RegRocket: Scalable Multinomial Autologistic Regression with Unordered Categorical Variables Using Markov Logic Networks

IBRAHIM SABEK, University of Minnesota, USA
MASHAAL MUSLEH, University of Minnesota, USA
MOHAMED F. MOKBEL, Qatar Computing Research Institute, Qatar and University of Minnesota, USA

Autologistic regression is one of the most popular statistical tools to predict spatial phenomena in several applications including epidemic diseases detection, species occurrence prediction, earth observation and business management. In general, autologistic regression divides the space into a two-dimensional grid, where the prediction is performed at each cell in the grid. The prediction at any location is based on a set of predictors (i.e., features) at this location and predictions from neighboring locations. In this paper, we address the problem of building efficient autologistic models with multinomial (i.e., categorical) prediction and predictor variables, where the categories represented by these variables are unordered. Unfortunately, existing methods to build autologistic models are designed for binary variables in addition to being computationally expensive (i.e., do not scale up for large-scale grid data such as fine-grained satellite images). Therefore, we introduce *RegRocket*; a scalable framework to build multinomial autologistic models for predicting large-scale spatial phenomena. *RegRocket* considers both the accuracy and efficiency aspects when learning the regression model parameters. To this end, *RegRocket* is built on top of Markov Logic Network (MLN), a scalable statistical learning framework, where its internals and data structures are optimized to process spatial data. *RegRocket* provides an equivalent representation of the multinomial prediction and predictor variables using MLN where the dependencies between these variables are transformed into first-order logic predicates. Then, *RegRocket* employs an efficient framework that learns the model parameters from the MLN representation in a distributed manner. Extensive experimental results based on two large real datasets show that *RegRocket* can build multinomial autologistic models, in minutes, for 1 million grid cells with 0.85 average F1-score.

CCS Concepts: • **Mathematics of computing → Factor graphs**; **Probabilistic reasoning algorithms**; • **Computing methodologies → Learning in probabilistic graphical models**; **Distributed algorithms**;

Additional Key Words and Phrases: Multinomial Spatial Regression, Autologistic Models, Markov Logic Networks, First-order Logic, Factor Graph

ACM Reference Format:
Ibrahim Sabek, Mashaal Musleh, and Mohamed F. Mokbel. 2019. RegRocket: Scalable Multinomial Autologistic Regression with Unordered Categorical Variables Using Markov Logic Networks. *ACM Trans. Spatial Algorithms Syst.* x, x, Article xx (October 2019), 27 pages. https://doi.org/0000001.0000001

Authors' addresses: Ibrahim Sabek, University of Minnesota, Department of Computer Science and Engineering, Minnesota, USA, email:sabek@cs.umn.edu; Mashaal Musleh, University of Minnesota, Department of Computer Science and Engineering, Minnesota, USA, email:musle005@cs.umn.edu; Mohamed F. Mokbel, Qatar Computing Research Institute, Hamad Bin Khalifa Research Complex, Doha, Qatar , University of Minnesota, Minnesota, USA, email:mmokbel@hbku.edu.qa.

2374-0353/2019/10-ARTxx $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

Autologistic regression [3, 6, 26] is an important statistical tool for predicting and analysing spatial phenomena in many scientific domains (e.g., Earth observations [28, 59], Epidemiology [13, 32, 47], Ecology [1, 44], Agriculture [19, 23], Archeology [24] and Management [5, 34]). Unlike standard logistic regression [2, 8] that assumes predictions of spatial phenomena over neighbouring locations are completely independent of each other, autologistic regression takes into account the spatial dependence between neighbouring locations while building and running the prediction model (i.e., neighbouring locations tend to systematically affect each other [53]). Typically, autologistic regression divides the geographical space (e.g., the whole world) by a two-dimensional grid, where each grid cell is represented with a variable indicating the *prediction* of the spatial phenomena in that grid cell, and a set of *predictor* variables (i.e., features) that help predicting the value of this prediction variable. Then, the prediction problem at any grid cell is formulated as: Given a set of predictors defined over this cell along with a set of observed or predicted values at neighbouring cells, estimate the value of the prediction variable at this cell. For example, ornithologists would use autologistic regression to predict the existence of a certain bird species in a given location based on two predictors such as the number of bird observers and the observing duration in this cell, along with the predictions at neighbouring locations [50]. Meteorologists would need to predict the hurricane strength at a certain area based on the wind direction at this area as a predictor and the hurricane level at neighbouring areas. Epidemiologist would need to measure the infection level of a disease (e.g., Ebola) in a certain country based on the preventive care level at this country (i.e., predictor) and the infection levels of surrounding countries.

Myriad applications require the autologistic regression model to be built over large *multinomial* (i.e., categorical) spatial data. Examples of these applications include multinomial brain [37] and satellite images [51] analysis. In these applications, the prediction and/or predictor variables in the regression model are multinomial, which means that the value of any variable comes from a set of possible values (i.e., domain values). However, existing methods for autologistic regression (e.g., see [4, 26, 33, 56]) face two main limitations. The *first limitation* is that these methods are specifically designed for autologistic models with *binary* prediction and predictor variables (i.e., each variable takes either 0 or 1) only, and hence are not applicable for the multinomial case [58]. The *second limitation* is that these methods are prohibitively computationally expensive for large grid data, e.g., fine-grained satellite images [36, 59], and large spatial epidemiology datasets [31]. For example, it could take about week to infer the model parameters using the training data of only few gigabytes [26]. As a means of dealing with such scalability issues, existing techniques tend to sacrifice their accuracy through two simplified strategies: (1) Use only a small sample of the available training data, and (2) Only allow individual pairwise dependency between neighbouring cells. For example, if a prediction cell variable C_1 depends on two neighbouring cells C_2 and C_3, then existing methods assume that C_1 depends on each of them individually, and hence define two pairwise dependency relations (C_1, C_2) and (C_1, C_3). Both approaches lead to significant inaccuracy and invalidate the use of autologistic regression for predicting spatial phenomena of current applications with large-scale training data sets.

In this paper, we introduce *RegRocket*; a scalable framework, which overcomes the above mentioned two limitations, for building autologistic models with *multinomial* prediction and predictor variables. *RegRocket* does not need to sample training data sets. It can support the prediction over grids of 1 million cells in few minutes. Moreover, *RegRocket* allows its users to define high degrees of dependency relations among neighbours, which opens the opportunity for capturing more precise spatial dependencies in regression. For example, for the case where a prediction cell variable C_1 depends on two neighbouring cells C_2 and C_3, *RegRocket* is scalable enough to be able to define a ternary dependency relation (C_1, C_2, C_3), which gives much higher accuracy than having two independent binary relations.

RegRocket overcomes the *first limitation* by extending the standard multinomial logistic regression [12, 29, 30] to include spatial dependencies among prediction variables. We refer to this as *multinomial autologistic regression*. To overcome the *second limitation*, *RegRocket* exploits Markov Logic Networks (MLN) [11] (a scalable statistical learning framework) to learn the multinomial autologistic regression parameters in an accurate and efficient manner. Then, *RegRocket* aims to provide an equivalent first-order logic [16] representation to dependency relations among neighbours in autologistic models. This is necessary to accurately express the autologistic models using MLN. *RegRocket* transforms each neighbouring dependency relation into a predicate with bitwise-AND operation on all variables involved in this relation. For example, a ternary dependency relation between neighbouring variables C_1, C_2 and C_3 is transformed to $C_1 \quad C_2 \quad C_3$. This simple logical transformation allows non-expert users to express the dependency relations within autologistic models in a simple way without needing to specify complex models in a tedious detail.

RegRocket proposes an efficient framework that learns the model parameters over MLN in a distributed manner. It employs a spatially-indexed learning graph structure, namely factor graph [57], along with an efficient weights optimization technique based on gradient descent optimization [63]. *RegRocket* represents the MLN bitwise-AND predicates using the spatially-indexed factor graph. Then, *RegRocket* runs multiple instances of learning algorithms in parallel, where each instance handles the learning process over exactly one factor graph partition. At the end, the obtained results from all learning instances are merged together to provide the final autologistic model parameters. Using the proposed framework, *RegRocket* converges to the optimal model parameters of large prediction grids (e.g., 1 million cells) in just few minutes.

RegRocket is the successor of *TurboReg* [42], from which it is distinguished by: (1) Providing a new MLN representation along with its theoretical foundation for multinomial autologistic models, unlike *TurboReg* that considers binary autologistic regression models only. The MLN representation of *RegRocket* can be considered as a generalization of its counterpart in *TurboReg*. (2) Adapting the MLN transformer, factor graph constructor, and model parameters learner modules of *TurboReg* to efficiently implement the new MLN representation of the multinomial case. (3) Providing experimental study of the different system settings in terms of running time, and prediction accuracy while employing the MLN-based multinomial autologistic models.

We experimentally evaluate *RegRocket* using two real datasets of the daily distribution of bird species [50], and the land cover distribution of Minnesota, USA [35, 54]. We compare the accuracy and scalability of the built autologistic models over each dataset using the basic *RegRocket* and two generalized variations of *RegRocket*, that consider higher neighbouring interactions between predictions, with a state-of-the-art open-source autologistic model computational method, namely ngspatial [25]. Our experiments show that *RegRocket* is scalable to large-scale autologistic models, while achieving a high-level of accuracy in estimating the model parameters.

The rest of this paper is organized as follows: Section 2 gives a brief background of autologistic models, both binary and multinomial, and the MLN framework. Section 3 describes how multinomial autologistic regression is modeled using MLN. Section 4 gives an overview of the *RegRocket* system architecture. Section 5 describes how the first-order logic predicates are generated for multinomial autologistic models. Section 6 provides details about the spatially-indexed factor graph structure. Section 7 illustrates the details of the weights learning phase. Section 8 provides the experimental analysis of *RegRocket*. Section 9 covers the related work, while Section 10 concludes the paper.

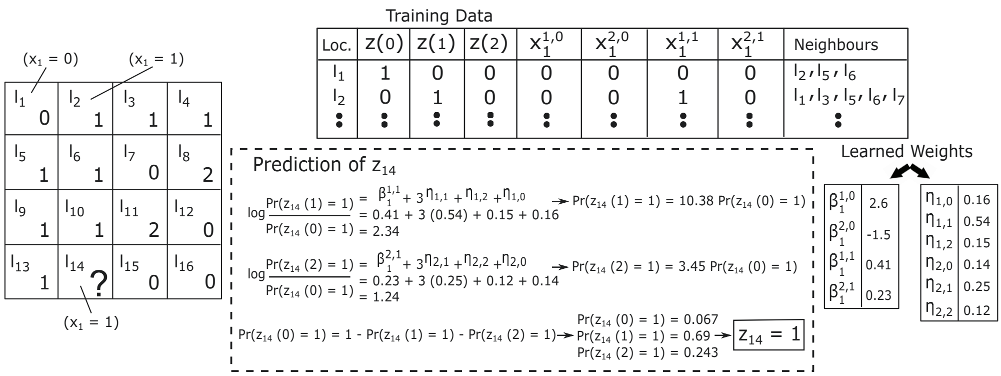


Fig. 1. An Example on Multinomial Autologistic Regression.

2 PRELIMINARIES

This section provides a brief discussion about the autologistic models (Section 2.1) and Markov Logic Networks (MLN) (Section 2.2).

2.1 Autologistic Regression

Binary Models. Binary autologistic regression builds a regression model that predicts the value of a binary random variable (i.e., prediction variable that takes either 0 or 1) at a certain location based on a set of binary predictors (i.e., features that help in the prediction process) at the same location and a set of observed predictions from variables at neighbouring locations (i.e., spatial dependence). Formally, binary autologistic models assume a set of n binary prediction variables $\mathcal{Z} = \{z_1, ..., z_n\}$ (i.e., $z_i \in \{0, 1\}$) at n locations $\mathcal{L} = \{l_1, ..., l_n\}$, and a set of m predictor variables $\mathcal{X}(i) = \{x_1(i), ..., x_m(i)\}$ where the value of each predictor variable $x_j(i)$ is a function of location l_i (e.g., a predicator about the existence of water which could have a different value for each location), and each location l_i has a set of neighbouring locations $\mathcal{N}_i$. Given a specific location l_i, the conditional probability of prediction variable z_i given the values of current predictors $\mathcal{X}$ and the neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as follows [3, 26, 42]:

$$\log \frac{Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i = 0 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})} = \sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k \tag{1}$$

where the weights $\beta = \{\beta_1, ..., \beta_m\}$ and η form the model parameters $\theta = \{\beta, \eta\}$. As shown in Equation 1, for each prediction variable z_i, there are two types of regression terms: *predictor-based* terms $\{\beta_j x_j(i) \mid j = 1, .., m\}$, and *neighbour-based* terms $\{\eta z_k \mid k \in \mathcal{N}_i\}$. Note that the values of θ are shared among all locations $\mathcal{L}$.

Multinomial Models. Binary autologistic models can be extended to the case of multinomial (i.e., categorical) predictions and predictors. In this case, each prediction variable z_i has r possible outcomes $\mathcal{D}_{z_i} = \{\lambda_1, \lambda_2, ..., \lambda_r\}$, and each predictor variable $x_j(i)$ has q possible domain values $\mathcal{D}_{x_j(i)} = \{t_1, t_2, ..., t_q\}$. Since the variables are not binary, the model in Equation 1 is no longer valid for the multinomial case. Our approach to obtain the appropriate model for a prediction variable with r possible outcomes is to build $r - 1$ independent binary models, in which one outcome is chosen as a *pivot* and then the other $r - 1$ outcomes are separately regressed against the pivot outcome. Eventually, the probability of predicting the pivot outcome is calculated based on these built $r - 1$ binary models (i.e., $1 - \sum_{\lambda \neq p} Pr(outcome\ is\ \lambda)$ where p is the pivot outcome).

Such approach is already implemented in classical multinomial logistic regression [12, 29, 30], yet, without considering the spatial dependence (i.e., neighbour-based regression terms).

In the generated binary models, each multinomial prediction variable z_i will be represented with r binary random variables $\{z_i(\lambda) \in \{0, 1\} \mid \lambda \in D_{z_i}\}$, where $z_i(\lambda)$ indicates whether the prediction value at location l_i is λ or not. In addition, each multinomial predictor $x_j(i)$ will be represented as a set of $(r-1)q$ binary random variables $\{x_j^{\lambda,t}(i) \mid \lambda \in D_{z_i} - \{p\}, t \in D_{x_j(i)}\}$, where λ is a non-pivot outcome to be predicted at location l_i and t is a possible domain value of $x_j(i)$. The variable $x_j^{\lambda,t}(i)$ represents a binary predictor (i.e., $x_j^{\lambda,t}(i) \in \{0, 1\}$) in the autologistic regression model that is built for the binary prediction variable $z_i(\lambda)$. Assuming the pivot outcome of prediction variable z_i is λ_r, the $r-1$ conditional probabilities corresponding to z_i given the values of current predictors X and the neighbouring predictions Z_{N_i} can be estimated as follows:

$$
\begin{aligned}
\log \frac{Pr(z_i(\lambda_1) = 1 \mid X(i), Z_{N_i})}{Pr(z_i(\lambda_r) = 1 \mid X(i), Z_{N_i})} &= \sum_{j=1}^{m} \sum_{t \in D_{x_j(i)}} \beta_j^{\lambda_1,t} x_j^{\lambda_1,t}(i) + \sum_{k \in N_i} \sum_{s \in D_{z_k}} \eta_{\lambda_1,s} z_k(s) \\
\log \frac{Pr(z_i(\lambda_2) = 1 \mid X(i), Z_{N_i})}{Pr(z_i(\lambda_r) = 1 \mid X(i), Z_{N_i})} &= \sum_{j=1}^{m} \sum_{t \in D_{x_j(i)}} \beta_j^{\lambda_2,t} x_j^{\lambda_2,t}(i) + \sum_{k \in N_i} \sum_{s \in D_{z_k}} \eta_{\lambda_2,s} z_k(s) \\
&\vdots \\
\log \frac{Pr(z_i(\lambda_{r-1}) = 1 \mid X(i), Z_{N_i})}{Pr(z_i(\lambda_r) = 1 \mid X(i), Z_{N_i})} &= \sum_{j=1}^{m} \sum_{t \in D_{x_j(i)}} \beta_j^{\lambda_{r-1},t} x_j^{\lambda_{r-1},t}(i) + \sum_{k \in N_i} \sum_{s \in D_{z_k}} \eta_{\lambda_{r-1},s} z_k(s)
\end{aligned}
\tag{2}
$$

As shown in Equation 2, each binary predictor $x_j^{\lambda,t}(i)$ is associated with one weight $\beta_j^{\lambda,t}$. Moreover, in contrast to Equation 1 which has one weight η for the whole *neighbour-based* regression terms, the multinomial autologistic model defines a weight $\eta_{\lambda,s}$ for each possible pair of variables $(z_i(\lambda), z_k(s))$, where $z_i(\lambda)$ and $z_k(s)$ correspond to the predictions of a non-pivot outcome λ at location l_i (i.e., $\lambda \in D_{z_i} - \{p\}$) and any outcome s at each neighbouring location $k \in N_i$ (i.e., $s \in D_{z_k}$). The main reason for having multiple η weights is to capture more precise spatial dependencies among neighbouring predictions compared to the one weight in traditional binary models. The objective of our work is to build multinomial autologistic models that achieve both high prediction accuracy and low running time by learning the values of model parameters $\theta = \{\beta, \eta\}$, where $\beta = \{\beta_1^{\lambda_1,t}, \ldots, \beta_m^{\lambda_{r-1},t}\}$ and $\eta = \{\eta_{\lambda_1,\lambda_1}, \ldots, \eta_{\lambda_{r-1},\lambda_r}\}$, from previous observations (i.e., training data) of predictions and predictors at locations L, in a scalable and efficient manner. Note that the total number of weights to be learned in β and η are $mq(r-1)$ and $r(r-1)$, respectively.

Assumptions. In general, prediction and predictor variables can be either binary, multinomial, or continuous. However, we focus only on binary and multinomial variables. The extension to continuous case is out of scope of this paper.

Example. Figure 1 shows a numerical example of multinomial autologistic regression. In this example, we have a 4 x 4 grid (i.e., 16 cells), where each cell l_i has a prediction variable z_i with three possible outcomes (i.e., $D_{z_i} = \{0, 1, 2\}$), and one binary predictor variable $x_1(i)$ (i.e., $D_{x_1(i)} = \{0, 1\}$). Assuming the prediction outcome 0 as pivot, each location i has 3 binary prediction variables $\{z_i(0), z_i(1), z_i(2)\}$, and 4 binary predictor variables $\{x_1^{1,0}(i), x_1^{2,0}(i), x_1^{1,1}(i), x_1^{2,1}(i)\}$. As a result, we have 4 predictor-based and 6 neighbour-based weights. These weights are trained by observations from all locations except l_{14} which is unknown (i.e., needs to be predicted). The example also shows the calculations to predict the value of z_{14} using the learned parameters. The probabilities of the three possible outcomes of z_i are first calculated, and then the outcome corresponding to the highest probability is selected as the prediction value.

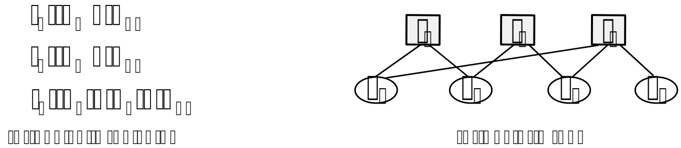


Fig. 2. Translating First-order Logic Predicates into A Factor Graph in MLN.

2.2 Markov Logic Networks

Markov Logic Network (MLN) has recently emerged as a powerful framework to efficiently learn parameters of data models with complex dependencies and distributions [7, 11, 39]. MLN combines probabilistic graphical models (e.g., factor graphs [57]) with first-order logic [16] to perform probabilistic learning based on logic constraints, where the logic handles model complexities and the probability handles uncertainty. MLN has been successfully applied in a wide span of data intensive applications including knowledge bases construction [48], machine learning models [9], and genetic analysis [43]. The success stories in such applications motivate us to explore MLN in computing the parameters of models with spatial dependencies such as autologistic regression.

2.2.1 Modeling with MLN. Any model can be represented with MLN, only if it has two main properties: (1) the model can be represented as a set of p binary random variables v_1 $v_{\boxtimes}$ ($v_{\boxtimes}$ 0 1). (2) the dependencies between model variables can be described with a set of weighted constraints f_1 $f_{\boxtimes}$ defined over them, where these weights w_1 $w_{\boxtimes}$ are the model parameters that need to be learned. The constraints describe how the values of variables correlate with each other. A model with these two properties can exploit MLN to learn the weights that maximize the probability of satisfying the model constraints .

Example. Assume a model of two variables $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$, where $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ denotes whether a person is a professor or not, and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$ denotes whether a person teaches or not. We can define a constraint that "if a person is a professor then she teaches, and vice versa". In this case, MLN learns a weight w that maximizes the probability of $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$ having the same value (i.e., either $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ 1 and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$ 1 or $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ 0 and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$ 0).

2.2.2 First-order Logic. MLN employs first-order logic predicates [16] (e.g., conjunction, disjunction and implication) to represent the model constraints. For example, the constraint defined over $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ and $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$ can be represented as a bitwise-AND predicate $v_{\boxtimes\boxtimes\boxtimes\boxtimes}$ $v_{\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes}$. Efficient logic programming frameworks were proposed to express and generate first-order logic predicates on a large-scale such as DDlog [48] and XLog [46].

2.2.3 Factor Graph. To learn the values of weights associated with predicates (i.e., constraints), MLN translates these predicates into an equivalent probabilistic graphical model, namely factor graph [57], which has weights as the parameters of its joint probability distribution. By doing that, the problem of learning weights is reduced into the problem of learning the joint distribution of this factor graph. A factor graph is a bipartite graph that represents each model variable v and constraint f as a node, and there is an edge between any constraint node f and each variable node v that appears in f.

Figure 2 shows an example of translating three bitwise-AND logical predicates (f_1, f_2, and f_3) defined over a model of four variables (v_1, v_2, v_3 and v_4) into a factor graph.

Probability Distribution. The full joint distribution of variables in a factor graph can be estimated in terms of the constraints (i.e., predicates) and their weights with a log-linear model [11] as follows:

Predicate	Weight
f_1: 1	0
f_2: $z_1(0) \wedge z_2(1)$	$\eta_{2,1}$
f_3: $z_1(0) \wedge z_2(2)$	$\eta_{2,0}$
f_4: $z_1(1) \wedge x_1^{1,0}(1)$	$\beta_1^{1,0}$
f_5: $z_1(1) \wedge x_1^{1,1}(1)$	$\beta_1^{1,1}$
f_6: $z_1(1) \wedge z_2(0)$	$\eta_{1,0}$
f_7: $z_1(1) \wedge z_2(1)$	$\eta_{1,1}$
f_8: $z_1(1) \wedge z_2(2)$	$\eta_{1,2}$
f_9: $z_1(2) \wedge x_1^{2,0}(1)$	$\beta_1^{2,0}$
f_{10}: $z_1(2) \wedge x_1^{2,1}(1)$	$\beta_1^{2,1}$
f_{11}: $z_1(2) \wedge z_2(0)$	$\eta_{2,0}$
f_{12}: $z_1(2) \wedge z_2(1)$	$\eta_{2,1}$
f_{13}: $z_1(2) \wedge z_2(2)$	$\eta_{2,2}$
f_{14}: 1	0
f_{15}: $z_2(1) \wedge x_1^{1,0}(2)$	$\beta_1^{1,0}$
f_{16}: $z_2(1) \wedge x_1^{1,1}(2)$	$\beta_1^{1,1}$
f_{17}: $z_2(2) \wedge x_1^{2,0}(2)$	$\beta_1^{2,0}$
f_{18}: $z_2(2) \wedge x_1^{2,1}(2)$	$\beta_1^{2,1}$

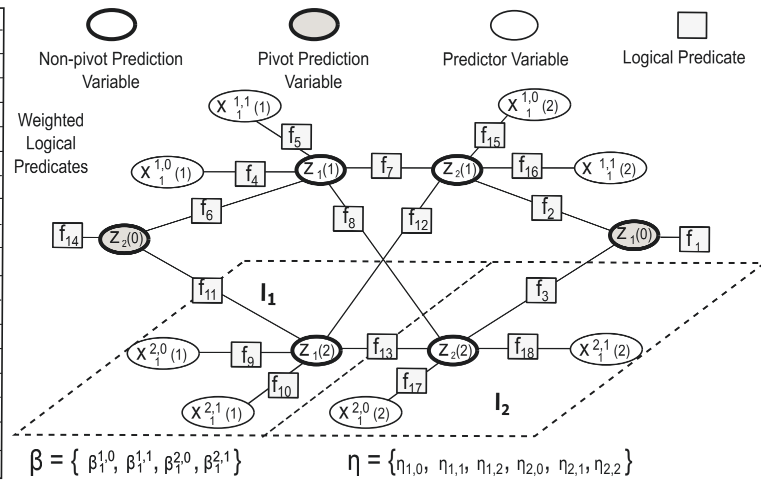


Fig. 3. MLN Representation of A Multinomial Autologistic Model (logical predicates and their factor graph).

$$Pr(\mathcal{V} = v) = \frac{1}{C} \exp\left(\sum_{i=1}^{h} w_i f_i(v)\right) \tag{3}$$

where C is a normalization constant, $f_i(v)$ is the value of whether the i-th constraint is satisfied or not, and w_i is its weight. Scalable optimization techniques have been proposed to efficiently learn the values of weights $\mathcal{W}$ in factor graph, such as gradient descent optimization [27, 60, 63].

3 MULTINOMIAL AUTOLOGISTIC REGRESSION VIA MARKOV LOGIC NETWORK

In this section, we describe how MLN is exploited to efficiently solve the multinomial autologistic regression problem. We start by discussing the MLN-based model for the basic multinomial autologistic regression in Equation 2 (Section 3.1). Then, we extend this model in case of more complicated multinomial autologistic regression scenarios (Section 3.2).

3.1 MLN-based Multinomial Autologistic Model

To represent a multinomial autologistic model using MLN, *RegRocket* extends the MLN-based binary autologistic model in *TurboReg* [42] to support the multinomial case. *TurboReg* represents all binary autologistic regression terms, whether predictor-based or neighbour-based, as a set of weighted bitwise-AND logical predicates (i.e., weighted MLN constraints). In *RegRocket*, we follow the same approach of mapping from regression terms to logical predicates, however, with two main modifications. The first modification is to apply this mapping on each regression term defined in the $r-1$ binary regression models of the multinomial case (Equation 2). For each prediction variable $z_i(\lambda)$ corresponding to a non-pivot possible outcome λ at location l_i, each *predictor-based* regression term $\beta_j^{\lambda,t} x_j^{\lambda,t}(i)$ has an equivalent bitwise-AND predicate defined over $z_i(\lambda)$ and $x_j^{\lambda,t}(i)$ (i.e., $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$) with weight $\beta_j^{\lambda,t}$. Similarly, each *neighbour-based* regression term $\eta_{\lambda,s} z_k(s)$ has an equivalent bitwise-AND predicate defined over $z_i(\lambda)$ and $z_k(s)$ with weight $\eta_{\lambda,s}$. Recall that all prediction and predictor variables in Equation 2 are binary, and hence, it is valid to provide equivalent logical predicates to them. The second modification is to define a constant predicate of value 1 and weight 0 for any prediction variable $z_i(p)$ corresponding to a pivot outcome p at location l_i. The theoretical foundation of the proposed MLN-based multinomial autologistic model is described in the Appendix A. Note that, using the proposed model, the autologistic regression parameters $\theta \quad \beta\ \eta$ are translated into a set of weights of MLN constraints (i.e., proposed equivalent bitwise-AND predicates), and hence learning the autologistic model parameters θ becomes equivalent to learning the values of in MLN (See section 2.2.3).

Example. Figure 3 shows an example of translating a multinomial autologistic regression model into an equivalent MLN. This example defines a model with multinomial prediction of 3 possible outcomes $0\ 1\ 2$ (i.e., three binary prediction variables $z_{\boxtimes}(0)\ z_{\boxtimes}(1)\ z_{\boxtimes}(2)$ at each location $l_{\boxtimes}$) where the pivot outcome is 0, and one multinomial predictor of 2 domain values $0\ 1$ (i.e., four binary predictor variables $x_1^{1\ 0}(i)\ x_1^{2\ 0}(i)\ x_1^{1\ 1}(i)\ x_1^{2\ 1}(i)$ at each location $l_{\boxtimes}$). The model is built for two neighbouring locations $l_1\ l_2$. We first translate the autologistic model into a set of 16 bitwise-AND predicates and 2 constant predicates along with 4 predictor-based weights β_1 $\beta_1^{1\ 0}\ \beta_1^{2\ 0}\ \beta_1^{1\ 1}\ \beta_1^{2\ 1}$ and 6 neighbour-based weights $\eta \quad \eta_{1\ 0}\ \eta_{1\ 1}\ \eta_{1\ 2}\ \eta_{2\ 0}\ \eta_{2\ 1}\ \eta_{2\ 2}$. Then, these predicates are translated into a factor graph which can be used to learn the weights β_1 and η. Note that duplicate predicates that come from neighbouring variables are removed to avoid redundancy (e.g., the neighbouring variables $z_1(2)$ and $z_2(2)$ have two equivalent $z_1(2) \quad z_2(2)$ and $z_2(2) \quad z_1(2)$ neighbour-based predicates, respectively, however, we keep only one of them).

3.2 Generalized Multinomial Autologistic Models

Some applications assume models with more generalized neighbour-based regression terms $\eta_{\lambda\ \boxtimes} G(z_{\boxtimes_1}(s) \quad z_{\boxtimes_{\boxtimes}}(s))\quad k_1 \quad k_{\boxtimes} \quad \boxtimes\ s \quad D_{\boxtimes_{\boxtimes}}$ (i.e., complex spatial dependence), where the regression term has a function defined over neighbouring prediction variables $G(z_{\boxtimes_1}(s) \quad z_{\boxtimes_{\boxtimes}}(s))$, and not just their sum as in Equation 2 (e.g., Ecology [45] and Mineral Exploration [21]). Existing methods can not compute autologistic models with generalized regression terms because of their prohibitively expensive computations, such as high-order matrix multiplications [26]. In contrast, the MLN-based multinomial autologistic model can be easily extended to find an equivalent combination of first-order logic predicates [16] for any generalized regression term, as long as the function $G(z_{\boxtimes_1}(s) \quad z_{\boxtimes_{\boxtimes}}(s))$ holds logical semantics. For example, if the prediction variable $z_1(1)$ at location l_1 has a generalized regression function $G(z_2(2)\ z_3(2))$ over neighbours $z_2(2)$ and $z_3(2)$ which constraints the value of $z_1(1)$ to be 1 only if both values of $z_2(2)$ and $z_3(2)$ are 1 at the same time, then *RegRocket* would translate this into an equivalent bitwise-AND predicate $z_1(1) \quad z_2(2) \quad z_3(2)$. As another example, if $G(z_2(2)\ z_3(2))$ constraints the value of $z_1(1)$ to be 1 only if either $z_2(2)$ or $z_3(2)$ is 1, then it can be translated into a predicate $z_1(1) \quad (z_2(2) \quad z_3(2))$ that has a combination of bitwise-AND and bitwise-OR. Our experiments show that handling generalized regression terms using the MLN-based model increases the learning accuracy while not affecting the scalability performance (See Section 8).

4 OVERVIEW OF REGROCKET

Figure 4 depicts the system architecture of *RegRocket*. It includes three main modules, namely, *MLN Transformer*, *Factor Graph Constructor*, and *Weights Learner*, described briefly as follows:

MLN Transformer. This module receives the autologistic regression model from a *RegRocket* user and generates a set of bitwise-AND and constant predicates of an equivalent MLN. It employs an efficient logic programming framework, called DDlog [48], to produce predicates in a scalable manner. Details are in Section 5.

Factor Graph Constructor. This module prepares the input for the *Weights Learner* module by building a spatially-indexed factor graph out of the generated predicates. The factor graph is partitioned using a flat grid index, where each grid cell has a graph index for its factor graph part. Details are in Section 6.

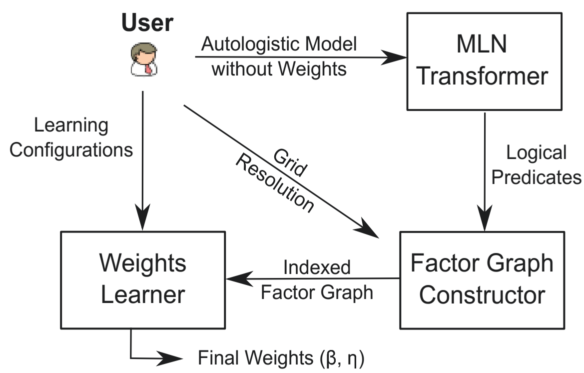


Fig. 4. *RegRocket* System Architecture.

```
#Schema Declaration
z?(@key locId bigint, @key outcomeId bigint, value numeric).
x?(@key locId bigint, @key featureId bigint, @key inDomainId bigint, value numeric).
neighbour(locId1 bigint, outcomeId1 bigint, locId2 bigint, outcomeId2 bigint).

#Derivation Rules
z(locId, outcomeId) ^ x(locId, featureId, inDomainId):- z(locId).
z(locId1, outcomeId1) ^ z(locId2, outcomeId2) :- neighbour(locId1, outcomeId1, locId2, outcomeId2).
@weight(0)  1 :- z(locId, outcomeId), outcomeId = 0.
```

Fig. 5. Example of Using DDlog to Generate Bitwise-AND Predicates for Multinomial Autologistic Model.

Weights Learner. This is the main module in *RegRocket* which efficiently learns the weights that are encoded in the spatially-indexed factor graphs. These weights represent the autologistic model parameters. It takes the built factor graph along with learning configurations (e.g., number of learning epochs) as input, and produces the final values of weights $\theta = \{\beta, \eta\}$. In this module, *RegRocket* provides a scalable variation of gradient descent [63] technique, that is highly optimized for learning the autologistic model parameters. Details are in Section 7.

5 MLN TRANSFORMER

The first step in *RegRocket* is to generate a set of equivalent logical predicates for the different regression terms in the autologistic model. However, this step is challenging in two cases. The first case is when the model has a large number (e.g., millions) of prediction variables $\mathcal{Z}$ and/or predictor variables $\mathcal{X}$, which results in generating a large number of neighbour-based and predictor-based predicates at the end. The number of prediction variables could explode in case of having large 2-dimensional grid (e.g., a 5000×5000 grid) and too many possible outcomes D_{z_i} (e.g., 50 outcomes) for each cell prediction. Similarly, there could be a large number of predictor variables due to a large number of synthetic features, each with too many possible domain values $D_{x_j(i)}$. The second case is when the model has very complicated generalized regression terms, which are translated into predicates with a large number of combinations of first-order logic symbols (e.g., bitwise-AND, bitwise-OR, and imply).

To remedy this challenge, *RegRocket* uses DDlog [48], a DBMS-based logic programming framework, to generate equivalent predicates for any autologistic model in a scalable manner. DDlog takes advantage of the scalability provided by DBMS when generating a large number and/or

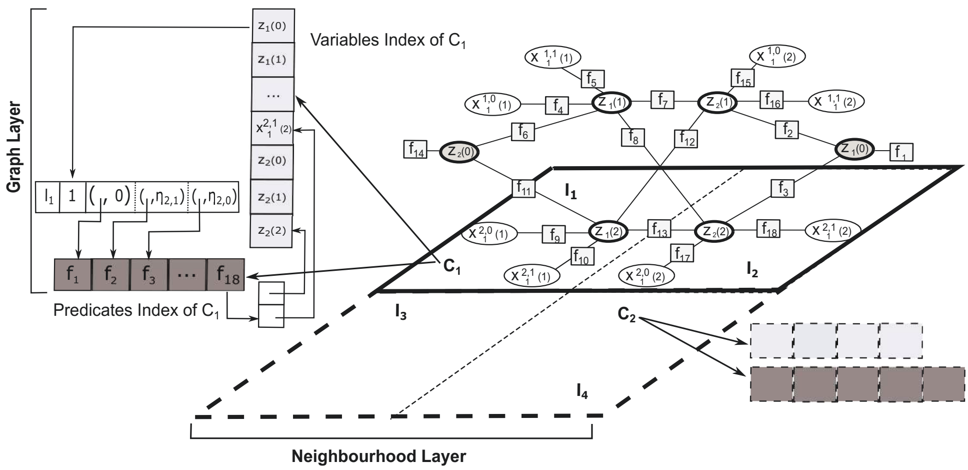


Fig. 6. Example of Spatially-indexed Factor Graph.

combinations of predicates. It provides users with a high-level declarative language to express logical predicates using few template rules. These rules are then translated into SQL queries and applied against the database relations of variables (e.g., $\mathcal{Z}$ and $\mathcal{X}$) to instantiate the actual set of predicates. DDlog has been widely adopted in many applications due to its usability and efficiency (e.g., knowledge bases [48] and data cleaning [40]).

Example. Figure 5 shows an example of using DDlog to express the bitwise-AND predicates of the multinomial autologistic model in Figure 3. DDlog has two types of syntax; *schema declaration* and *derivation rules*. Schema declaration defines the relational schema of variables that appear in predicates. For example, prediction and predictor variables are stored in relations z? and x?, respectively, where any row in each relation corresponds to one binary variable. In case of z?, each row (i.e., variable) stores location ID, outcome ID and the to-be-predicted value of a binary prediction variable in the attributes $locId$, $outcomeId$ and $value$, respectively. Similarly, in case of x?, each row stores location ID, feature ID, possible domain value ID, and the input value of a binary predictor variable in attributes $locId$, $featureId$, $inDomainId$ and $value$, respectively. Note that variable relations are differentiated from normal relations with a question mark at the end of their names. Derivation rules are templates to instantiate predicates. In this example, the first derivation rule is a template for bitwise-AND predicates coming from predictor-based regression terms (i.e., f_4, f_5, f_9, f_{10}, f_{15}, f_{16}, f_{17} and f_{18} in Figure 3), where the body of rule (i.e., right side after symbol ":-") specifies that a predicate is defined over any z and x only if they have the same location id (i.e., selection criteria). During execution, this rule is translated into a hash join between relations z and x with selection predicate over id. Similarly, the second derivation rule is a template for predicates corresponding to neighbour-based regression terms (i.e., f_2, f_3, f_5, f_6, f_7, f_8, f_{11}, f_{12}, and f_{13} in Figure 3), where a predicate is defined for each individual pair of neighbouring predication variables. Finally, the third derivation rule defines the constant predicates (i.e., predicates of value 1 and weight 0) over prediction variables of pivot outcome 0 (i.e., f_1 and f_{14}). Note that the first two derivation rules are not associated with the `@weight` tag because the weights associated with their predicates (i.e., β and η) are still unknown and will be learned later.

6 FACTOR GRAPH CONSTRUCTOR

Figure 6 depicts the organization of a spatially-indexed factor graph for the predicates that are generated in Figure 3. The index is composed of two main layers, namely, *neighbourhood* layer and *graph* layer, described as follows:

6.1 Neighbourhood Index Layer

The neighbourhood index layer is basically a two-dimensional index on the given factor graph. There is already a rich literature on two-dimensional index structures, classified into two categories: *Data-partitioning* index structures (e.g., R-tree [20]) that partition the data over the index and *space-partitioning* index structures (e.g., Quadtrees [14]) that partition the space. In *RegRocket* we decided to go with the Grid Index [38] as an example for space-partitioning data structures because it aligns with the nature of spatial phenomena that are predicted over grids. Having said this, *RegRocket* can accommodate other two-dimensional index structures as a replacement of our grid index. Each grid cell in the neighbourhood layer keeps a graph index for its factor graph part. Figure 6 gives an example of a neighbourhood index layer as a 2-cells grid (i.e., C_1 and C_2), where C_1 contains the factor graph part corresponding to predicates in locations l_1 and l_2, and C_2 holds predicates in locations l_3 and l_4. *RegRocket* takes the grid resolution as input from the user.

6.2 Graph Index Layer

Each cell in the two-dimensional neighbourhood grid points to two indexes of *variables* and *predicates*. Together, these two indexes form the factor graph part in this cell.

Variables Index. This index contains all predication and predictor variables that exist in the grid cell. Each node in the index corresponds to one variable, and points to a list that has three types of information (1) location as a first element in list, (2) value (i.e., 1 or 0) as a second element in the list and (3) predicates that this variable appears in, which are stored as a set of pairs in the rest of list. Each pair consists of a pointer to a predicate in the *predicates* index, and the weight associated with this predicate. Figure 6 shows the details of variable $z_1(0)$ in the variables index.

Predicates Index. This index contains all predicates that are defined over variables in this grid cell. Each node in the index corresponds to one predicate and has a list of pointers to variables that appear in this predicate. Figure 6 shows the details of predicate f_{18} in the predicates index. In the case of predicates with variables in two different cells, we replicate these predicates in each cell.

7 WEIGHTS LEARNER

This is the most important module in *RegRocket*, which takes the spatially-indexed factor graph along with learning configurations from the user and returns the final weights $\theta = \beta \cup \eta$ of the multinomial autologistic model.

Main Idea. A typical solution to efficiently learn the weights W of any MLN model is to provide an approximate log-likelihood function for the full joint distribution in Equation 3 as follows [27]:

$$\log Pr(X = v) = \sum_{i=1}^{n} w_i f_i(v) - \log C \tag{4}$$

Equation 4 provides an objective function to optimize when learning the weights W of the model. As shown in [27], we can estimate the gradient of any weight w_i as follows:

$$\frac{\partial}{\partial w_i} \log Pr(X = v) = f_i(v) - E[f_i(v)] \tag{5}$$

Algorithm 1 Function LearnWeights (FactorGraphCells C, LearningInstances S, LearningEpochs E, StepSize α)

β Random, η Random
e $\frac{\boxtimes}{\boxtimes}$ /* Num. of Learning Epochs Per Instance*/
while $e \neq 0$ **do**
  for all s 1 2 S **do in parallel**
    for all c C **do in parallel**
      $\boxtimes$ Variables index in cell c
      $\boxtimes$ Predicates index in cell c
      for each $v_{\boxtimes}$ $\boxtimes$ **do**
        UpdateWeights ($v_{\boxtimes}$, $\boxtimes$, $\boxtimes$, α) (Algorithm 2)
      end for
  β $\frac{\sum_{\boxtimes 1}^{\boxtimes} \beta_{\boxtimes}}{\boxtimes}$, η $\frac{\sum_{\boxtimes 1}^{\boxtimes} \eta_{\boxtimes}}{\boxtimes}$
  e
end while
return β and η

where $E[f_{\boxtimes}(v)]$ is the expected value of whether the i-th constraint (i.e., predicate) is satisfied or not. Equation 5 can direct how to incrementally converge to the weights that maximize the satisfaction of the MLN model. For example, by applying Equation 5 on the training data, if the gradient value of weight $w_{\boxtimes}$ is positive, then the current assignment of variables in $f_{\boxtimes}(v)$ increases the satisfaction of the MLN model, and hence the corresponding weight $w_{\boxtimes}$ should be rewarded (i.e., should be increased), otherwise it should be punished (i.e., should be decreased). However, estimating the value of $E[f_{\boxtimes}(v)]$ is known to be computationally-expensive in MLN models and requires approximate inference algorithms [27, 49].

As a result, instead of contrasting the satisfied value of the i-th predicate $f_{\boxtimes}(v)$ against its expectation value $E[f_{\boxtimes}(v)]$ (Equation 5), *RegRocket* contrasts the estimated prediction value of the autologistic model (Equation 2) where this predicate belongs to against the corresponding observed prediction from the training data. This approximation has been shown in a recent MLN-based application [48] to converge to the weights that maximize the satisfaction of the MLN model as long as there is no predicate whose observed value is unknown in the training data, which is the case of our regression models. In addition, this approximation is efficient-to-compute as the prediction is estimated by a direct substitution in Equation 2 (i.e., no need for approximate inference algorithms). As an example, to estimate the gradient value of the weight $\beta_{\boxtimes}^{\lambda \boxtimes}$ of the predicate $z_{\boxtimes}(\lambda)$ $x_{\boxtimes}^{\lambda \boxtimes}(i)$, *RegRocket* uses the following two items: (1) the estimated prediction value $\hat{z}_{\boxtimes}(\lambda_v)$ based on the current value of $x_{\boxtimes}^{\lambda \boxtimes}(i)$, and (2) the observed value of $z_{\boxtimes}(\lambda_v)$ from the training data. If the estimated prediction $\hat{z}_{\boxtimes}(\lambda_v)$ is similar to the observed prediction $z_{\boxtimes}(\lambda_v)$, then the weight $\beta_{\boxtimes}^{\lambda \boxtimes}$ should be rewarded, otherwise it should be punished. To that end, we adapt a variation of the gradient descent optimization [63] that punishes and rewards weights using the proposed gradient approximation. The details of algorithms that implement this idea are described below.

LearnWeights Algorithm. Algorithm 1 depicts the pseudo code for our scalable weights learner that takes the following four inputs: the spatially-partitioned factor graph C, the number of learning instances S that can run in parallel, the number of learning iterations E needed to converge to the final values of weights, and the step size α which is a specific parameter for the optimization algorithm 2 that will be described later. The algorithm keeps track of the current best values of

Algorithm 2 Function UPDATEWEIGHTS (Variable v, VariablesIndex , PredicatesIndex , StepSize α)

```
if v is not a prediction variable for a pivot outcome, and belongs to the training data then
    l⊠   [v].location, g   1 /* Gradient Value */
    ẑ⊠(λ_v)   Prediction of outcome λ_v at l⊠ using β and η (Equation 2)
    if   [v].value ≠ ẑ⊠(λ_v) then
        g   -1
    end if
    if v is any predictor variable x⊠^{λ_v ⊠_v}(i)   (i) then
        β⊠^{λ_v ⊠_v}   β⊠^{λ_v ⊠_v} + α g /* Gradient Descent on β⊠^{λ_v ⊠_v} */
    else
        for each β⊠^{λ_v ⊠_v}   β do
            β⊠^{λ_v ⊠_v}   β⊠^{λ_v ⊠_v} + α g /* Gradient Descent on β⊠^{λ_v ⊠_v} */
        end for
    end if
    if v is prediction variable z⊠(λ_v) then
        for each p   [v] do
            if p is a neighbour-based predicate then
                ẑ⊠(s⊠)   Prediction of outcome s⊠ at neighbour l⊠ in p using β and η
                if   [v⊠].value ≠ ẑ⊠(s⊠) then
                    g   -1
                else
                    g   1
                end if
                η_{λ_v ⊠⊠}   η_{λ_v ⊠⊠} + α g /* Gradient Descent on η_{λ_v ⊠⊠} */
            end if
        end for
    end if
end if
```

weights through variables: β and η, initialized by random values. The algorithm then starts by computing the number of learning epochs that can be handled per each learning instance and stores it in a variable e. Note that e represents the actual number of learning epochs that run sequentially because different learning instances execute in parallel. Each of these learning instances then starts to process one learning epoch in parallel (i.e., S learning epochs are running simultaneously). In such learning epoch, we learn an optimal instance of weights $\beta_{\boxtimes}$ and $\eta_{\boxtimes}$, where these values are incrementally learned from variables in the factor graph using UPDATEWEIGHTS function (Line 9 in Algorithm 1)(details of this function are described in Algorithm 2). To reduce the learning latency, we process the variables from different factor graph partitions in parallel (Lines 5 to 10 in Algorithm 1). After all learning instances finish their current learning epoch, we set the values of β and η with the average of the obtained weights from these instances (Line 11 in Algorithm 1) and then proceed to another learning epoch with the new weights. We repeat this process e times and then return the final values of weights.

UPDATEWEIGHTS Algorithm. Algorithm 2 gives the pseudo code for our weights optimizer that applies the gradient descent optimization [63] technique to incrementally update the values of weights given a certain variable v (either non-pivot prediction or predictor) from the training data.

Assume the outcome that appears in v is λ_v. Similarly, in case v is a predictor variable, assume the possible domain value in v is t_v. The main idea is to punish or reward current weights based on their performance in correctly estimating the prediction value $z_{\boxtimes}(\lambda_v)$ at location $l_{\boxtimes}$ where the variable v belongs to.

The algorithm takes the following inputs: a variable v that belongs to the training data, the variables and predicates indexes in the grid cell containing v (i.e., graph index), and a step size α that controls the amount of punishing/rewarding during the optimization process. The algorithm keeps track of the current status of whether weights need to be punished or rewarded through a variable g, where it takes either 1 in case of rewarding or 1 in case of punishing, and is initialized by 1. The algorithm starts by estimating the prediction $\hat{z}_{\boxtimes}(\lambda_v)$ at location $l_{\boxtimes}$ that contains v based on the current values of β and η using Equation 2. If the estimation $\hat{z}_{\boxtimes}(\lambda_v)$ never matches the observed prediction value from the training data, then we set the status g to 1 (i.e., the associated weight with current variable v needs to be punished), otherwise the status remains rewarding. In case v is a predictor variable $x_{\boxtimes}^{\lambda_v \boxtimes_v}(i)$, we only update its associated weight $\beta_{\boxtimes}^{\lambda_v \boxtimes_v}$ by evaluating the gradient descent equation using the current values of g and α (Line 8 in Algorithm 2) and jump to the end of algorithm. In case v is the prediction variable $z_{\boxtimes}(\lambda_v)$ itself, we apply the gradient descent optimization on all weights β associated with its predictors (Lines 10 to 12 in Algorithm 2), and on all weights η associated with the neighbouring predicates (Lines 15 to 25 in Algorithm 2) as well.
Complexity. The complexity of the two aforementioned algorithms can be estimated as $O(\frac{\boxtimes}{\boxtimes}\frac{(\boxtimes^2\boxtimes^2 \;\; \boxtimes\boxtimes\boxtimes\;\boxtimes(\boxtimes \;\; 1))}{\boxtimes})$ where nr is the number of prediction variables, $mq(r \quad 1)$ is the number of predictor variables, C is the number of factor graph partitions, E is the number of learning epochs and S is the number of learning instances. This complexity can be further approximated to be $O(\frac{\boxtimes}{\boxtimes}\frac{(\boxtimes^2\boxtimes^2)}{\boxtimes})$. Note that we assume having SC working threads to process C factor graph partitions in each of the S learning instances in parallel.

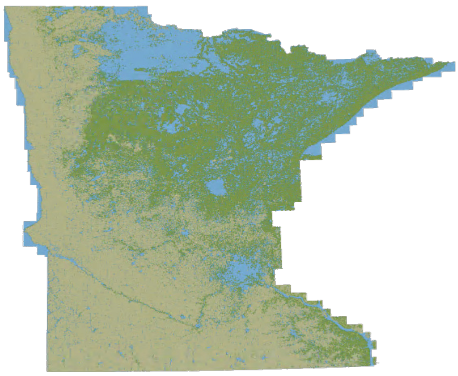
8 EXPERIMENTS

In this section, we experimentally evaluate the accuracy and scalability of *RegRocket* in building multinomial autologistic models (i.e., learning their weights). To the best of our knowledge, *RegRocket* is the first end-to-end system that supports multinomial autologistic regression (see Section 9). As a result, we compare the performance of *RegRocket* with multinomial models built on top of a state-of-the-art binary autologistic regression package, namely ngspatial [25]. Specifically, we compare our performance with multinomial models built on top of the most accurate algorithm in ngspatial that employs Bayesian inference using Markov Chain Monte Carlo (MCMC) [26]. In addition, we extensively investigate the performance of different variations of *RegRocket* under different grid size (Section 8.2), learning epoch (Section 8.3), optimization (Section 8.4), factor graph partitioning (Section 8.5) and parallelism (Sections 8.6) configurations.

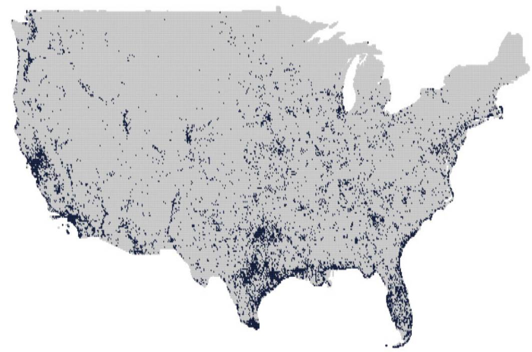
8.1 Experimental Setup

8.1.1 Datasets. All experiments in this section are based on the following two grid datasets:

MNLandCover dataset, which represents the land cover distribution in Minnesota state and is compiled from the USGS National Land Cover [54] and Multi-Resolution Land Cover Consortium [35] data repositories. Figure 7(a) depicts the land cover distribution in Minnesota, where each grid cell is either crops (yellow color), forest (green color) or others (blue color). Thus, we generate a multinomial (i.e., categorical) prediction variable at each grid cell, where each variable takes one of these three possible values. As shown in a recent study [52], the land cover prediction is influenced by three factors; the elevation and slope of the ground as well as the distance to nearby roads. Based on this study, we also generate three multinomial



(a) MNLandCover Dataset



(b) Ebird Dataset

Fig. 7. Datasets Used in Experiments.

predictors corresponding to these factors based on the elevation [10] and transportation [55] datasets of Minnesota at each grid cell. Each predictor takes one value out of 11 possible values. We generate six versions of this dataset with different grid sizes, ranging from 1000 to 1 million cells, to be used during most of our experiments.

Ebird dataset [50], which is a real dataset of the daily distribution of a certain bird species, namely Barn Swallow, over North America. Each grid cell holds a predication of the bird existence in the cell or not (i.e., binary prediction). Figure 7(b) shows the Ebird data distribution, where blue dots refer to cells with bird existence. We generate six versions of this dataset with different grid sizes, ranging from 250 to 84000 cells, to be used during most of our experiments. This dataset has three multinomial predictors at each grid cell including bird observers, observing duration, and the spatial area covered by observers. Each predictor takes one value out of 3 possible values.

8.1.2 Parameters. Unless otherwise mentioned, Table 1 shows the settings of both MNLandCover and Ebird datasets. We select the $250k$ and $84k$ variations of MNLandCover and Ebird, receptively, to be used by default. In each dataset, we divide the cells in each grid into training and testing sets, where we randomly select 20% of cells for testing and keep the rest for training. All training and testing cells have ground truth predictions (i.e., no missing values). During the testing phase, we use the following three inputs to perform the prediction at any testing cell: (1) the learned model parameters, (2) the values of the predictors at this cell, and (3) the ground truth predictions at the neighbours of this cell. Table 2 also shows the default learning configurations that are used with *RegRocket*. In most of the experiments, we run three variations of our system: the basic *RegRocket* that has pairwise neighbourhood relationships (i.e., neighbourhood degree of 1), and other two generalized variations with 4-ary and 8-ary neighbourhood relationships (i.e., neighbourhood degrees of 4 and 8), referred to as *RegRocket-4* and *RegRocket-8*, respectively (See Section 3.2). In *RegRocket-4*, each predication has a bitwise-AND predicate over the vertical and horizontal neighbours surrounding it (i.e., neighbours that share edges with this location only). Similarly, in *RegRocket-8*, each predication has a bitwise-AND predicate over the whole 8 neighbours (i.e., neighbours that share points with this location). Note that the neighbourhood relationships are pre-specified and fixed during both the training and testing phases. Table 3 shows the total number

Parameter	MNLandCover Dataset	Ebird Dataset
Grid Training Size	200000 cells	67200 cells
Grid Testing Size	50000 cells	16800 cells
Number of Predictors	3	3
Number of Possible Predictor Values	11	3
Number of Possible Prediction Values	3	2

Table 1. Default Dataset-specific Parameters.

Parameter	Default Value
Learning Epochs E	1000
Neighbourhood Degree D	1, 4, 8
Step Size α	0.001
Number of Threads	7
Factor Graph Partitions C	200

Table 2. Default Learning-specific Parameters.

Grid Size	*RegRocket*	*RegRocket-4*	*RegRocket-8*
$1k$	$70k$	66 $8k$	66 $7k$
$4k$	$280k$	267 $7k$	267 $4k$
$15k$	1 $05m$	$1m$	998 $5k$
$60k$	4 $2m$	$4m$	3 $9m$
$250k$	17 $5m$	16 $7m$	16 $5m$
$1m$	$70m$	66 $9m$	66 $7m$

Grid Size	*RegRocket*	*RegRocket-4*	*RegRocket-8*
250	3 $2k$	2 $4k$	2 $3k$
$1k$	$13k$	9 $8k$	9 $7k$
3 $5k$	45 $5k$	34 $7k$	34 $5k$
$5k$	$65k$	49 $7k$	49 $4k$
$21k$	$273k$	$209k$	208 $7k$
$84k$	1 $09m$	838 $8k$	837 $6k$

Table 3. Number of Predicates for the MNLandCover (Left) and Ebird (Right) datasets.

of predicates that are generated for both datasets when using the basic *RegRocket*, *RegRocket-4* and *RegRocket-8* during our experiments.

8.1.3 Environment. We run all experiments on a single machine with Ubuntu Linux 14.04, 8 quad-core 3.00 GHz processors, 64GB RAM, and 4TB hard disk.

8.1.4 Metrics. We use the total running time of learning weights as a scalability evaluation metric. To measure the model accuracy, we use the following three metrics to evaluate the prediction quality of each outcome λ (i.e., category):

Precision (Prec.): the number of *correctly* predicted cells with the outcome λ over the total number of predicted cells with the outcome λ.

Recall (Rec.): the number of *correctly* predicted cells with the outcome λ over the total number of testing cells that are actually labelled with the outcome λ from the ground truth.

F1-score (F1): the harmonic mean of precision and recall for the outcome λ as $2 \frac{(Prec \cdot Rec)}{Prec + Rec}$.

To handle the multinomial case, we calculate these three metrics for each outcome, and then report the average of each metric over the total number of outcomes.

8.2 Effect of Grid Size

In this section, we compare the performance, both accuracy and scalability, of basic *RegRocket* and two generalized variations *RegRocket-8* and *RegRocket-4* with ngspatial, while scaling up the

Grid Size	Metric	ngspatial	*RegRocket*	*RegRocket-4*	*RegRocket-8*
1*k*	Prec.	0.498	0.746	0.872	**0.731**
	Rec.	0.491	0.757	0.837	**0.763**
	F1	0.476	0.653	0.708	**0.683**
4*k*	Prec.	0.667	0.803	0.808	**0.933**
	Rec.	0.601	0.834	0.856	**0.871**
	F1	0.606	0.742	0.704	**0.782**
15*k*	Prec.	0.671	0.804	0.906	**0.962**
	Rec.	0.741	0.832	0.898	**0.903**
	F1	0.635	0.721	0.841	**0.834**
60*k*	Prec.	N/A	0.822	0.913	**0.976**
	Rec.	N/A	0.821	0.919	**0.919**
	F1	N/A	0.678	0.736	**0.798**
250*k*	Prec.	N/A	0.864	0.932	**0.967**
	Rec.	N/A	0.893	0.912	**0.915**
	F1	N/A	0.839	0.781	**0.806**
1*m*	Prec.	N/A	0.878	0.929	**0.961**
	Rec.	N/A	0.908	0.931	**0.895**
	F1	N/A	0.859	0.868	**0.873**

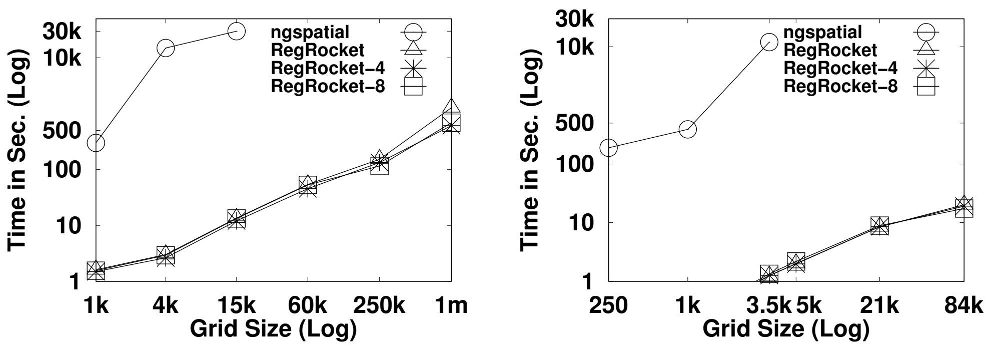
Grid Size	Metric	ngspatial	*RegRocket*	*RegRocket-4*	*RegRocket-8*
250	Prec.	0.551	0.846	0.847	**0.858**
	Rec.	0.951	0.966	0.976	**0.985**
	F1	0.698	0.902	0.907	**0.917**
1*k*	Prec.	0.503	0.801	0.876	**0.883**
	Rec.	0.981	0.986	0.965	**0.961**
	F1	0.665	0.884	0.918	**0.921**
3 5*k*	Prec.	0.477	0.865	0.916	**0.901**
	Rec.	0.977	0.991	0.992	**0.985**
	F1	0.641	0.924	0.952	**0.941**
5*k*	Prec.	N/A	0.885	0.875	**0.912**
	Rec.	N/A	0.984	0.986	**0.984**
	F1	N/A	0.932	0.927	**0.947**
21*k*	Prec.	N/A	0.864	0.866	**0.895**
	Rec.	N/A	0.984	0.991	**0.991**
	F1	N/A	0.921	0.924	**0.941**
84*k*	Prec.	N/A	0.889	0.929	**0.919**
	Rec.	N/A	0.991	0.993	**0.991**
	F1	N/A	0.937	0.956	**0.954**

Table 4. Effect of Grid Size on Accuracy for the MNLandCover (Left) and Ebird (Right) datasets.

prediction grid size. In each experiment, either accuracy or scalability, we report the average of 5 different runs (we follow the same approach in all the experiments in the paper).

Table 4 shows the precision, recall and F1-score values for each algorithm while scaling the grid size from 1k to 1 million cells in case of MNLandCover dataset, and from 250 to 84k cells in case of Ebird one. In all grid sizes, *RegRocket* and its variants *RegRocket-8* and *RegRocket-4* were able to significantly achieve better precision, recall and F1-score results than ngspatial. Specifically, in both datasets, *RegRocket* variants have an average precision of 0.87, recall of 0.92, and F1-score of 0.85, while ngspatial has an average precision of 0.56, recall of 0.79, and F1-score of 0.62. This indicates the efficiency of *RegRocket* in representing multinomial autologistic regression models. Note that the ngspatial results are incomplete after a grid size of 15k cells in case of the MNLandCover dataset, and 3.5k cells in case of the Ebird one, because of a failure in satisfying the memory requirements needed for its internal computations. We can also observe that the F1-score achieved by any *RegRocket* variation in both datasets is at least 0.65, which happens in the MNLandCover dataset with 1k cells, and can reach up to 0.95 at some cases. In general, the accuracy for small datasets tend to be lower than large ones due to the small number of grid cells used to train the model. As we can see from the table, the basic *RegRocket* has at maximum 20% lower F1-score than both *RegRocket-4* and *RegRocket-8*. The reason for that is the basic *RegRocket* captures less accurate neighbourhood dependencies than both of them. Note that *RegRocket-4* and *RegRocket-8* have very close accuracy results in some cases. This happens when the significant information between neighbourhoods with degrees 8 and 4 is very little, which makes the accuracy in the two cases are pretty similar.

Figures 8(a) and 8(b) depict the running time performance of each algorithm while using the same grid sizes in Table 4. We can observe that the three *RegRocket* variants and ngspatial have an average running time of 14 minutes and 8 hours, respectively. This means that *RegRocket* is at least 34 times faster than ngspatial. The poor performance of ngspatial comes from two reasons: (1) although ngspatial relies on parallel processing in its sampling, prior estimation and parameters optimization steps, it runs a centralized approximate Bayesian inference algorithm [26]. In contrast, *RegRocket* is a fully distributed framework. (2) ngspatial requires estimating a prior distribution for each predictor variable, and hence it suffers from a huge latency before starting the actual learning



(a) MNLandCover Dataset (b) Ebird Dataset

Fig. 8. Effect of Grid Size on Scalability.

Num. of Epochs	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
100	Prec.	0.815	0.883	0.906
	Rec.	0.845	0.864	0.854
	F1	0.772	0.732	0.715
1000	Prec.	**0.864**	**0.932**	**0.967**
	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
10*k*	Prec.	0.881	0.931	0.966
	Rec.	0.866	0.909	0.915
	F1	0.826	0.785	0.795

Num. of Epochs	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
100	Prec.	0.849	0.899	0.909
	Rec.	0.845	0.835	0.825
	F1	0.847	0.866	0.865
1000	Prec.	**0.889**	**0.929**	**0.919**
	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.961**	**0.954**
10*k*	Prec.	0.909	0.919	0.919
	Rec.	0.925	0.935	0.995
	F1	0.917	0.927	0.955

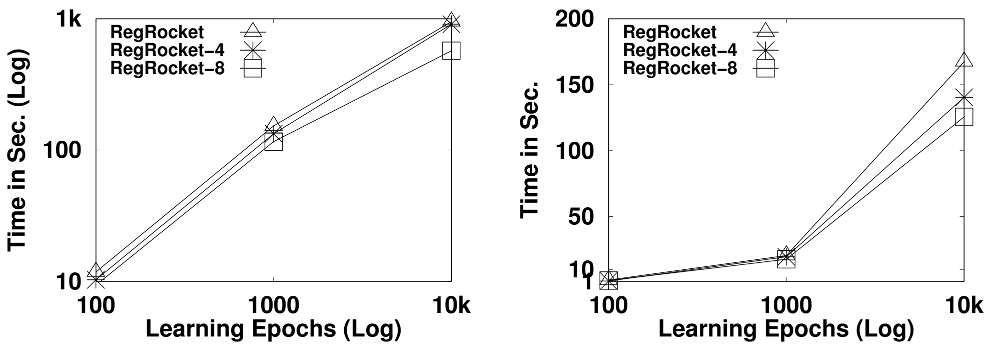
Table 5. Effect of Learning Epochs on Accuracy for the MNLandCover (Left) and Ebird (Right) datasets.

process. Note that the ngspatial curve is incomplete for grids with sizes more than 15k cells in case of the MNLandCover dataset, and 3.5k cells in case of the Ebird one as in Table 4. We also find that in case of datasets with large grids (e.g., 1 million cells in the MNLandCover dataset), both *RegRocket-4* and *RegRocket-8* achieve lower latency overhead than basic *RegRocket*. For example, at 1 million case, *RegRocket-4* and *RegRocket-8* variations are two times faster on average. This is because increasing the neighbourhood degree leads to producing less number of predicates (See Table 3), and hence less number of factor graph nodes to process, which makes the weights learning process faster. In this experiment, the performance of the three *RegRocket* variations are almost similar in case of small grid sizes (i.e., the average accuracy difference between the three variations is less than 20 seconds). However, the difference becomes significant in the case of large grid sizes (average of 600 seconds difference for grid size of 1 million cells). This shows that *RegRocket* is efficient when scaling up the grid size regardless of the neighbourhood degree. Note that both figures 8(a) and 8(b) follow a log-scale.

8.3 Effect of Learning Epochs E

In this section, we evaluate the performance, both accuracy and scalability, of basic *RegRocket*, *RegRocket-4* and *RegRocket-8*, while having three different values of learning epochs. In the following experiments, we fix the grid size in both datasets to the default values.

Table 5 shows the values of accuracy metrics for the three variations of *RegRocket* while changing the number of epochs from 100 to 10k. The results show an interesting observation that all variations



(a) MNLandCover Dataset

(b) Ebird Dataset

Fig. 9. Effect of Learning Epochs on Scalability.

Step Size	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
	Prec.	0.829	0.921	0.966
0.0001	Rec.	0.816	0.789	0.915
	F1	0.782	0.825	0.875
	Prec.	**0.864**	**0.932**	**0.967**
0.001	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
	Prec.	0.819	0.871	0.926
0.01	Rec.	0.806	0.838	0.875
	F1	0.756	0.745	0.795
	Prec.	0.779	0.861	0.916
0.1	Rec.	0.766	0.828	0.865
	F1	0.676	0.745	0.785

Step Size	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
	Prec.	0.914	0.909	0.929
0.0001	Rec.	0.993	0.998	0.995
	F1	0.952	0.951	0.961
	Prec.	**0.889**	**0.929**	**0.919**
0.001	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.956**	**0.954**
	Prec.	0.879	0.909	0.899
0.01	Rec.	0.985	0.985	0.985
	F1	0.929	0.945	0.941
	Prec.	0.779	0.884	0.879
0.1	Rec.	0.985	0.895	0.895
	F1	0.871	0.889	0.887

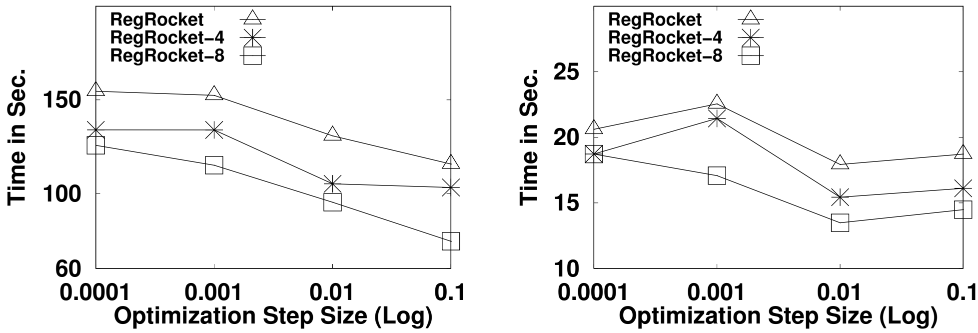
Table 6. Effect of Optimization Step Size on Accuracy for the MNLandCover (Left) and Ebird (Right) datasets.

of *RegRocket* can rapidly converge to their optimal values of weights (i.e., number of learning epochs 1000 only). The rapid convergence happens because weights are shared among all locations which makes their values updated multiple times using the gradient descent optimization in each epoch. As a result, *RegRocket* just needs a small number of epochs for weights convergence. In general, basic *RegRocket* needed a higher number of epochs (i.e., 10k), compared to *RegRocket-4* and *RegRocket-8*, to achieve higher accuracy. This matches our performance hint that generalized variations such as *RegRocket-4* and *RegRocket-8* could be more efficient than the basic *RegRocket*.

Figures 9(a) and 9(b) show the running time for the different variations given the same setup in Table 5. In general, *RegRocket* with all variations is extremely efficient because of the parallel processing of learning epochs in *RegRocket*. However, we observe that *RegRocket-8* significantly outperforms *RegRocket-4* and *RegRocket*. It is at least 40% and 25% faster than both of them in the MNLandCover and Ebird datasets, respectively. Note that Figure 9(a) follows a log-scale, but Figure 9(b) is not.

8.4 Effect of Optimization Step Size α

In this section, we evaluate the performance, both accuracy and scalability, of the different variations of *RegRocket* while varying the value of the step size α that is used during the execution of gradient decent optimization (See Section 7). We use the same datasets used in the previous experiments.



(a) MNLandCover Dataset (b) Ebird Dataset

Fig. 10. Effect of Optimization Step Size on Scalability.

Num. of Partitions	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
	Prec.	0.945	0.962	0.971
50	Rec.	0.894	0.931	0.912
	F1	0.852	0.877	0.914
	Prec.	0.913	0.954	0.961
100	Rec.	0.891	0.923	0.931
	F1	0.843	0.812	0.861
	Prec.	**0.864**	**0.932**	**0.967**
200	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
	Prec.	0.782	0.812	0.815
300	Rec.	0.734	0.831	0.821
	F1	0.689	0.701	0.712

Num. of Partitions	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
	Prec.	0.967	0.944	0.968
50	Rec.	0.992	0.991	0.982
	F1	0.979	0.967	0.975
	Prec.	0.923	0.941	0.937
100	Rec.	0.971	0.981	0.983
	F1	0.946	0.961	0.959
	Prec.	**0.889**	**0.929**	**0.919**
200	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.959**	**0.953**
	Prec.	0.674	0.789	0.792
300	Rec.	0.782	0.712	0.812
	F1	0.724	0.748	0.802

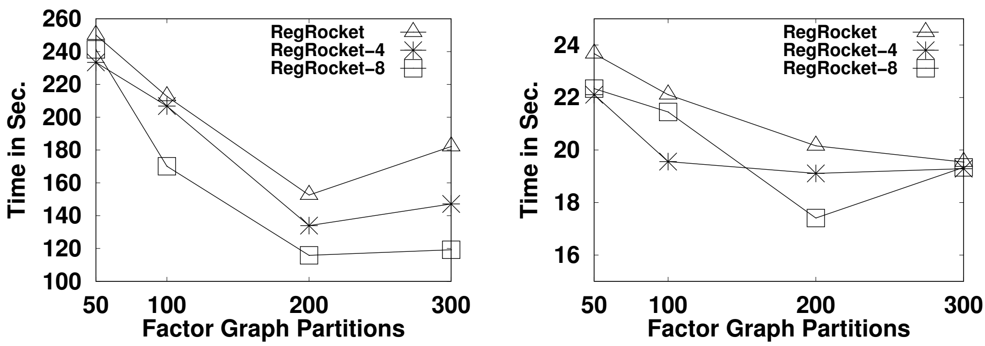
Table 7. Effect of Number of Factor Graph Partitions on Accuracy for the MNLandCover (Left) and Ebird (Right) datasets.

Table 6 depicts the effect of increasing the value of step size from 0.0001 to 0.1. In general, the accuracy of all *RegRocket* variations decreases by increasing the step size in both datasets. We also observe that the highest prediction accuracy tends to saturate in most cases at value 0.001. The main reason behind this is that large step sizes lead to large updates while optimizing the weights and hence they cannot smoothly converge to the optimal values. We conclude that the step size should be kept relatively small on average in *RegRocket*. However, this comes with higher latency as in Figures 10(a) and 10(b) that show the corresponding running times. For example, decreasing the step size from 0.01 to 0.001 in the MNLandCover dataset incurs 26% additional latency overhead while running *RegRocket-4*.

8.5 Effect of Number of Factor Graph Partitions C

In this section, we evaluate the performance, both accuracy and scalability, of the different variations of *RegRocket* while varying the number of factor graph partitions C (See Section 6.1) from 50 to 300. We use the same datasets used in the previous experiments with the default configurations.

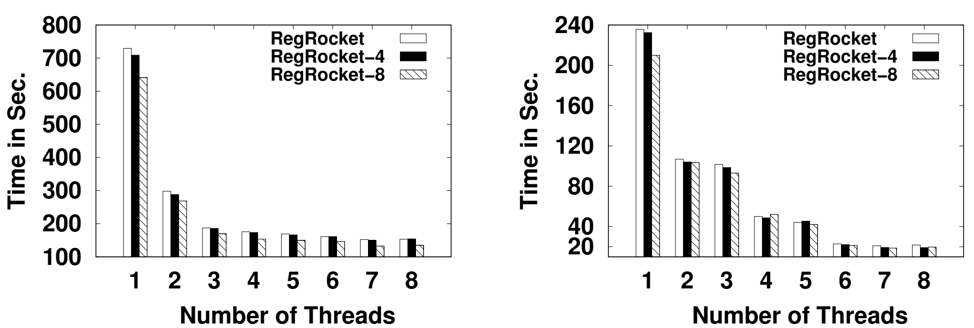
Table 7 shows the effect of increasing the number of factor graph partitions on the precision, recall and F1-score values. We observe that the F1-score values decrease when increasing the number of partitions because the number of predicates that are replicated among partitions increases. This



(a) MNLandCover Dataset

(b) Ebird Dataset

Fig. 11. Effect of Number of Factor Graph Partitions on Scalability.



(a) MNLandCover Dataset

(b) Ebird Dataset

Fig. 12. Effect of Number of Threads on Scalability.

results in more iterations to update the weights as in shown in Algorithm 2 (Lines 15 to 25), which makes the weights suffer from an over-fitting issue, and hence the accuracy decreases. For example, in both datasets, when increasing the number of partitions from 200 to 300, the F1-scores in case of *RegRocket*, *RegRocket-8* and *RegRocket-4* decrease by 17%, 10% and 11%, respectively, on average.

Figures 11(a) and 11(b) depict the running time for the different variations of *RegRocket* given the same setup in Table 7. Increasing the number of partitions leads to more parallelism, and hence the running time starts to decrease. However, increasing the number of partitions after a certain threshold (e.g., 200) makes the running time overhead to handle the predicates replication significant, and hence the whole running time slightly increases again. For example, in both datasets, the average F1-score decrease is 30% in all variations when changing the number of partitions from 100 to 200. After that, the F1-score starts to increase due to the replication overhead. In our experiments, we choose the number of partitions to be 200 in order to balance between the accuracy and the running time.

8.6 Effect of Number of Threads

Figures 12(a) and 12(b) show the effect of increasing the number of threads from 1 to 8 on the three variations of *RegRocket* for both datasets. These threads are used to parallelize the work in the weights learner module of *RegRocket*. As expected, the performance of all variations significantly improves by increasing the number of threads. For example, the running time of the basic *RegRocket* using 8 threads is at least 4 times faster than using 1 thread in the MNLandCover dataset. This shows the ability of *RegRocket* to scale up with system threads. Note that the performance difference between 7 and 8 threads is almost the same because all cores are exploited in both cases.

9 RELATED WORK

In this section, we first provide an overview of existing theoretical models and computational methods of autologistic regression. Then, we briefly mention other related spatial regression models.

Autologistic Theoretical Models. There are two main theoretical models of autologistic regression: (1) *Traditional model* [3] simply estimates the logistic function of the predication probability at any location as a linear combination of predictors at this location and the predictions of its neighbours. However, this model incurs prediction biasing in case of sparse training data. (2) *Centered model* [6] is similar to the traditional model, however, the model parameters are normalized to avoid the biased cases. This adds more complexity when learning the model parameters. Other than *TurboReg* [42], the predecessor of *RegRocket*, all existing implementations of both traditional and centered autologistic models provide a trade-off between the running time complexity and the accuracy of learning model parameters. In contrast, both *RegRocket* and *TurboReg* achieve the performance efficiency while preserving the model accuracy at the same time.

Recent research has proposed an extension for spatio-temporal autologistic models [61, 62] (and centered variants [15, 56]), which incorporates the temporal dependence between predictions at the same location. However, this line of research is out of the scope of this paper.

Autologistic Computational Methods. A wide array of techniques that are capable of learning the autologistic model parameters on a small scale (see [26, 58] for a comprehensive survey, and [25] for open-source implementations). Learning the autologistic model parameters is much harder than learning parameters of classical non-spatial regression models due to the spatial dependence effect. Thus, the techniques are categorized into three main categories based on their methods of approximation to the original parameters distributions: Pseudo likelihood estimation [4, 61](and centered variants [26]), Monte Carlo likelihood estimation [17](and centered variants [26, 61]), Bayesian inference estimation [5, 33] (and centered variants [25, 61]). *TurboReg* [42], conversely, applies Markov Logic Networks (MLN) [11] to learn the autologistic regression parameters, yet, for binary predication and predictor variables only. *RegRocket*, on the other hand, is the first framework to support both multinomial prediction and predictor variables in the same autologistic model.

Other Spatial Regression Models. Autologistic models belong to the class of non-Gaussian spatial modelling [22], in which the spatial dependence between predictions is conditionally modelled through direct neighbours. However, there are three other classes: (1) linear spatial models [22], (2) spatial generalized linear models [18] and (3) Gaussian Markov random field models [41], that encode the spatial dependence through a distance-based covariance matrix. This matrix defines how much the prediction in one location is affected by predictions in all other locations based on their relative distances. Another main difference is that these spatial regression classes are mainly developed for prediction of continuous variables, which is out of the scope of this paper.

10 CONCLUSIONS

This paper has presented *RegRocket*, a scalable framework for building multinomial autologistic regression models to predict spatial categorical data. *RegRocket* focuses on the autologistic models that consist of prediction and predictor variables with unordered categories. *RegRocket* provides an efficient modeling for the multinomial autologistic regression problem using Markov Logic Network (MLN), which is a scalable statistical learning framework. *RegRocket* employs first-order logic predicates, a spatially-partitioned factor graph data structure, and an efficient gradient descent-based optimization technique to learn the autologistic model parameters. Experimental analysis using real data sets shows that *RegRocket* is efficient, scalable and provides accurate capturing for the multinomial autologistic regression problem.

REFERENCES

[1] Nathalie Augustin, Moira A. Mugglestone, and Stephen T. Buckland. An Autologistic Model for the Spatial Distribution of Wildlife. *Journal of Applied Ecology*, 33(2):339–347, 1996.

[2] Colin M. Beale, Jack J. Lennon, Jon M. Yearsley, Mark J. Brewer, and David A. Elston. Regression Analysis of Spatial Data. *Ecology Letters*, 13(2):246–264, 2010.

[3] Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society*, 36(2):192–236, 1974.

[4] Julian Besag. Statistical Analysis of Non-Lattice Data. *Journal of the Royal Statistical Society*, 24(3):179–195, 1975.

[5] Brenda Betancourt, Abel RodrÃŋguez, and Naomi Boyd. Investigating Competition in Financial Markets: A Sparse Autologistic Model for Dynamic Network Data. *Journal of Applied Statistics*, 45(7):1157–1172, 2018.

[6] Petruta C. Caragea and Mark S. Kaiser. Autologistic Models with Interpretable Parameters. *Journal of Agricultural, Biological, and Environmental Statistics*, 14(3):281, 2009.

[7] Yang Chen and Daisy Zhe Wang. Web-Scale Knowledge Inference Using Markov Logic Networks. In *Proceedings of the ICML workshop on Structured Learning: Inferring Graphs from Structured and Unstructured Inputs*, 2013.

[8] David R. Cox. The Regression Analysis of Binary Sequences (with discussion). *Journal of the Royal Statistical Society*, 20:215–242, 1958.

[9] Robert Crane and Luke K. McDowell. Evaluating Markov Logic Networks for Collective Classification. In *Proceedings of the MLG Workshop at ACM SIGKDD*, 2011.

[10] Digital Elevation Model (DEM) of Minnesota. http://www.mngeo.state.mn.us/chouse/metadata/dem24ras.html.

[11] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 2009.

[12] J. Engel. Polytomous Logistic Regression. *Statistica Neerlandica*, 42(4):233–252, 1988.

[13] Juan Ferrandiz, Antonio Lopez, Agustin Llopis, Maria Morales, and Maria Luisa Tejerizo. Spatial Interaction between Neighbouring Counties: Cancer Mortality Data in Valencia (Spain). *Biometrics*, 51(2):665–678, 1995.

[14] R.A. Finkel and J.L. Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[15] Anne Gégout-Petit and Shuxian Li. Two-step Centered Spatio-temporal Auto-logistic Regression Model. In *Applied Statistics for Development in Africa, SADA*, 2016.

[16] Michael Genesereth and Nils Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1987.

[17] Charles J. Geyer. On the Convergence of Monte Carlo Maximum Likelihood Calculations. *Journal of the Royal Statistical Society*, 56:261–274, 1994.

[18] C. A. Gotway and W. W. Stroup. A Generalized Linear Model Approach to Spatial Data Analysis and Prediction. *Journal of Agricultural, Biological, and Environmental Statistics*, 2(2):157–178, 1997.

[19] Marcia L. Gumpertz, Jonathan M. Graham, and Jean B. Ristaino. Autologistic Model of Spatial Pattern of Phytophthora Epidemic in Bell Pepper: Effects of Soil Variables on Disease Presence. *Journal of Agricultural, Biological, and Environmental Statistics*, 1997.

[20] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *ACM SIGMOD Record*, 14(2):47–57, 1984.

[21] Tjelmeland Hakon and Besag Julian. Markov Random Fields with Higher-order Interactions. *Scandinavian Journal of Statistics*, 25(3):415–433, 1998.

[22] Murali Haran. *Gaussian Random Field Models for Spatial Data*, pages 449–478. Chapman and Hall/CRC, 2011.

[23] Fangliang He, Julie Zhou, and Hongtu Zhu. Autologistic Regression Model for the Distribution of Vegetation. *Journal of Agricultural, Biological, and Environmental Statistics*, 8(2):205, 2003.

[24] Jennifer A. Hoeting, Molly Leecaster, and David Bowden. An Improved Model for Spatially Correlated Binary Responses. *Journal of Agricultural, Biological, and Environmental Statistics*, 5(1):102–114, 1999.
[25] John Hughes. ngspatial: A Package for Fitting the Centered Autologistic and Sparse Spatial Generalized Linear Mixed Models for Areal Data. *The R Journal*, 6(2):81–95, 2014.
[26] John Hughes, Murali Haran, and Petruta C. Caragea. Autologistic Models for Binary Data on a Lattice. *Environmetrics*, 22(7):857–871, 2011.
[27] Tuyen N. Huynh and Raymond J. Mooney. Discriminative Structure and Parameter Learning for Markov Logic Networks. In *Proceedings of the International Conference on Machine Learning, ICML*, pages 416–423, 2008.
[28] Nikos Koutsias. An Autologistic Regression Model for Increasing the Accuracy of Burned Surface Mapping using Landsat Thematic Mapper Data. *International Journal of Remote Sensing*, 24(10):2199–2204, 2003.
[29] J. Li, J. M. Bioucas-Dias, and A. Plaza. Spectral-Spatial Hyperspectral Image Segmentation Using Subspace Multinomial Logistic Regression and Markov Random Fields. *IEEE Transactions on Geoscience and Remote Sensing*, 50(3):809–823, 2012.
[30] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer, 3rd edition, 2009.
[31] Catherine Linard and Andrew J. Tatem. Large-scale Spatial Population Databases in Infectious Disease Research. *International Journal of Health Geographics*, 11(1):7, 2012.
[32] Daphne Lopez, M. Gunasekaran, and B. Senthil Murugan. Spatial Big Data Analytics of Influenza Epidemic in Vellore, India. In *Proceedings of the IEEE International Conference on Big Data, BigData*, 2014.
[33] J. Moller, A. N. Pettitt, R. Reeves, and K. K. Berthelsen. An Efficient Markov Chain Monte Carlo Method for Distributions with Intractable Normalising Constants. *Biometrika*, 93(2):451–458, 2006.
[34] Sangkil Moon and Gary J. Russell. Predicting Product Purchase from Inferred Customer Similarity: An Autologistic Model Approach. *Management Science*, 54(1):71–82, 2008.
[35] Multi-Resolution Land Cover Characteristics (MRLC) Consortium. https://www.mrlc.gov/data.
[36] NASA EarthData. https://earthdata.nasa.gov/earth-observation-data.
[37] T. Nguyen, I. Hettiarachchi, A. Khatami, L. Gordon-Brown, C. P. Lim, and S. Nahavandi. Classification of Multi-Class BCI Data by Common Spatial Pattern and Fuzzy System. *IEEE Access*, 6:27873–27884, 2018.
[38] J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems, TODS*, 9(1):38–71, 1984.
[39] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2011.
[40] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *VLDB Journal*, 10(11):1190–1201, August 2017.
[41] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 2005.
[42] Ibrahim Sabek, Mashaal Musleh, and Mohamed Mokbel. TurboReg: A Framework for Scaling Up Spatial Logistic Regression Models. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, Seattle, WA, November 2018.
[43] Nikita A. Sakhanenko and David J. Galas. Markov Logic Networks in the Analysis of Genetic Data. *Journal of Computational Biology*, 17(11):1491–1508, 2010.
[44] R. Sanderson, M. D. Eyre, S. P. Rushton, and Kaj Sand-Jensen. Distribution of Selected Macroinvertebrates in a Mosaic of Temporary and Permanent Freshwater Ponds as Explained by Autologistic Models. *Ecography*, 28(3):355–362, 2005.
[45] J. Michael Scott, Patricia J. Heglund, and Michael L. Morrison et al. Predicting Species Occurrences: Issues of Accuracy and Scale. *Journal of Mammalogy*, 2002.
[46] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2007.
[47] Michael Sherman, Tatiyana V. Apanasovich, and Raymond J. Carroll. On Estimation in Binary Autologistic Spatial Models. *Journal of Statistical Computation and Simulation*, 76(2):167–179, 2006.
[48] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental Knowledge Base Construction Using DeepDive. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2015.
[49] Parag Singla and Pedro Domingos. Discriminative Training of Markov Logic Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, pages 868–873, 2005.
[50] Brian L. Sullivan, Christopher L. Wood, Marshall J. Iliff, Rick E. Bonney, Daniel Fink, and Steve Kelling. eBird: A Citizen-based Bird Observation Network in the Biological Sciences. *Biological Conservation*, 142(10):2282–2292, 2009.
[51] K. Tangthaikwan, N. Keeratipranon, and A. Agsornintara. Multiclass Support Vector Machine for Classification Spatial Data from Satellite Image. In *Proceedings of the International Conference on Knowledge and Smart Technology, KST*, pages 111–115, 2017.

[52] Amin Tayyebi, Mahmoud Reza Delavar, Mohammad Javad Yazdanpanah, Bryan Christopher Pijanowski, Sara Saeedi, and Amir Hossein Tayyebi. A Spatial Logistic Regression Model for Simulating Land Use Patterns: A Case Study of the Shiraz Metropolitan Area of Iran. In *Advances in Earth Observation of Global Change*, 2010.

[53] W. R. Tobler. *Cellular Geography: Philosophy in Geography*. Springer, 1979.

[54] USGS National Land Cover Dataset (NLCD). https://catalog.data.gov/dataset/usgs-national-land-cover-dataset-nlcd-downloadable-data-collection.

[55] USGS National Transportation Dataset (NTD). https://catalog.data.gov/dataset/usgs-national-transportation-dataset-ntd-downloadable-data-collectionde7d2.

[56] Zilong Wang and Yanbing Zheng. Analysis of Binary Data via a Centered Spatial-temporal Autologistic Regression Model. *Journal of Environmental and Ecological Statistics*, 20(1):37–57, 2013.

[57] Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable Probabilistic Databases with Factor Graphs and MCMC. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2010.

[58] Mark A. Wolters. Better Autologistic Regression. *Frontiers in Applied Mathematics and Statistics*, 3:24, 2017.

[59] Mark A. Wolters and C. B. Dean. Classification of Large-Scale Remote Sensing Images for Automatic Identification of Health Hazards: Smoke Detection Using an Autologistic Regression Classifier. *Statistics in Biosciences*, 9(2):622–645, 2017.

[60] Ce Zhang and Christopher Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2013.

[61] Yanbing Zheng and Jun Zhu. Markov Chain Monte Carlo for a Spatial-Temporal Autologistic Regression Model. *Journal of Computational and Graphical Statistics*, 17(1):123–137, 2008.

[62] Jun Zhu, Hsin-Cheng Huang, and Jungpin Wu. Modeling Spatial-temporal Binary Data using Markov Random Fields. *Journal of Agricultural, Biological, and Environmental Statistics*, 10(2):212, 2005.

[63] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized Stochastic Gradient Descent. In *Proceedings of the Annual Conference on Neural Information Processing Systems, NIPS*, 2010.

A THEORETICAL FOUNDATION OF *REGROCKET* USING MLN

Theorem 1. *Given an autologistic model defined over n locations* l_1 $l_⊠$ *with*

n multinomial prediction variables z_1 $z_⊠$ *, each has r possible outcomes* ⊠ λ_1 $\lambda_⊠$ *, and are represented with a set of nr binary variables* $z_1(\lambda_1)$ $z_⊠(\lambda_⊠)$ *,*

m weighted multinomial predictor variables (i) $x_1(i)$ $x_⊠(i)$ *at each location i, each has q possible domain values* ⊠ t_1 tq *, and are represented with a set of mq(r* $1)$ *weighted binary variables* $\beta_1^{\lambda_1\ 1} x_1^{\lambda_1\ 1}(i)$ $\beta_⊠^{\lambda_⊠\ 1\ ⊠} x_⊠^{\lambda_⊠\ 1\ ⊠}(i)$ *, where* $\lambda_⊠$ *is a pivot outcome of any multinomial prediction* $z_⊠$*, and*

$r(r$ $1)$ *neighbouring weights* η $\eta_{\lambda_1\ \lambda_1}$ $\eta_{\lambda_⊠\ 1\ \lambda_⊠}$

there is an equivalent Markov Logic Network (MLN) to this autologistic model, if and only if:

each predictor-based regression term $\beta_⊠^{\lambda\ ⊠} x_⊠^{\lambda\ ⊠}(i)$ *at location* $l_⊠$ *has an equivalent bitwise-AND predicate* $z_⊠(\lambda)$ $x_⊠^{\lambda\ ⊠}(i)$ *with weight* $\beta_⊠^{\lambda\ ⊠}$*, where* λ *is not a pivot outcome (i.e.,* $\lambda \neq \lambda_⊠$*).*

each neighbour-based regression term $\eta_{\lambda\ ⊠} z_⊠(s)$ *at location* $l_⊠$ *has an equivalent bitwise-AND predicate* $z_⊠(\lambda)$ $z_⊠(s)$ *with weight* $\eta_{\lambda\ ⊠}$*, where* λ *is not a pivot outcome (i.e.,* $\lambda \neq \lambda_⊠$*).*

each prediction variable $z_⊠(\lambda_⊠)$ *at location* $l_⊠$ *is associated with a constant predicate of value 1 and weight 0.*

Proof. Based on the conditional probabilities of multinomial autologistic model in Equation 2, the probability of predicting any possible non-pivot outcome λ (i.e., $\lambda \neq \lambda_⊠$) at location $l_⊠$ given the predictor variables (i) at $l_⊠$ and the neighbouring prediction variables $_⊠$ can be estimated as:

$$Pr(z_⊠(\lambda) \quad 1 \quad \phi_⊠) \quad Pr(z_⊠(\lambda_⊠) \quad 1 \quad \phi_⊠) \exp \sum_{⊠\ 1\ ⊠}^{⊠} \sum_{⊠_⊠(⊠)} \beta_⊠^{\lambda\ ⊠} x_⊠^{\lambda\ ⊠}(i) \sum_{⊠ \quad _⊠ ⊠} \sum_{⊠_⊠} \eta_{\lambda\ ⊠} z_⊠(s) \tag{6}$$

where $\phi_{\boxtimes}$ (i) $_{\boxtimes}$. As a result, the probability of predicting the pivot outcome $\lambda_{\boxtimes}$ at location $l_{\boxtimes}$ can be calculated as follows:

$$
\begin{aligned}
Pr(z_{\boxtimes}(\lambda_{\boxtimes}) \quad 1 \quad \phi_{\boxtimes}) \quad & 1 \sum_{\lambda \quad \boxtimes_{\boxtimes}\ \lambda \neq \lambda_{\boxtimes}} Pr(z_{\boxtimes}(\lambda) \quad 1 \quad \phi_{\boxtimes}) \\
& 1 \sum_{\lambda \quad \boxtimes_{\boxtimes}\ \lambda \neq \lambda_{\boxtimes}} Pr(z_{\boxtimes}(\lambda_{\boxtimes}) \quad 1 \quad \phi_{\boxtimes}) \exp \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda\ \boxtimes} x_{\boxtimes}^{\lambda\ \boxtimes}(i) \sum_{\boxtimes} \sum_{\boxtimes^{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda\ \boxtimes} z_{\boxtimes}(s) \\
& \frac{1}{1 \sum_{\lambda \quad \boxtimes_{\boxtimes}\ \lambda \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda\ \boxtimes} x_{\boxtimes}^{\lambda\ \boxtimes}(i) \sum_{\boxtimes} \sum_{\boxtimes^{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda\ \boxtimes} z_{\boxtimes}(s)}
\end{aligned} \tag{7}
$$

From equations 6 and 7, the probability distribution of any outcome prediction $z_{\boxtimes}(\lambda)$ is:

$$
Pr(z_{\boxtimes}(\lambda) \quad 1 \quad \phi_{\boxtimes}) \quad
\begin{cases}
\dfrac{\exp \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda,\boxtimes} x_{\boxtimes}^{\lambda,\boxtimes}(\boxtimes) \sum_{\boxtimes} \sum_{\boxtimes^{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)}{1 \sum_{\boxtimes \quad \boxtimes_{\boxtimes},\boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes,\boxtimes} x_{\boxtimes}^{\boxtimes,\boxtimes}(\boxtimes) \sum_{\boxtimes} \sum_{\boxtimes^{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)} & \lambda \neq \lambda_{\boxtimes} \\
\dfrac{1}{1 \sum_{\boxtimes \quad \boxtimes_{\boxtimes},\boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes,\boxtimes} x_{\boxtimes}^{\boxtimes,\boxtimes}(\boxtimes) \sum_{\boxtimes} \sum_{\boxtimes^{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)} & \lambda \quad \lambda_{\boxtimes}
\end{cases} \tag{8}
$$

Now, assume a model that consists of $nr \quad nmq(r \quad 1)$ binary random variables $z_1(\lambda_1) \quad z_{\boxtimes}(\lambda_{\boxtimes})\ x_1^{\lambda_1\ 1}(i) \quad x_{\boxtimes}^{\lambda_{\boxtimes}\ 1\ \boxtimes}(i)$, coming from nr prediction and $nmq(r \quad 1)$ predictor variables over all locations . In addition, in case $\lambda \neq \lambda_{\boxtimes}$, assume a set of constraints $F_1 \quad F_{\boxtimes}$ are defined over variables , where constraints $F_{\boxtimes}$ at location $l_{\boxtimes}$ consist of two subsets of constraints. The first subset consists of $mq(r \quad 1)$ bitwise-AND predicates $z_{\boxtimes}(\lambda) \quad x_{\boxtimes}^{\lambda\ \boxtimes}(i)$ with β weights (each predicate corresponds to a predictor-based regression term). The second subset consists of $r(r \quad 1)s_{\boxtimes}$ bitwise-AND predicates $z_{\boxtimes}(\lambda) \quad z_{\boxtimes}(s)$ with η weights (each corresponds to a neighbour-based regression term) where $s_{\boxtimes}$ is the size of neighbouring locations $_{\boxtimes}$ of location $l_{\boxtimes}$. Finally, in case $\lambda \quad \lambda_{\boxtimes}$ at each location $l_{\boxtimes}$, assume one constant predicate of value 1 with weight 0. Based on these assumptions, the model satisfies the two main properties in Section 2.2.1 that are needed to represent it using MLN, and hence its joint probability distribution over is estimated based on Equation 3 as follows:

$$
Pr(\quad) \quad
\begin{cases}
\frac{1}{\boxtimes} \exp \sum_{\boxtimes\ 1\ \boxtimes\ 1\ (\lambda\ \boxtimes)}^{\boxtimes \quad \boxtimes} \sum_{\boxtimes_{\boxtimes}} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda\ \boxtimes} f_{\lambda\ \boxtimes}(z_{\boxtimes}(\lambda) \quad x_{\boxtimes}^{\lambda\ \boxtimes}(i)) \\
\quad \sum_{\boxtimes\ 1\ \boxtimes}^{\boxtimes} \sum_{\boxtimes(\lambda\ \boxtimes)} \sum_{\boxtimes_{\boxtimes}} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda\ \boxtimes} f_{\lambda\ \boxtimes}(z_{\boxtimes}(\lambda) \quad z_{\boxtimes}(s)) & \lambda \neq \lambda_{\boxtimes} \\
\frac{1}{\boxtimes} & \lambda \quad \lambda_{\boxtimes}
\end{cases} \tag{9}
$$

where $f_{\lambda\ \boxtimes}(\ )$ and $f_{\lambda\ \boxtimes}(\ )$ represent functions to evaluate the bitwise-AND predicates $z_{\boxtimes}(\lambda) \quad x_{\boxtimes}^{\lambda\ \boxtimes}(i)$ and $z_{\boxtimes}(\lambda) \quad z_{\boxtimes}(s)$, respectively, and return either 1 or 0 as output. Note that in the case of pivot outcome $\lambda_{\boxtimes}$, we have a constant predicate of value 1 with weight 0, and hence its probability becomes $\frac{1}{\boxtimes} \exp(0(1)) \quad \frac{1}{\boxtimes}$, where C is the normalization constant of the model.

Based on Equation 9, the conditional probability distribution of any prediction variable $z_{\boxtimes}(\lambda)$ at location $l_{\boxtimes}$ given the predictor variables (i) at $l_{\boxtimes}$ and its neighbouring prediction variables $_{\boxtimes}$ (i.e., $\phi_{\boxtimes}$ (i) $_{\boxtimes}$) can be estimated as:

$$Pr(z_{\boxtimes}(\lambda) \quad 1 \quad \phi_{\boxtimes}) \quad \begin{cases} \frac{1}{\boxtimes} \exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda \boxtimes} f_{\lambda \boxtimes}(1 \;\; x_{\boxtimes}^{\lambda \boxtimes}(i)) \\ \qquad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda \boxtimes} f_{\lambda \boxtimes}(1 \;\; z_{\boxtimes}(s)) & \lambda \neq \lambda_{\boxtimes} \\ \frac{1}{\boxtimes} & \lambda \quad \lambda_{\boxtimes} \end{cases} \tag{10}$$

where the normalization constant C is calculated over all possible outcomes of $z_{\boxtimes}$, to ensure the probability value of any outcome $[0 \;\; 1]$, as follows:

$$C \quad \sum_{\boxtimes} \sum_{\boxtimes_{\boxtimes} \boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes \boxtimes} f_{\boxtimes \boxtimes}(1 \;\; x_{\boxtimes}^{\boxtimes \boxtimes}(i)) \quad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes \boxtimes} f_{\boxtimes \boxtimes}(1 \;\; z_{\boxtimes}(s)) \quad \exp(0)$$

$$1 \quad \sum_{\boxtimes} \sum_{\boxtimes_{\boxtimes} \boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes \boxtimes} f_{\boxtimes \boxtimes}(1 \;\; x_{\boxtimes}^{\boxtimes \boxtimes}(i)) \quad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes \boxtimes} f_{\boxtimes \boxtimes}(1 \;\; z_{\boxtimes}(s))$$

Since all variables are binary, the evaluation of $f_{\lambda \boxtimes}$ and $f_{\lambda \boxtimes}$ can be represented as a mathematical multiplication (i.e., the value of $f_{\lambda \boxtimes}(1 \;\; x_{\boxtimes}^{\lambda \boxtimes}(i))$ is $x_{\boxtimes}^{\lambda \boxtimes}(i)$ and the value of $f_{\lambda \boxtimes}(1 \;\; z_{\boxtimes}(s))$ is $z_{\boxtimes}(s)$). As a result, the joint probability distribution of $z_{\boxtimes}(\lambda)$ from Equation 10 becomes:

$$Pr(z_{\boxtimes}(\lambda) \quad 1 \quad \phi_{\boxtimes}) \quad \begin{cases} \dfrac{\exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\lambda,\boxtimes} \boxtimes_{\boxtimes}^{\lambda,\boxtimes}(\boxtimes) \quad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\lambda,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)}{1 \quad \sum_{\boxtimes} \sum_{\boxtimes_{\boxtimes},\boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}^{\boxtimes,\boxtimes}(\boxtimes) \quad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)} & \lambda \neq \lambda_{\boxtimes} \\ \dfrac{1}{1 \quad \sum_{\boxtimes} \sum_{\boxtimes_{\boxtimes},\boxtimes \neq \lambda_{\boxtimes}} \exp \sum_{\boxtimes 1 \boxtimes}^{\boxtimes} \sum_{\boxtimes_{\boxtimes}(\boxtimes)} \beta_{\boxtimes}^{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}^{\boxtimes,\boxtimes}(\boxtimes) \quad \sum_{\boxtimes} \sum_{\boxtimes \boxtimes} \sum_{\boxtimes_{\boxtimes}} \eta_{\boxtimes,\boxtimes} \boxtimes_{\boxtimes}(\boxtimes)} & \lambda \quad \lambda_{\boxtimes} \end{cases} \tag{11}$$

which is the same probability distribution of the autologistic model defined in Equation 8. This means that the assumed model at the beginning, which can be represented with MLN, is equivalent to the multinomial autologistic model.

Received November 2018; revised XXX 2019; accepted XXX 2019