TurboReg: A Framework for Scaling Up Spatial Logistic Regression Models

Ibrahim Sabek
University of Minnesota, USA
sabek@cs.umn.edu

Mashaal Musleh
University of Minnesota, USA
musle005@cs.umn.edu

Mohamed F. Mokbel*
Qatar Computing Research Institute
mmokbel@hbku.edu.qa

ABSTRACT

Predicting the presence or absence of spatial phenomena has been of great interest to scientists pursuing research in several applications including epidemic diseases detection, species occurrence prediction and earth observation. In this operation, a geographical space is divided by a two-dimensional grid, where the prediction (i.e, either 0 or 1) is performed at each cell in the grid. A common approach to solve this problem is to build spatial logistic regression models (a.k.a autologistic models) that estimate the prediction at any location based on a set of predictors (i.e., features) at this location and predictions from neighboring locations. Unfortunately, existing methods to build autologistic models are computationally expensive and do not scale up for large-scale grid data (e.g., fine-grained satellite images). This paper introduces *TurboReg*, a scalable framework to build autologistic models for predicting large-scale spatial phenomena. *TurboReg* considers both the accuracy and efficiency aspects when learning the regression model parameters. *TurboReg* 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. A set of experiments using large real and synthetic data show that *TurboReg* achieves at least three orders of magnitude performance gain over existing methods while preserving the model accuracy.

CCS CONCEPTS

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

KEYWORDS

Spatial Regression, Autologistic Models, Markov Logic Networks, First-order Logic, Factor Graph

1 INTRODUCTION

Predicting the presence or absence of spatial phenomena in a certain geographical area is a crucial task in many scientific domains (e.g., Earth observations [25, 43], Epidemiology [13, 27, 37], Ecology [4, 35], Agriculture [18, 22] and Management [29]). For example, ornithologists would need to predict the presence or absence of a certain bird species across a certain area [39]. Meteorologists would need to predict the hurricane or tornado boundaries. Epidemiologist would need to understand the spread of diseases across various areas in the world. Typically, this is done by dividing the geographical space (e.g., the whole world) by a two-dimensional grid, where each grid cell is represented with a binary variable (i.e., takes either 0 or 1) indicating the presence or absence of the spatial phenomena in that grid cell, and a set of predictor variables (i.e., features) that help predicting the value of this binary 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, predict the value of the binary variable at this cell.

A common approach to solve the prediction problem is to build a standard logistic regression model [5, 10] that uses a logistic function to predict the value of each grid cell based on the values of predictors in the same grid cell. However, standard logistic regression models are deemed inappropriate for predicting spatial phenomena as they assume that neighboring locations are completely independent of each other. This is definitely not the case for spatial phenomena as neighboring locations tend to systematically affect each other [40].

As a result, spatial variants of logistic regression models (a.k.a., autologistic regression) were proposed to take into account the spatial dependence between neighboring grid cells [6, 8, 24]. However, existing methods for autologistic regression (e.g., see [7, 24, 28, 41]) are prohibitively computationally expensive for large grid data, e.g., fine-grained satellite images [2, 43], and large spatial epidemiology datasets [26]. For example, it could take about week to infer the model parameters using training data of only few gigabytes [24]. 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 neighboring cells. For example, if a prediction cell variable C_1 depends on two neighboring cells C_2 and C_3, then current 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 *TurboReg*; a scalable framework for using autologistic models in predicting large-scale spatial phenomena. *TurboReg* does not need to sample training data sets. It can support prediction over grids of 85000 cells in 10 seconds. Moreover, *TurboReg* allows its users to define high degrees of dependency relations among neighbors, 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 neighboring cells C_2 and C_3, *TurboReg* 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.

* Also affiliated with University of Minnesota, MN, USA.

This work is partially supported by the National Science Foundation, USA, under Grants IIS-1525953, and CNS-1512877.

TurboReg exploits Markov Logic Networks (MLN) [12] (a scalable statistical learning framework) to learn the autologistic regression parameters in an accurate and efficient manner. Then, *TurboReg* aims to provide an equivalent first-order logic [15] representation to dependency relations among neighbors in autologistic models. This is necessary to accurately express the autologistic models using MLN. Since we focus on binary prediction variables, *TurboReg* transforms each neighboring dependency relation into a predicate with bitwise-AND operation on all variables involved in this relation. For example, a ternary dependency relation between neighboring variables C_1, C_2 and C_3 is transformed to $C_1 \wedge C_2 \wedge C_3$. 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.

TurboReg 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 [42], along with an efficient weights optimization technique based on gradient descent optimization [47]. *TurboReg* represents the MLN bitwise-AND predicates using the spatially-indexed factor graph. Then, *TurboReg* 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, *TurboReg* converges to the optimal model parameters faster than existing computational methods by at least three orders of magnitude.

We experimentally evaluate *TurboReg* using a real dataset of the daily distribution of bird species [39], and a synthetic dataset about the crime types in Minneapolis, MN area. For each dataset, we compare the accuracy and scalability of the built autologistic models using *TurboReg* and a state-of-the-art open-source autologistic model computational method, namely ngspatial [23]. Our experiments show that *TurboReg* is scalable to large-scale autologistic models compared to existing techniques, while preserving 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 and MLN framework. Section 3 describes how autologistic regression is modeled using MLN. Section 4 gives an overview of *TurboReg*. Section 5 describes how the first-order logic predicates are generated for 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 experimental analysis of *TurboReg*. Section 9 covers related work, while Section 10 concludes the paper.

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

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 predictors (i.e., features that help in the prediction process) at the same location and a set of observed predictions from variables at neighbouring

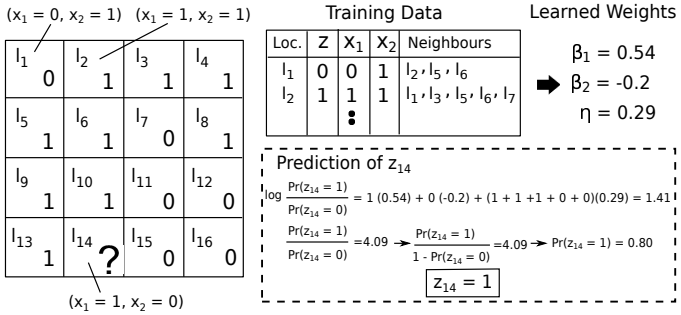


Figure 1: An Example on Autologistic Regression.

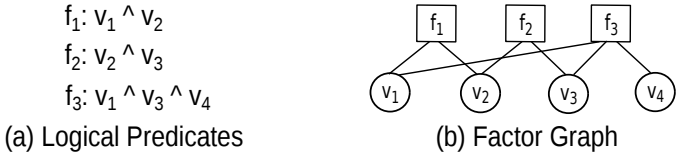


Figure 2: Translating First-order Logic Predicates into A Factor Graph in MLN.

locations (i.e., spatial dependence). Formally, 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 [6, 24]:

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

where the weights $\beta = \{\beta_1, ..., \beta_m\}$ and η form the model parameters $\theta = \{\beta, \eta\}$. The objective of our work is to learn the values of θ from previous observations (i.e., training data) of predictions and predictors at locations $\mathcal{L}$, in a scalable and efficient manner. Note that the values of θ are shared among the whole locations $\mathcal{L}$.

Assumptions. In general, predictor variables $\mathcal{X}$ can be either binary, categorical, or continuous. However, we focus only on binary predictors (i.e., $x_m(i) \in \{0, 1\}$). The extension to categorical and continuous cases is intuitive as well, but, out of scope of this paper.

Example. Figure 1 shows a numerical example of 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 and two predictor variables $x_1(i)$ and $x_2(i)$. The model parameters β_1 and β_2 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.

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 [9, 12, 31]. MLN combines probabilistic graphical models (e.g., factor graphs [42]) with first-order logic [15] to perform probabilistic learning based on logic constraints, where logic handles model complexities and probability handles uncertainty. MLN has been successfully applied in a wide span of data intensive applications including knowledge bases construction [38], machine learning models [11], and genetic analysis [34]. 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 $\mathcal{V} = \{v_1, ..., v_p\}$ ($v_i \in \{0, 1\}$). (2) the dependencies between model variables $\mathcal{V}$ can be described with a set of weighted constraints $\mathcal{F} = \{f_1, ..., f_h\}$ defined over them, where these weights $\mathcal{W} = \{w_1, ..., w_h\}$ are the model parameters that need to be learned. The constraints describe how the values of variables $\mathcal{V}$ correlate with each other. A model with these two properties can exploit MLN to learn weights $\mathcal{W}$ that maximize the probability of satisfying model constraints $\mathcal{F}$.

Example. Assume a model of two variables v_{prof} and v_{teach}, where v_{prof} denotes whether a person is professor or not, and v_{teach} 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_{prof} and v_{teach} having the same value (i.e., either $v_{prof} = 1$ and $v_{teach} = 1$ or $v_{prof} = 0$ and $v_{teach} = 0$).

2.2.2 First-order Logic. MLN employs first-order logic predicates [15] (e.g., conjunction, disjunction and implication) to represent the model constraints. For example, the constraint defined over v_{prof} and v_{teach} can be represented as a bitwise-AND predicate $v_{prof} \wedge v_{teach}$. Efficient logic programming frameworks were proposed to generate first-order logic predicates on a large-scale such as DDlog [38].

2.2.3 Factor Graph. To learn the values of weights $\mathcal{W}$ associated with predicates (i.e., constraints), MLN translates these predicates into an equivalent probabilistic graphical model, namely factor graph [42], which has $\mathcal{W}$ as the parameters of its joint probability distribution. By doing that, the problem of learning $\mathcal{W}$ is reduced into the problem of learning the joint distribution of this factor graph. A factor graph $\mathcal{G}$ is a bipartite graph that represents each model variable $v \in \mathcal{V}$ and constraint $f \in \mathcal{F}$ 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 $\mathcal{V}$ in a factor graph $\mathcal{G}$ can be estimated in terms of the constraints (i.e., predicates) $\mathcal{F}$ and their weights $\mathcal{W}$ as a log-linear model [12]:

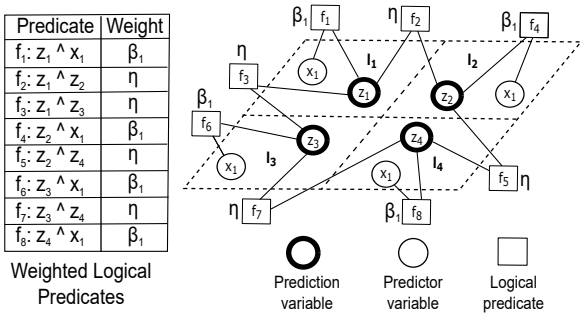


Figure 3: MLN Representation of An Autologistic Model (logical predicates and their factor graph).

$$Pr(\mathcal{V} = v) = \frac{1}{C} \exp\Big(\sum_{i=1}^{h} w_i f_i(v) \Big) \tag{2}$$

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 learning the values of weights $\mathcal{W}$ in factor graph, such as gradient descent optimization [44, 47].

3 AUTOLOGISTIC REGRESSION VIA MARKOV LOGIC NETWORK

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

3.1 MLN-based Autologistic Model

To represent an autologistic model using MLN, the model should have the two main properties mentioned in Section 2.2.1. Obviously, the first property is satisfied because all prediction and predictor variables (i.e., $\mathcal{Z}$ and $\mathcal{X}$) are already binary. However, to achieve the second property, the model is required to have a set of equivalent constraints (i.e., logical predicates) that capture the autologistic regression semantics. As shown in Equation 1, there are two types of regression terms: (1) *predictor-based* terms $\{\beta_j x_j(i) \mid j = 1, .., m\}$ where m is the number of predictors at any location l_i, and (2) *neighbour-based* terms $\{\eta z_k \mid k \in \mathcal{N}_i\}$ where $\mathcal{N}_i$ is the set of neighbours at location l_i. *TurboReg* provides an equivalent weighted first-order logic predicate to each regression term, either predictor-based or neighbour-based, that preserves the semantic of autologistic regression and can be represented with MLN as well. For each prediction variable z_i at location l_i, each *predictor-based* regression term $\beta_j x_j(i)$ has an equivalent bitwise-AND predicate defined over z_i and $x_j(i)$ (i.e., $z_i \wedge x_j(i)$) with weight β_j. Similarly, each *neighbour-based* regression term ηz_k has an equivalent bitwise-AND predicate defined over z_i and z_k (i.e., $z_i \wedge z_k$) with weight η. The theoretical foundation of the proposed MLN-based autologistic model is described in the Appendix A. Note that, using the proposed model, the autologistic regression parameters $\theta = \{\beta, \eta\}$ are

translated into a set of weights $\mathcal{W}$ 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 $\mathcal{W}$ in MLN (See section 2.2.3).

Example. Figure 3 shows an example of translating an autologistic regression model with one predictor variable x_1 (i.e., $\log \frac{Pr(z_i=1|\mathcal{X},\mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i=0|\mathcal{X},\mathcal{Z}_{\mathcal{N}_i})} = \beta_1 x_1(i) + \eta \sum_{k\in\mathcal{N}_i} z_k$) into an equivalent MLN. The model is built for a 4-cells grid, where the neighbourhood $\mathcal{N}_i$ of any cell l_i is assumed to be cells that share edges with l_i (i.e., first-order neighbourhood). The model is first translated into a set of 8 bitwise-AND predicates with two weights β_1 and η. 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 and z_2 have two equivalent $z_1 \wedge z_2$ and $z_2 \wedge z_1$ neighbour-based predicates, respectively, however, we keep only one of them).

3.2 Generalized Autologistic Models

Some applications assume models with more generalized neighbour-based regression terms $\{\eta G(z_{k_1}, ..., z_{k_d}) \mid k_1, ..., k_d \in \mathcal{N}_i\}$ (i.e., complex spatial dependence), where the regression term has a function defined over neighbouring prediction variables $G(z_{k_1}, ..., z_{k_d})$, and not just their sum as in Equation 1 (e.g., Ecology [36] and Mineral Exploration [20]). Existing methods can not compute autologistic models with generalized regression terms because of their prohibitively expensive computations, such as high-order matrix multiplications [24]. In contrast, the MLN-based autologistic model can be easily extended to find an equivalent combination of first-order logic predicates [15] for any generalized regression term, as long as the function $G(z_{k_1}, ..., z_{k_d})$ holds logical semantics. For example, if prediction variable z_1 at location l_1 has a generalized regression function $G(z_2, z_3)$ over neighbours z_2 and z_3 which constraints the value of z_1 to be 1 only if both values of z_2 and z_3 are 1 at the same time, then *TurboReg* would translate this into an equivalent bitwise-AND predicate $z_1 \wedge z_2 \wedge z_3$. As another example, if $G(z_2, z_3)$ constraints the value of z_1 to be 1 only if the either z_2 or z_3 is 1, then it can be translated into a predicate $z_1 \wedge (z_2 \vee z_3)$ that has a combination of bitwise-AND and bitwise-OR. The proof of this extension is pretty similar to the proof in Appendix A, however, it is removed due to space constraints. 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 TURBOREG

Figure 4 depicts the system architecture of *TurboReg*. 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 *TurboReg* user and generates a set of bitwise-AND predicates of an equivalent MLN. It employs an efficient logic programming framework, called DDlog [38], 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 bitwise-AND 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.

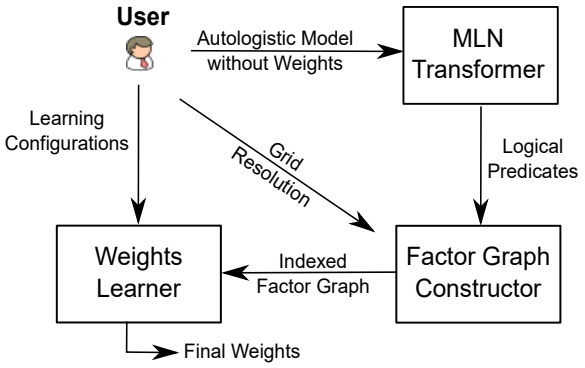


Figure 4: *TurboReg* System Architecture.

Weights Learner. This is the main module in *TurboReg* 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, *TurboReg* provides a scalable variation of gradient descent [47] technique, that is highly optimized for learning the autologistic model parameters. Details are in Section 7.

5 MLN TRANSFORMER

The first step in *TurboReg* is to generate a set of equivalent logical predicates for the different regression terms in the autologistic model. However, this step is challenging if : (1) the model has a large number of prediction variables $\mathcal{Z}$ (i.e., large 2-dimensional grid) and/or predictor variables $\mathcal{X}$ (e.g., large number of synthetic features), which results in generating a large number of neighbour-based and predictor-based predicates at the end. (2) the model has very complicated generalized regression terms, which are translated into predicates with large number of combinations of first-order logic symbols (e.g., bitwise-AND, bitwise-OR, and imply).

To remedy this challenge, *TurboReg* uses DDlog [38], an 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 large number or 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 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 [38] and data cleaning [32]).

Example. Figure 5 shows an example of using DDlog to express the bitwise-AND predicates of the 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 variables $\mathcal{Z}$ and predictor variables $\mathcal{X}$ are stored in relations *z*? and *x*?, respectively,

```
#Schema Declaration
z?(@key id bigint, value numeric).
x?(@key id bigint, value numeric).
neighbor(id1 bigint, id2 bigint).

#Derivation Rules
z(id) ^ x(id):- z(id).
z(id1) ^ z(id2) :- neighbor(id1, id2)
```

Figure 5: Example of Using DDlog to Generate Bitwise-AND Predicates for Autologistic Model.

where any row in each relation corresponds to one variable and stores its location ID and its to-be-predicted value in attributes *id* 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_1, f_4, f_6 and f_8 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 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 and f_7 in Figure 3), where a predicate is defined for each individual pair of neighbouring predication variables.

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 [19]) that partition the data over the index and *space-partitioning* index structures (e.g., Quadtrees [14]) that partition the space. In *TurboReg* we decided to go with the Grid Index [30] 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, *TurboReg* 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_3, and C_2 holds predicates in locations l_2 and l_4. *TurboReg* 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

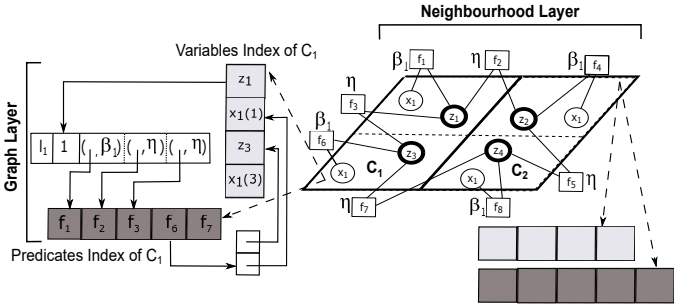


Figure 6: Example of Spatially-indexed Factor Graph.

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 appear 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 in the variables index.

Predicates Index. This index contains all predicates that 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_6 in the predicates index. Note that we replicate predicates that have variables in two different grid cells (e.g., f_7 is duplicated in C_1 and C_2 because it has variable $z_3 \in C_1$ and $z_4 \in C_2$).

7 WEIGHTS LEARNER

This is the most important module in *TurboReg*, which takes the spatially-indexed factor graph along with learning configurations from user and returns the final weights $\theta = \{\beta, \eta\}$ of the autologistic model. The main idea is to incrementally converge to weights that maximize the satisfaction of bitwise-AND predicates represented in the factor graph. For example, if the predicate $z_i \wedge x_i$ is satisfied, then the current value of its weight η should be rewarded (i.e., should be increased), otherwise should be punished (i.e., should be decreased). To that end, we adapt a technique that punishs and rewards weights using gradient descent optimization [47]. To test satisfaction of any bitwise-AND predicate in autologistic regression, we suggest to substitute in Equation 1 with the values of variables that appear in the predicate along with its weight. 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 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 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

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

1: $\beta \leftarrow$ Random, $\eta \leftarrow$ Random
2: $e \leftarrow \frac{E}{S}$ /* Num. of Learning Epochs Per Instance*/
3: **while** $e \neq 0$ **do**
4: **for all** $s \in \{1, 2, ..., S\}$ **do in parallel**
5: **for all** $c \in C$ **do in parallel**
6: $\mathcal{V}_c \leftarrow$ Variables index in cell c
7: $\mathcal{P}_c \leftarrow$ Predicates index in cell c
8: **for each** $v_i \in \mathcal{V}_c$ **do**
9: UpdateWeights ($v_i, \mathcal{V}_c, \mathcal{P}_c, \alpha$) (Algorithm 2)
10: **end for**
11: $\beta \leftarrow \frac{\sum_{s=1}^{S} \beta_s}{S}$, $\eta \leftarrow \frac{\sum_{s=1}^{S} \eta_s}{S}$
12: $e - -$
13: **end while**
14: **return** β and η

in parallel (i.e., S learning epochs are running simultaneously). In such learning epoch, we learn an optimal instance of weights β_s and η_s, where these values are incrementally learned from variables in 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 gradient descent optimization [47] technique to incrementally update the values of weights given a certain variable v_i (either prediction or predictor). The main idea is to punish or reward current weights based on their performance in correctly estimating the prediction value z_i at location l_i where variable v_i belongs to. The algorithm takes the following inputs; a variable v_i, the variables $\mathcal{V}$ and predicates $\mathcal{P}$ indexes in the grid cell containing v_i (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 weights through 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 $\hat{z_i}$ at location l_i that contains v_i using Equation 1. If the estimation $\hat{z_i}$ never matches the observed prediction value z_i from training data, then we set the status g to -1 (i.e., the associated weight with current variable v_i needs to be punished), otherwise the status remains rewarding. In case v_i is a predictor variable $x_j(i)$, we only update its associated weight β_j by evaluating the gradient descent equation using current values of g and α (Line 7 in Algorithm 2) and jump to the end of algorithm. In case v_i is the prediction variable z_i itself, we apply gradient descent optimization on all weights β associated with its predictors (Lines 9 to 11 in Algorithm 2), and on weight η associated with neighbouring predicates (Lines 14 to 24 in Algorithm 2) as well.

Algorithm 2 Function UpdateWeights (Variable v_i, VariablesIndex $\mathcal{V}$, PredicatesIndex $\mathcal{P}$, StepSize α)

1: $l_i \leftarrow \mathcal{V}[v_i]$.location, $g \leftarrow 1$ /* Gradient Value */
2: $\hat{z_i} \leftarrow$ Prediction at l_i using β *and* η (Equation 1)
3: **if** $\mathcal{V}[v_i]$.value $\neq \hat{z_i}$ **then**
4: $g \leftarrow$ -1
5: **end if**
6: **if** v_i is any predictor variable $x_j(i) \in \mathcal{X}(i)$ **then**
7: $\beta_j \leftarrow \beta_j + \alpha\ g$ /* Gradient Descent on β_j*/
8: **else**
9: **for each** $\beta_j \in \beta$ **do**
10: $\beta_j \leftarrow \beta_j + \alpha\ g$ /* Gradient Descent on β_j*/
11: **end for**
12: **end if**
13: **if** v_i is prediction variable z_i **then**
14: **for each** $p \in \mathcal{P}[v_i]$ **do**
15: **if** p is a neighbour-based predicate **then**
16: $\hat{z_k} \leftarrow$ Prediction at neighbour l_k in p using β and η
17: **if** $\mathcal{V}[v_k]$.value $\neq \hat{z_k}$ **then**
18: $g \leftarrow$ -1
19: **else**
20: $g \leftarrow 1$
21: **end if**
22: $\eta \leftarrow \eta + \alpha\ g$ /* Gradient Descent on η */
23: **end if**
24: **end for**
25: **end if**

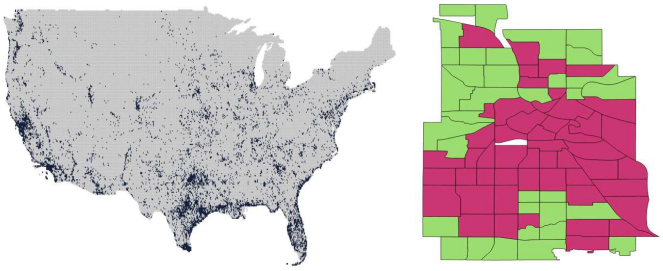
Complexity. The complexity of the aforementioned algorithms can be estimated as $O(\frac{E}{S}\frac{(n^2+nm)}{C})$ where n is number of predictions, m is number of predictors, C is number of factor graph partitions, E is number of learning epochs and S is number of learning instances. This complexity can be further approximated to be $O(\frac{E}{S}\frac{(n^2)}{C})$. 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 *TurboReg* in building autologistic models (i.e., learning their weights). We compare the performance of *TurboReg* with ngspatial [23], a state-of-the-art open-source package for autologistic model implementations. Specifically, we compare our performance with the most accurate algorithm in ngspatial that employs bayesian inference using Markov Chain Monte Carlo (MCMC) [24]. We extensively investigate the accuracy and scalability of both systems under different grid sizes (Section 8.2), learning epochs (Section 8.3), and neighbourhood structures (Sections 8.4 and 8.5).

8.1 Experimental Setup

Datasets. All experiments are based on the following two grid datasets: (1) *Ebird* dataset [39], 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. Figure 7(a) shows the Ebird data distribution, where blue dots refer to cells with bird existence. We generate eight



(a) Ebird Data (b) MNCrime Data

Figure 7: Datasets Used in Experiments.

Parameter	Default Value
Grid Training Size (Ebird)	860 Cells
Grid Testing Size (Ebird)	140 Cells
Number of Predictors (Ebird)	3
Grid Training Size (MNCrimes)	72 Cells
Grid Testing Size (MNCrimes)	12 Cells
Number of Predictors (MNCrimes)	11

Table 1: Dataset-specific Parameters.

Parameter	Default Value
Learning Epochs E	1000
Neighbourhood Degree D	1, 8
Step Size α (*TurboReg*)	0.001
Number of Threads (*TurboReg*)	7
Factor Graph Partitions (*TurboReg*)	200
Standard Deviation σ (ngspatial)	1000

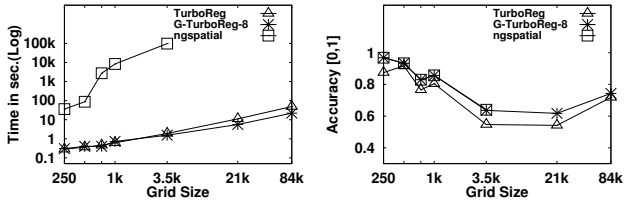
Table 2: Learning-specific Parameters.

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 binary predictors including whether number of bird observers high or not, whether the observing duration is long or not, and whether observers cover large spatial area or not. (2) *MNCrimes* dataset, which is a synthetic dataset about predicting the existence of bike theft crime in 87 neighborhoods in Minneapolis. Figure 7(b) shows the MNCrimes data distribution, where red and green cells refer to the crime existence and non-existence, respectively. This grid is constructed based on three public datasets about Minneapolis neighbourhoods [3], census [1] and crime incidents [3]. It also uses the information about other 11 crime types as binary predictors. In both Ebird and MNCrimes datasets, we randomly select 15% of the grid cells as testing data, and use the rest 85% for training. We use Ebird dataset in all experiments, except the experiment in the last Figure 11(b) which uses MNCrimes dataset.

Parameters. Unless otherwise mentioned, table 1 shows the default settings of both Ebird and MNCrimes datasets. Note that we use small number of grid cells as a default value, because the ngspatial technique fails in large cases. However, we have standalone experiments to show the scalability of *TurboReg* with large number of grid cells. Table 2 also shows the default learning configurations that are used with *TurboReg* and ngspatial. In most of experiments, we run two variations of our system: the basic *TurboReg* that has pairwise neighbourhood relationships (i.e., neighbourhood degree of 1), and another generalized variation with 8-ary neighbourhood relationships (i.e., neighbourhood degree of 8), referred to as *G-TurboReg-8* (See Section 3.2). In *G-TurboReg-8*, each predication has a bitwise-AND predicate over the whole 8 neighbours surrounding it. In case of ngspatial, we set the default standard deviation α of any bayesian prior distributions with the recommended value 1000 as in their documentation [23].

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

Metrics. In all experiments, we use the total running time of learning weights as a scalability evaluation metric, and the ratio of correctly predicted cells to the total number of test cells as an accuracy evaluation metric.



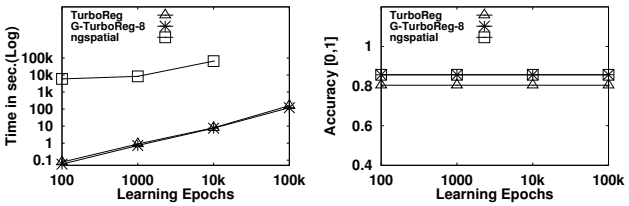
(a) Grid Size vs. Scalability (b) Grid Size vs. Accuracy

Figure 8: Effect of Grid Size on Scalability and Accuracy.

8.2 Effect of Grid Size

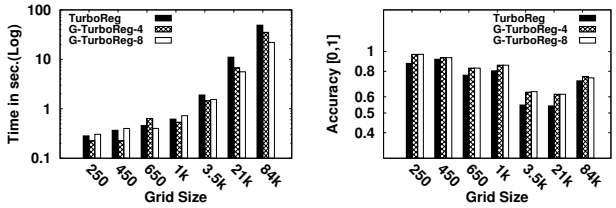
In this section, we compare the performance, both scalability and accuracy, of basic *TurboReg* and *G-TurboReg-8* with ngspatial, while having five different sizes of prediction grids.

Figure 8(a) shows the running time for each algorithm while scaling the grid size from 250 to 84k cells. For the five grid sizes, both *TurboReg* and *G-TurboReg-8* were able to significantly reduce the running time compared to ngspatial. Specifically, both *TurboReg* variants and ngspatial have an average running time of 6 seconds and 6 hours, respectively. This means that *TurboReg* has at least three orders of magnitude reduction in the running time over 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 [24]. In contrast, *TurboReg* 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 process. Note that the ngspatial curve in Figure 8(a) is incomplete after a grid size of $3.5k$ cells because of a failure in satisfying the memory requirements needed for its internal computations. The running times of *TurboReg* and *G-TurboReg-8* are almost identical, except with grid sizes larger than $21k$ cells which have 13 seconds average difference. This shows that *TurboReg* is efficient when scaling up the grid size regardless of the neighbourhood degree.



(a) Learning Epochs vs. Scalability (b) Learning Epochs vs. Accuracy

Figure 9: Effect of Number of Learning Epochs on Scalability and Accuracy.



(a) Neighbourhood Degree vs. Scalability (b) Neighbourhood Degree vs. Accuracy

Figure 10: Effect of Neighbourhood Degree on Scalability and Accuracy.

Figure 8(b) shows the accuracy for each algorithm while using the same grid sizes in Figure 8(a). In this experiment, we divide the cells in each grid into training and testing sets, where we randomly select 15% of cells for testing and keep the rest for training. We repeat this process 5 times and then average the accuracy results (we follow the same approach in the whole accuracy experiments in the paper). As can be seen in the figure, *G-TurboReg-8* has the same accuracy achieved by ngspatial, while basic *TurboReg* is at maximum 20% less accurate than both of them on average. The reason for that is the basic *TurboReg* captures less accurate neighbourhood dependencies than *G-TurboReg-8*. Note that the ngspatial curve is incomplete for grids with sizes more than $3.5k$ cells as in Figure 8(a).

8.3 Effect of Learning Epochs

In this section, we evaluate the performance, both scalability and accuracy, of basic *TurboReg* and *G-TurboReg-8* with ngspatial, while having four different values of learning epochs. In the following experiments, we fix the grid size to be $1k$ cells.

Figure 9(a) shows the running time for the different algorithms while changing the number of epochs from 100 to 100k. Both basic *TurboReg* and *G-TurboReg-8* significantly outperform ngspatial. They are at least 2 orders of magnitude faster than ngspatial. This is because of the parallel processing of learning epochs in *TurboReg* compared to the sequential learning in ngspatial. The results of ngspatial are incomplete after $10k$ epochs because its learning process requires saving huge intermediate state. In contrast, *TurboReg* never needs an intermediate state because it updates the model weights in place using the gradient descent optimization technique (See Algorithm 2). The figure also shows that the running times of *TurboReg* and *G-TurboReg-8* are identical and never depend on the neighbourhood degree. This confirms the complexity estimation of the weights learning algorithm in Section 7.

Figure 9(b) shows the accuracy of the different algorithms given the same setup in Figure 9(a). This experiment shows an interesting observation that both *TurboReg* and ngspatial can rapidly converge to their optimal values of weights (i.e., number of learning epochs less than 100). This is because ngspatial provides a good estimate to the prior of its predication and predictor variables, which makes the convergence process faster. In case of *TurboReg*, 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, *TurboReg* just needs small number of epochs for weights convergence.

8.4 Effect of Neighbourhood Degree

In this section, we evaluate the performance, both scalability and accuracy, of basic *TurboReg* and two generalized variations *G-TurboReg-8* and *G-TurboReg-4*, while scaling up the grid size. Unlike *G-TurboReg-8*, *G-TurboReg-4* considers neighbourhood degree of 4, in which each location prediction depends on neighbours that share edges with this location only.

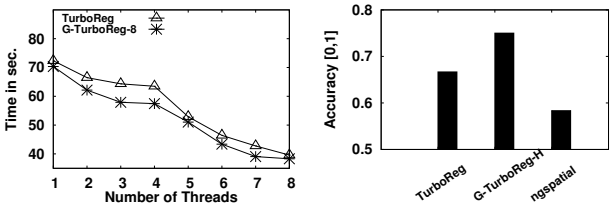
Figure 10(a) depicts the performance for each algorithm while using the same grid sizes in Figure 8(a). The results of this experiment confirm the previous ones we have shown in Figure 8(a). Increasing the neighbourhood degree leads to producing less number of predicates, and hence less number of factor graph nodes to process, which makes the weights learning process faster. In this experiment, the performance of different algorithms are almost similar in case of small grid sizes (i.e., the average accuracy difference between the three algorithms is less than 0.1 seconds). However, the difference becomes significant in case of large grid sizes (average of 16 seconds difference for grid size of $84k$ cells).

Figure 10(b) shows the accuracy of *TurboReg*, *G-TurboReg-4* and *G-TurboReg-8* using the same setup of grid sizes. As we can see, the accuracy of *TurboReg* is 9% less accurate than both *G-TurboReg-4* and *G-TurboReg-8*. Note that *G-TurboReg-4* and *G-TurboReg-8* almost have the same accuracy. This is a spatial case for the Ebird dataset because we observe that the significant information between neighbourhoods with degrees 8 and 4 is very little, which makes the accuracy in the two cases are pretty similar.

8.5 Effect of Number of Threads and Hybrid Neighbourhood Degrees

Figure 11(a) shows the effect of increasing the number of threads from 1 to 8 on *TurboReg* and *G-TurboReg-8*. These threads are used to parallelize the work in the weights learner module of *TurboReg*. As expected, the performance of both algorithms linearly improves. For example, the running time of *TurboReg* using 8 threads is 2 times faster than using 1 thread. This shows the ability of autologistic to scale up with system threads.

Figure 11(b) shows the accuracy of both autologistic and ngspatial in case of having hybrid neighbourhood degrees (i.e., each location has a different neighbourhood degree). In this experiment,



(a) Number of Threads vs. Scalability (b) Hybrid Niehgbourhood Degrees vs. Accuracy

Figure 11: Effect of Number of Threads on Scalability, and Hybrid Neighbourhood Degrees on Accuracy.

we use the MNCrimes dataset which consists of locations with 1 to 9 neighbourhood degrees. We compare the basic *TurboReg* that runs pairwise neighbouring depdencies, *G-TurboReg-H* that runs adaptive neighbourhood dependencies, and ngspatial. We find that the accuracy of *G-TurboReg-H* is higher than *TurboReg* with 12% and ngspatial with 29%.

9 RELATED WORK

Autologistic Theoretical Models. There are two main theoretical models of autologistic regression: (1) *Traditional model* [6] 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 biases its prediction to the presence case (i.e., predicted value is 1) in case of sparse training data. (2) *Centered model* [8] 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. *TurboReg* is the first framework to implement those models on a large-scale without sacrificing the accuracy of learned model parameters. Recent research has proposed an extension for spatio-temporal autologistic models [45, 46] (and centered variants [41]), 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 [24] for a comprehensive survey, and [23] 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 [7, 45](and centered variants [24]), Monte Carlo likelihood estimation [16](and centered variants [24, 45]), Bayesian inference estimation [28] (and centered variants [23, 45]). *TurboReg*, conversely, is the first technique to apply large-scale Markov Chain Monte Carlo estimation to learn the model parameters.

Other Spatial Regression Models. Autologistic models belong to the class of non-Gaussian spatial modelling [21], in which the spatial dependence between predictions is conditionally modelled through direct neighbours. However, there are three other classes: (1) linear spatial models [21], (2) spatial generalized linear models [17] and (3) Gaussian Markov random field models [33], 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 autologistic models focus on binary predictions, while other classes are mainly developed for continuous and categorical predictions.

10 CONCLUSIONS

This paper has delivered *TurboReg*, a scalable framework for building spatial logistic regression models (a.k.a autologistic models) to predict spatial binary data. *TurboReg* provides an efficient modeling for the autologistic regression problem using Markov Logic Network (MLN), which is a scalable statistical learning framework. *TurboReg* 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 and synthetic data sets shows that *TurboReg* achieves at least three orders of magnitude performance gain over existing state-of-the-art techniques while preserving the same accuracy.

REFERENCES

[1] MinnesotaCompass. http://www.mncompass.org/.
[2] NASA EarthData. https://earthdata.nasa.gov/earth-observation-data.
[3] OpenDataMinneapolis. http://opendata.minneapolismn.gov/.
[4] Nathalie Augustin, Moira A. Mugglestone, and Stephen T. Buckland. An Autologistic Model for the Spatial Distribution of Wildlife. *J. Appl. Ecol.*, 1996.
[5] Colin M. Beale, Jack J. Lennon, Jon M. Yearsley, Mark J. Brewer, and David A. Elston. Regression Analysis of Spatial Data. *Ecol. Lett.*, 2010.
[6] Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *J. Royal Stat. Soc.*, 1974.
[7] Julian Besag. Statistical Analysis of Non-Lattice Data. *J. Royal Stat. Soc.*, 1975.
[8] Petruta C. Caragea and Mark S. Kaiser. Autologistic Models with Interpretable Parameters. *JABES*, 2009.
[9] Yang Chen and Daisy Zhe Wang. Web-Scale Knowledge Inference Using Markov Logic Networks. *ICML SLG*, 2013.
[10] David R. Cox. The Regression Analysis of Binary Sequences (with discussion). *J. Royal Stat. Soc.*, 1958.
[11] Robert Crane and Luke K. McDowell. Evaluating Markov Logic Networks for Collective Classification. In *SIGKDD MLG*, 2011.
[12] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence.* Morgan and Claypool Publishers, 2009.
[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*, 1995.
[14] R.A. Finkel and J.L. Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 1974.
[15] Michael Genesereth and Nils Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers Inc., 1987.
[16] Charles J. Geyer. On the Convergence of Monte Carlo Maximum Likelihood Calculations. *J. Royal Stat. Soc.*, 1994.
[17] C. A. Gotway and W. W. Stroup. A Generalized Linear Model Approach to Spatial Data Analysis and Prediction. *JABES*, 1997.
[18] 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. *JABES*, 1997.
[19] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD Rec.*, 1984.
[20] Tjelmeland Hakon and Besag Julian. Markov Random Fields with Higher-order Interactions. *Scand. Stat. Theory Appl.*, 1998.
[21] Murali Haran. *Gaussian Random Field Models for Spatial Data.* Chapman and Hall/CRC, 2011.
[22] Fangliang He, Julie Zhou, and Hongtu Zhu. Autologistic Regression Model for the Distribution of Vegetation. *JABES*, 2003.
[23] John Hughes. ngspatial: A Package for Fitting the Centered Autologistic and Sparse Spatial Generalized Linear Mixed Models for Areal Data. *The R Journal*,

2014.
[24] John Hughes, Murali Haran, and Petruta C. Caragea. Autologistic Models for Binary Data on a Lattice. *Environmetrics*, 2011.
[25] Nikos Koutsias. An Autologistic Regression Model for Increasing the Accuracy of Burned Surface Mapping using Landsat Thematic Mapper Data. *Int. J. Remote Sens.*, 2003.
[26] Catherine Linard and Andrew J. Tatem. Large-scale Spatial Population Databases in Infectious Disease Research. *Int. J. Health Geogr.*, 2012.
[27] Daphne Lopez, M. Gunasekaran, and B. Senthil Murugan. Spatial Big Data Analytics of Influenza Epidemic in Vellore, India. In *IEEE Big Data*, 2014.
[28] 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*, 2006.
[29] Sangkil Moon and Gary J. Russell. Predicting Product Purchase from Inferred Customer Similarity: An Autologistic Model Approach. *Management Science*, 2008.
[30] J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *TODS*, 9(1), 1984.
[31] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. *VLDB*, 2011.
[32] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *VLDB J.*, 2017.
[33] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 2005.
[34] Nikita A. Sakhanenko and David J. Galas. Markov Logic Networks in the Analysis of Genetic Data. *J. Comput. Biol.*, 2010.
[35] 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*, 2005.
[36] J. Michael Scott, Patricia J. Heglund, and Michael L. Morrison et al. Predicting Species Occurrences: Issues of Accuracy and Scale. *Journal of Mammalogy*, 2002.
[37] Michael Sherman, Tatiyana V. Apanasovich, and Raymond J. Carroll. On estimation in binary autologistic spatial models. *J. Stat. Comput. Simul.*, 2006.
[38] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental Knowledge Base Construction Using DeepDive. *PVLDB*, 2015.
[39] 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*, 2009.
[40] W. R. Tobler. *Cellular Geography: Philosophy in Geography*. Springer, Dordrecht, 1979.
[41] Zilong Wang and Yanbing Zheng. Analysis of Binary Data via a Centered Spatial-temporal Autologistic Regression Model. *Environ. and Ecol. Stat.*, 2013.
[42] Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable Probabilistic Databases with Factor Graphs and MCMC. *PVLDB*, 2010.
[43] 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*, 2017.
[44] Ce Zhang and Christopher Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *SIGMOD*, 2013.
[45] Yanbing Zheng and Jun Zhu. Markov Chain Monte Carlo for a Spatial-Temporal Autologistic Regression Model. *J. Comput. Graph. Stat.*, 2008.
[46] Jun Zhu, Hsin-Cheng Huang, and Jungpin Wu. Modeling Spatial-temporal Binary Data using Markov Random Fields. *JABES*, 2005.
[47] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized Stochastic Gradient Descent. In *NIPS*, 2010.

A THEORETICAL FOUNDATION OF *TURBOREG* USING MLN

Theorem 1. *Given an autologistic model with a set of n prediction variables $\mathcal{Z} = \{z_1, ..., z_n\}$ defined over n locations $\mathcal{L} = \{l_1, ..., l_n\}$, a set of m weighted predictor variables $\beta\mathcal{X}(i) = \{\beta_1 x_1(i), ..., \beta_m x_m(i)\}$ at each location i and neighbouring weight η, there is an equivalent Markov Logic Network (MLN) to this model, if and only if: (1) each predictor-based regression term $\beta_j x_j(i)$ at location l_i has an equivalent bitwise-AND predicate $z_i \wedge x_j(i)$ with weight β_j. (2) each neighbour-based regression term ηz_k at location l_i has an equivalent bitwise-AND predicate $z_i \wedge z_k$ with weight η.*

Proof. Assume a model that consists of $n + nm$ binary random variables $\mathcal{V} = \{\mathcal{Z}, \mathcal{X}\} = \{z_1, ..., z_n, x_1(1), ..., x_m(n)\}$. In addition, assume a set of constraints $\mathcal{F} = \{F_1, ..., F_n\}$ are defined over variables $\mathcal{V}$, where constraints F_i at location l_i consist of two subsets of constraints: (1) a set of m bitwise-AND predicates $\{z_i \wedge x_j(i) \mid j = 1, ..., m\}$ with β weights (each predicate corresponds to a predictor-based regression term), and (2) a set of s_i bitwise-AND predicates $\{z_i \wedge z_k \mid k \in \mathcal{N}_i\}$ with η weight (each corresponds to a neighbour-based regression term) where s_i is the size of neighbouring locations $\mathcal{N}_i$ of location l_i. 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 $\mathcal{V}$ is estimated using Equation 2.

Since $\mathcal{Z}$ and $\mathcal{X}$ are binary variables, the evaluation of any bitwise-AND predicate over them can be represented as a mathematical multiplication (i.e., the value of $z_i \wedge x_j(i)$ is $z_i x_j(i)$ and the value of $z_i \wedge z_k$ is $z_i z_k$). As a result, the joint probability distribution of $\mathcal{V}$ from Equation 2 becomes:

$$Pr(\mathcal{V} = v) = Pr(\mathcal{Z}, \mathcal{X}) = \frac{1}{C} \exp\Big(\sum_{i=1}^{n} \sum_{j=1}^{m} \beta_j z_i x_j(i) + \eta \sum_{i=1}^{n} \sum_{k \in \mathcal{N}_i} z_i z_k \Big) \tag{3}$$

Based on Equation 3, the conditional probability distribution of any prediction variable z_i at location l_i given the predictor variables $\mathcal{X}(i)$ at l_i and its neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as:

$$Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i}) = \frac{1}{C} \exp\Big(\sum_{j=1}^{m} \beta_j z_i x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_i z_k \Big) \tag{4}$$

By substituting with possible values of z_i (i.e., either 1 or 0) in Equation 4, we can obtain the ratio between the conditional probabilities of $z_i = 1$ and $z_i = 0$ as follows:

$$\begin{aligned} \frac{Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i = 0 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})} &= \frac{\exp\Big(1 \times \sum_{j=1}^{m} \beta_j x_j(i) + 1 \times \eta \sum_{k \in \mathcal{N}_i} z_k\Big)}{\exp\Big(0 \times \sum_{j=1}^{m} \beta_j x_j(i) + 0 \times \eta \sum_{k \in \mathcal{N}_i} z_k\Big)} \\ &= \frac{\exp\Big(\sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k \Big)}{\exp\Big(0\Big)} \\ &= \exp\Big(\sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k \Big) \end{aligned} \tag{5}$$

By taking the log value of both LHS and RHS of Equation 5, we obtain the autologistic model defined in Equation 1. This means that the assumed model at the beginning, which can be represented with MLN, is equivalent to the basic autologistic model.