson regression trees gives a head start for the additive updates but not for the multiplicative ones; they are still significantly faster. These are affirmative answers to questions **(Q3–7)**.

Taking all results together, the experimental evaluation clearly shows that all six questions **(Q1–7)** can be answered affirmatively and indicate that PDNs have the potential to be a fast alternative to existing Poisson graphical models.

7 Conclusion

Count data are increasingly ubiquitous in data science settings. Example data are bag-of-X representations of, e.g., collections of images or text documents, genomic sequencing data,

[12] https://archive.ics.uci.edu/ml/datasets/MSNBC.com+Anonymous+Web+Data

[13] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

user-ratings data, spatial incidence data, climate studies, and site visits, among others. Unfortunately, standard graphical models such as multinomial or Gaussian ones are often ill-suited for modeling this data. We have therefore introduced Poisson Dependency Networks (PDNs), a new graphical model for multivariate count data. Its representation naturally facilitates a Gibbs sampler and a very simple, non-parametric training procedure: starting from a simple initial model, which in the simplest case can be a constant value, log-linear Poisson model or a Poisson regression tree, PDNs are represented as sums resp. products of regression models grown in a stage-wise optimization. On several real-world datasets we have demonstrated empirically that PDNs are competitive multivariate count models, both in terms of efficiency and predictive performance, although they do not guarantee a consistent joint distribution in closed form.

The intent of our paper has been to introduce and explore the basic idea of non-parametric dependency networks for multivariate count data. Consequently, there is significant additional work to be done. More work is needed to characterize the set of consistent PDNs, i.e., PDNs with a consistent joint distribution in closed form. This would allow one to characterize the complexity of inference in terms of tree-width, adapt other inference methods, e.g., based on message-passing-like inference, and other learning methods, e.g., based on Expectation Maximization. As another example, additional work is needed to understand when the joint distribution of an (inconsistent) PDN has low predictive accuracy. Generally, one should explore PDNs within other machine learning tasks such as characterizing neural dependencies (Berkes et al. 2008), training topic models (Gehler et al. 2006) also capturing word dependencies within each topic (Inouye et al. 2014a, b), predicting user behavior such as retention and churn (Hadiji et al. 2014), and recommendation (Gopalan et al. 2014). Another interesting avenue for future work is to exploit functional gradients for learning hybrid multivariate models. Along the way, one should investigate dependency networks for the complete family of generalized linear models; for instance, Dobra (2009) has shown that hybrid dependency networks among Gaussian and logistic variables perform well for discovering genetic networks, and Guo and Gu have shown logistic (conditional) dependency networks to perform well for multi-label classification (Guo and Gu 2011). Upgrading the resulting non-parametric, hybrid dependency networks to relational domains may provide novel structure learning approaches for BLOG (Milch et al. 2005) and probabilistic programming languages (Goodman 2013), who feature Poisson distributions, and would complement relational Gaussian models (Singla and Domingos 2007; Choi and Amir 2010; Ahmadi et al. 2011) as well as relational copulas as proposed by Xiang and Neville (2013) for relational collective classification; additionally this line of research could pave the way to novel evaluation methods for statistical relational models. In general, copula models for multivariate count data (Hee Lee 2014) are an interesting option, in particular extending them to the relational case based on Xiang and Neville (2013). All this could lead to better methods for mining web-populated knowledge bases such as TextRunner, NELL, YAGO, and KnowledgeGraph. In such open information retrieval tasks, one cannot easily assume a fixed number of “entities” and in turn use existing probabilistic relational models such as Markov Logic Networks: we have to deal with multivariate count distributions.

Acknowledgments The authors would like to thank Ying-Wooi Wan and Zhandong Liu for making the code for Poisson graphical models and for the simulated network recovery experiments from multivariate count data publically available. This work was partly supported by the DFG Collaborative Research Center SFB 876 project B4 and by the DFG, KE 1686/2-1, as part of the Coordination Project SPP 1527.

References

Ahmadi, B., Kersting, K., & Sanner, S. (2011). Multi-evidence lifted message passing, with application to PageRank and the Kalman filter. In *Proceedings of the 22nd international joint conference on artificial intelligence (IJCAI)*.

Allen, G., & Liu, Z. (2013). A local poisson graphical model for inferring networks from sequencing data. *IEEE Transactions on Nanobioscience*, *12*, 189–198.

Bengio, Y., Thibodeau-Laufer, É., Alain, G., & Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31th international conference on machine learning (ICML)* (pp. 226–234).

Berkes, P., Wood, F., & Pillow, J. (2008). Characterizing neural dependencies with copula models. In *Proceedings of the twenty-second annual conference on neural information processing systems (NIPS)* (pp. 129–136).

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, *36*(2), 192–236.

Bolker, B. M., Brooks, M. E., Clark, C. J., Geange, S. W., Poulsen, J. R., Stevens, M. H. H., et al. (2009). Generalized linear mixed models: A practical guide for ecology and evolution. *Trends in Ecology and Evolution*, *24*, 127–135.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. Belmont: Wadsworth.

Bucila, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the twelfth ACM SIGKDD international conference on knowledge discovery and data mining (KDD)* (pp. 535–541).

Chaudhuri, P., Lo, W. D., Loh, W. Y., & Yang, C. C. (1995). Generalized regression trees. *Statistica Sinica*, *5*, 641–666.

Chen, Y., Pavlov, D., & Canny, J. (2009). Large-scale behavioral targeting. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining (CIKM)* (pp. 209–218).

Choi, J., & Amir, E. (2010). Lifted inference for relational continuous models. In *Proceedings of the 26th conference on uncertainty in artificial intelligence (UAI)*.

Clarke, R. D. (1946). An application of the poisson distribution. *Journal of the Institute of Actuaries*, *72*, 481.

Dietterich, T. G., Hao, G., & Ashenfelter, A. (2008). Gradient tree boosting for training conditional random fields. *Journal of Machine Learning Research*, *9*, 2113–2139.

Dobra, A. (2009). Variable selection and dependency networks for genomewide data. *Biostatistics*, *19*, 621–639.

Dobra, A., & Gehrke, J. (2002). SECRET: A scalable linear regression tree algorithm. In *Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining (pp. 481–487)*.

Elith, J., Leathwick, J., & Hastie, T. (2008). A working guide to boosted regression trees. *Journal of Animal Ecology*, *77*, 802–813.

Feller, W. (1968). *An introduction to probability theory and its applications*. London: Wiley.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232.

Gehler, P., Holub, A., & Welling, M. (2006). The rate adapting poisson model for information retrieval and object recognition. In *Proceedings of the twenty-third international conference (ICML)* (pp. 337–344).

Geman, S., & Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*, 721–741.

Ghitany, M., Karlis, D., Al-Mutairi, D., & Al-Awadhi, F. (2012). An em algorithm for multivariate poisson regression models and its application. *Applied Mathematical Sciences*, *6*, 6843–6856.

Goodman, N. (2013). The principles and practice of probabilistic programming. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)* (pp. 399–402).

Gopalan, P., Charlin, L., & Blei, D. (2014). Content-based recommendations with poisson factorization. In *Proceedings of the annual conference on neural information processing systems (NIPS)* (pp. 3176–3184).

Guo, Y., & Gu, S. (2011). Multi-label classification using conditional dependency networks. In *Proceedings of the 22nd international joint conference on artificial intelligence (IJCAI)* (pp. 1300–1305).

Hadiji, F., Kersting, K., Bauckhage, C., & Ahmadi, B. (2013). GeoDBLP: Geo-tagging DBLP for mining the sociology of computer science. arXiv preprint arXiv:1304.7984.

Hadiji, F., Sifa, R., Drachen, A., Thurau, C., Kersting, K., & Bauckhage, C. (2014). Predicting player churn in the wild. In *Proceedings of the IEEE conference on computational intelligence and games (CIG)*.

Heckerman, D., Chickering, D., Meek, C., Rounthwaite, R., & Kadie, C. (2000). Dependency networks for density estimation, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, *1*, 49–76.

Hoff, P. (2003). Random effects models for network data. In R. Breiger, K. Carley, & P. Pattison (Eds.), *Dynamic social network modeling and analysis: Workshop summary and papers* (pp. 303–312). Washington: The National Academies Press.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Inouye, D., Ravikumar, P., & Dhillon, I. (2014a). Admixture of poisson mrfs: A topic model with word dependencies. In *Proceedings of the 31th international conference on machine learning (ICML)* (pp. 683–691).

Inouye, D., Ravikumar, P., & Dhillon, I. (2014b). Capturing semantically meaningful word dependencies with an admixture of Poisson MRFs. In *Proceedings of the annual conference on neural information processing systems (NIPS)* (pp. 3158–3166).

Kaiser, M. S., & Cressie, N. (1997). Modeling poisson variables with positive spatial dependence. *Statistics and Probability Letters*, *35*(4), 423–432.

Karlis, D. (2003). An EM algorithm for multivariate poisson distribution and related models. *Journal of Applied Statistics*, *30*, 63–77.

Karlis, D., & Ntzoufras, I. (2003). Analysis of sports data by using bivariate poisson models. *Journal of the Royal Statistical Society: Series D (The Statistician)*, *52*(3), 381–393.

Kersting, K., & Driessens, K. (2008). Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the twenty-fifth international conference (ICML)* (pp. 456–463).

Khot, T., Natarajan, S., Kersting, K., & Shavlik, J. (2011). Learning markov logic networks via functional gradient boosting. In *Proceedings of the 11th IEEE international conference on data mining (ICDM)* (pp. 320–329).

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models*. Cambridge: The MIT Press.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings 18th international conference on machine learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Lee, E. H. (2014). Copula analysis of correlated counts. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in econometrics* (Chap. 16, pp. 325–348). Bradford: Emerals Group Publishing.

Lee, D., & Seung, H. S. (2000). Algorithms for non-negative matrix factorization. In *Proceedings of neural information processing systems (NIPS)* (pp. 556–562).

Lehmussola, A., Ruusuvuori, P., Selinummi, J., Huttunen, H., & Yli-Harja, O. (2007). Computational framework for simulating fluorescence microscope images with cell populations. *IEEE Transactions on Medical Imaging*, *26*(7), 1010–1016.

Lowd, D., & Davis, J. (2014). Improving Markov network structure learning using decision trees. *Journal of Machine Learning Research*, *15*(1), 501–532.

McCullagh, P., & Nelder, J. (1989). *Generalized linear models*. London: Chapman and Hall.

Meinshausen, N., & Bühlmann, P. (2006). High dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, *34*(3), 1436–1462.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., & Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI)* (pp. 1352–1359).

Natarajan, S., Kersting, K., Khot, T., & Shavlik, J. (2014a). *Boosted statistical relational learners: From benchmarks to data-driven medicine*. Berlin: Springer.

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., & Shavlik, J. (2012). Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning Journal*, *86*(1), 25–56.

Natarajan, S., Leiva, J. M. P., Khot, T., Kersting, K., Re, C., & Shavlik, J. (2014b). Effectively creating weakly labeled training examples via approximate domain knowledge. In *ILP*.

Natarajan, S., Saha, B., Joshi, S., Edwards, A., Khot, T., Davenport, E. M., et al. (2014c). Relational learning helps in three-way classification of alzheimer patients from structural magnetic resonance images of the brain. *International Journal of Machine Learning and Cybernetics, 5*(5), 659–669.

Radev, D., Muthukrishnan, P., & Qazvinian, V. (2009). The ACL anthology network corpus. In *Proceedings, ACL workshop on natural language processing and information retrieval for digital libraries*. Singapore.

Ravikumar, P., Wainwright, M. J., & Lafferty, J. D. (2010). High-dimensional ising model selection using a l1-regularized logistic regression. *The Annals of Statistics*, *38*(3), 1287–1936.

Ridgeway, G. (2006). Generalized boosted models: A guide to the GBM package. *R vignette*.

Saul, L., & Lee, D. (2001). Multiplicative updates for classification by mixture models. In *Proceedings of neural information processing systems (NIPS)* (pp. 897–904).

Sha, F., Saul, L. K., & Lee, D. D. (2003). Multiplicative updates for large margin classifiers. In *Proceedings of the 16th annual conference on computational learning theory (COLT)* (pp. 188–202).

Singla, P., & Domingos, P. (2007). Markov logic in infinite domains. In *Proceedings of the twenty-third conference on uncertainty in artificial intelligence (UAI)* (pp. 368–375).

Therneau, T. M., Atkinson, B., & Ripley, B. (2011). *rpart: Recursive Partitioning*. http://CRAN.R-project.org/package=rpart

Tsiamyrtzis, P., & Karlis, D. (2004). Strategies for efficient computation of multivariate poisson probabilities. *Communications in Statistics, Simulation and Computation*, *33*, 271–292.

Weiss, J., Natarajan, S., Peissig, P., McCarty, C., & Page, D. (2012). Statistical relational learning to predict primary myocardial infarction from electronic health records. In *Proceedings of the twenty-fourth annual conference on innovative applications of artificial intelligence (IAAI-12)*.

Xiang, R., & Neville, J. (2013). Collective inference for network data with copula latent markov networks. In *Proceedings of the sixth ACM international conference on web search and data mining (WSDM)* (pp. 647–656).

Yang, E., Ravikumar, P., Allen, G., & Liu, Z. (2012). Graphical models via generalized linear models. In *Proceedings of the annual conference on neural information processing systems (NIPS)* (pp. 1367–1375).

Yang, E., Ravikumar, P., Allen, G.I., & Liu, Z. (2013). On poisson graphical models. In *Proceedings of the annual conference on neural information processing systems (NIPS)* (pp. 1718–1726).

Yang, Z., & Laaksonen, J. (2007). Multiplicative updates for non-negative projections. *Neurocomputing*, *71*(1–3), 363–373.