UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

LEARNING ENSEMBLES OF LINEAR GAUSSIAN NETWORKS FOR NONLINEAR AND SPATIAL DATA

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

MORGAN ROBERTSON
Norman, Oklahoma

2014

LEARNING ENSEMBLES OF LINEAR GAUSSIAN NETWORKS FOR NONLINEAR AND SPATIAL DATA

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dedication

To Frederick, my friend and companion in the pursuit of knowledge and truth.

Acknowledgements

I would like to thank David John Gagne II for answering my questions related to meteorolgy, providing suggestions regarding visualization tools, and generating the WRF dataset used in this thesis. Scott Hellman, whose work provides much of the basis for this thesis, answered many of my questions about his work and provided ideas for my own. I am grateful to Frederick Wagner both for helping me bounce around ideas and for his support throughout the course of my research. Finally, I want to thank my advisor, Dr. Amy McGovern, for the guidance and insight that made this thesis possible.

Table of Contents

List Of Tables

List Of Figures

Abstract

The ability to accurately model nonlinear relationships is important in many real-world domains. Ensembled linear Gaussian networks (ELGNs), a class of ensembled Bayesian networks in which each conditional probability distribution (CPD) is a linear Gaussian distribution, have been shown to be capable of performing nonlinear regression. In this thesis, we introduce weighted bagging for ELGNs, which allows the components of the ensemble to train on individual clusters within the data and better model local relationships. When combined with Gaussian mixture regression (GMR), we have a powerful model for performing nonlinear regression. Using both synthetic and real datasets, we demonstrate ELGN's effectiveness and explore how various parameter settings affect performance.

In addition to possessing nonlinear relationships between variables, many domains include spatial and temporal aspects. Dynamic Bayesian networks (DBNs) are a common model for representing the evolution of a system over time. To model correlations between variables at different points in space, we introduce a method for performing spatial inference. Rather than making predictions for a set of query variables based only on evidence observed at the same point in space, we also take into account observed evidence at nearby points. Nearby evidence is weighted by a kernel function, which results in strong correlations between predictions at points close together in space. We apply spatial inference to predict reflectivity in a weather dataset.

Chapter 1

Introduction

Over the last few decades, the use of Bayesian networks in machine learning has become more widespread, offering an alternative to other models such as decision trees, neural networks, and support vector machines. In contrast to many other commonly used models, a Bayesian network is a true probabilistic model, capable of directly encoding a joint probability distribution. Due to their probabilistic nature, Bayesian networks offer an easy-to-interpret model of the dependence relationships among variables, and also provide information about the uncertanty in their predictions.

As with other models, Bayesian networks become more powerful when ensembled (Utz 2010) and standard ensembling techniques including randomization (Breiman 2001) and bagging (Breiman 1996) can be applied. For domains with continuous data, ensembles of linear Gaussian networks (ELGNs) (Hellman 2012) can be employed. By using Gaussian mixture regression (Sung 2004) for inference, Hellman demonstrated that an ELGN has the ability to perform nonlinear regression. Unfortunately, the use of traditional bagging during the learning process severely hinders the ability of ELGNs to accurately model nonlinear relationships. As an alternative, we present a form of weighted bagging, which allows each network in the ensemble to better model local relationships for clusters within the data.

Dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1989) are commonly used to represent systems that evolve over time. They can easily be combined with linear Gaussian networks and ensembling techniques to model continuous data over a sequence of discrete time steps. Unfortunately, there is not a commonly accepted method for modeling spatial relationships with Bayesian networks. Unlike time, space can be multidimensional and influence can flow in more than one direction. These characteristics make spatial relationships much more difficult to capture in comparison to temporal relationships. To help us take into account spatial correlations within the data, we present a method for performing spatial inference in Bayesian networks. Rather than using observations only at the location for which we wish to make a prediction, we also consider obversations of variables at nearby points.

We take Chapter 2 to provide the background for Bayesian networks. We discuss independence properties, learning algorithms, and linear Gaussian and dynamic Bayesian networks. In Chapter 3, we present weighted bagging for ELGNs as well as spatial inference. Chapter 4 presents experiments that we carried out to test our new methods and Chapter 5 discusses our conclusions and ideas for future work.

Chapter 2

Background

In this chapter, we provide some background about Bayesian networks. We first discuss their representation, which encodes a joint probability distribution, and independence relationships. Next, we examine structure learning and parameter estimation. Finally, we discuss linear Gaussian and dynamic Bayesian networks, which are used to model continuous and temporal data, respectively.

2.1 Notation

For ease of readability, we follow a few notational conventions to distinguish between different types of entities. Variables are represented by capital letters, such as X. Collections of variables, including sets, vectors, and matrices are denoted by a boldface capital letter, such as $\mathbf{X}$. Multivariate normal distributions are represented as $N(\mu; \mathbf{\Sigma})$ where μ is the mean vector and $\mathbf{\Sigma}$ is the covariance matrix. $\mu_{\mathbf{X}}$ and $\mathbf{\Sigma}_{\mathbf{XY}}$ denote partitions of the mean vector and covariance matrix corresponding to the sets of variables $\mathbf{X}$ and $\mathbf{X} \cup \mathbf{Y}$, respectively. Variance of a single variable X is denoted by σ^2_X.

2.2 Bayesian Networks

A Bayesian network is a probabilistic graphical model that encodes a joint probability distribution over a set of random variables. Variables and dependencies are represented by a directed acyclic graph (DAG) where nodes correspond to variables and edges correspond to direct dependencies between connected nodes (Pearl 1985). The parents of a node X, denoted $\mathbf{Pa}_X$, are the nodes in the graph that have a directed edge to X. The value of a random variable corresponding to a particular node depends directly on the values of its parent nodes. Thus at each node we have a conditional probability distribution (CPD) of the form $P(X|\mathbf{Pa}_X)$.

Consider the problem of representing a joint distribution over a set of n discrete binary-valued random variables. Since each variable has two possible instantiations, we have 2^n total possible instantiations of all variables. If we represent the joint distribution as a table with a row for each possible instantiation, the size of the table grows exponentially as the number of variables increases. Clearly, working with a model that explicitly lists a probability for each instantiation is intractable for any dataset with more than a handful of variables. If we wish to use multi-valued variables instead of binary variables, as is often required in practice, the situation is even worse. Fortunately, we can exploit conditional independencies between variables to greatly reduce the number of parameters required to specify the joint distribution. This is exactly what happens in Bayesian networks, resulting in a relatively compact model for the joint distribution (Pearl 1988).

Using the chain rule from probability theory, we can factorize a joint distribution into a product of terms:

$$P(X_n, ..., X_1) = P(X_n|X_{n-1}, ..., X_1)...P(X_2|X_1)P(X_1). \tag{2.1}$$

Given an arbitrary ordering of the variables, each term consists of a conditional distribution over a variable given its predecessors in the ordering. If we know that conditional independencies exist between variables, we can simplify each term in the factorization by removing some variables to the right of the conditioning bar. Formally, we have from probability theory that $P(X|Y)P(Y) = P(X)P(Y)$ if and only if X and Y are independent. For each variable X_i, $1 \leq i < n$, define $\mathbf{Pa}_i$ as the set of variables in the ordering that X_i is conditionally dependent upon. Then we can factorize the joint distribution:

$$P(X_n, ..., X_1) = P(X_n|\mathbf{Pa}_n)...P(X_2|\mathbf{Pa}_2)P(X_1). \tag{2.2}$$

A Bayesian network encodes this factoring (Pearl 1985), where $\mathbf{Pa}_i$ are the parents of node X_i and the ordering of variables is a topological sort of the graph. In the worst case, this factoring can have as many parameters as the full specification of the joint distribution if no conditional independencies exist between the variables, but this is rare in practice. For a Bayesian network, this case manifests as a fully connected graph. An example of a Bayesian network is shown in Figure 2.1 and will be analyzed more closely in Section 2.3.

Aside from their ability to compactly represent a joint distribution, Bayesian networks have several other useful properties. In contrast to many other machine learning models, such as decision trees, SVMs, and many common types of neural networks, Bayesian networks can be treated as a form of unsupervised learning. That is, there need not be an explicit target attribute to be learned. Instead, relationships between all variables are learned in conjunction. This model of learning is very powerful, as it allows us to make predictions for any subset of variables given any other subset of variables. We can easily make predictions even if observation data is missing; that is, we have only a partial instantiation of observed variables.

In addition, the structure of a Bayesian network provides an easy-to interpret model of the relationships between variables in a dataset. Unlike a neural network, which is often considered a black-box model, we can easily examine the edges of a Bayesian network to see how different variables are correlated. Using properties of conditional independence encoded by the graph, it is possible to determine how variables indirectly influence one another. The notion of d-separation, discussed in the next section, is very powerful for this purpose.

2.3 Independence Properties of Bayesian Networks

As already mentioned, the structure of a Bayesian network encodes a set of conditional independencies between variables. Due to the importance of dependence relationships in knowledge discovery and model interpretation, we will now describe how these relationships are encoded and how observing evidence affects these relationships.

To illustrate how dependencies are represented in Bayesian networks, suppose that we have obtained the network structure shown in Figure 2.1. The network represents the process of a student finding a job after completing graduate school. Intelligence influences the student's grades, while technical skills influence both grades and how well the job interview goes. The interview combined with grades contribute to the company's perception of the candidate, and the company's perception determines whether she gets the job. Suppose we know that the company has a very positive perception of the job candidate. Would observing other variables in the network, such as *Skills* or *Grades*, give us more information about whether she will get the job? Given the network structure, the answer is no. It turns out that *Job* is independent of all other variables in the network given its parent variable, *Perception*. Next, we'll formally state the conditional independence properties of Bayesian networks that leads to this and other conclusions about relationships between variables in the network.

We already know that two variables X and Y are directly dependent (i.e. correlated) if they share an edge. However, variables can also be dependent if there isn't a direct edge between them (Pearl 1986). If we have three variables X, Y, and Z connected as $X - Z - Y$, there are four possible ways to orient the edges:

1. $X \rightarrow Z \rightarrow Y$
2. $X \leftarrow Z \leftarrow Y$
3. $X \rightarrow Z \leftarrow Y$
4. $X \leftarrow Z \rightarrow Y$

In the first two cases, X, Z, and Y form a chain. In either case, X and Y are conditionally independent given Z. If we know the value of Z and want to predict Y, for example, knowing the value of X gives us no additional information that is useful in predicting Y. In the third case, X and Y are both parents of Z, which is to say that Z is a common effect of X and Y. This is also referred to as a *v-structure* in the literature. Given the value of Z, X and Y are conditionally dependent. Intuitively, if we know the value of Z and then observe X, we can then make inferences about Y based on our knowledge of X. Take the

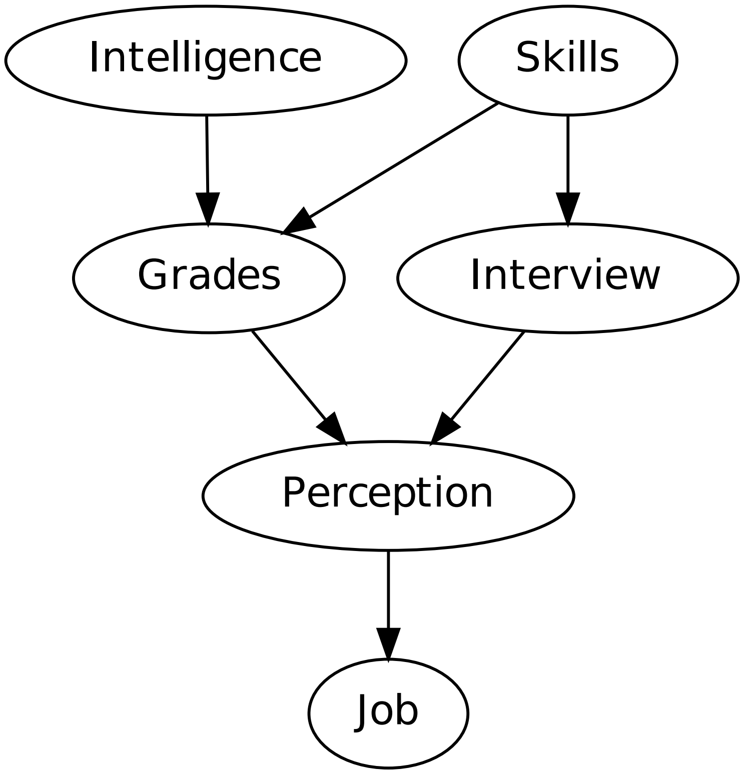


Figure 2.1: *A Bayesian network representing the process of finding a job.*

job search example to illustrate this. Assume that the company's perception of the candidate is very positive. Given this, it is likely that the interview went well, the candidate's grades are high, or both. Now suppose we find out that the interview went poorly. We can infer that the candidate's grades are likely high since the company has a positive view of her in spite of the interview. The ability to make inferences of this sort is known as *explaining away*. In the fourth case, Z is a parent of both X and Y; that is, Z is a common cause of X and Y. In this case, X and Y are conditionally independent given Z as in the first two cases. In the job search example, suppose we know the candidate has outstanding technical skills. Given this information, also knowing about her grades gives us no additional information about how the interview went.

The four cases outlined above can be used to establish conditional independence relationships for groups of variables, leading to the notion of *d-separation* (Pearl 1986; Geiger et al. 1990). In cases (1), (2), and (4), the path between X and Y is said to be blocked when Z is observed and active when Z is not observed. In case (3) on the other hand, the path between X and Y is blocked when Z is unobserved and active when Z is observed. Intuitively, influence can flow between X and Y when the path between them is active (i.e. not blocked). For an arbitrary path p between X and Y and a set of variables $\mathbf{Z}$, p is blocked by $\mathbf{Z}$ if either or both of the following conditions hold:

1. p contains a subpath $A \rightarrow C \rightarrow B$, $A \leftarrow C \leftarrow B$, or $A \leftarrow C \rightarrow B$ such that C is in $\mathbf{Z}$

2. p contains a subpath $A \rightarrow C \leftarrow B$ such that neither C nor any descendant of C is in $\mathbf{Z}$

For sets of variables $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, we define $\mathbf{X}$ and $\mathbf{Y}$ to be d-separated if and only if the path between every pair of nodes (X, Y) for $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ is blocked by $\mathbf{Z}$. D-separation in a graphical model corresponds to conditional independence: if $\mathbf{X}$ and $\mathbf{Y}$ are d-separated given $\mathbf{Z}$, then $\mathbf{X}$ and $\mathbf{Y}$ are conditionally independent given $\mathbf{Z}$.

2.4 Learning Bayesian Networks

Although the structure and parameters of a Bayesian network can sometimes be specified by a domain expert, it is often more practical to automatically learn the network from data, especially if the network consists of more than a handful of variables. In this section, we will discuss algorithms and issues related to learning Bayesian networks from data.

Assume that we have a dataset D consisting of m instances which are independent and identically distributed (i.i.d.) samples generated from some underlying distribution P. The learning task is to construct a model $M = (G, \theta)$ that consists of a graph structure G and a set of parameters θ such that M provides a good approximation for P. The meaning of "good" depends on a number of issues and will be discussed later. Learning our model M consists of two distinct phases: determining the structure of G and estimating the parameters θ. Approaches to structure learning include constraint-based, search-and-score, and hybrid algorithms. For parameter estimation, we will discuss maximum likelihood estimation (MLE).

2.4.1 Structure Learning

As we have seen, a Bayesian network represents a set of conditional independence relationships between variables. The goal of structure learning algorithms is to determine these relationships, which are manifested as a set of directed edges between variables in the network.

In constraint-based algorithms, direct tests of independence are used to establish whether two variables share an edge. Well-known test statistics such as χ^2 and G^2 are computed for discrete data, and measures of correlation can be computed for continuous data. In the constraint-based approach, two variables X and Y share an edge if they cannot be shown to be conditionally independent given another subset of variables. Formally, if there exists a subset of variables $\mathbf{Z} \subseteq \mathbf{V}$ such that $Ind(X, Y|\mathbf{Z})$, then X and Y do not share an edge; if no such subset exists, then X and Y do share an edge. The general approach of constraint-based algorithms is as follows: For each pair of variables X, $Y \in \mathbf{V}$, attempt to find a subset $\mathbf{Z} \subseteq \mathbf{V} - \{X, Y\}$ such that $Ind(X, Y|\mathbf{Z})$. If no such $\mathbf{Z}$ can be found, then X and Y must share an edge.

The PC Algorithm (Spirtes et al. 2000) is an example of one of the first constraint-based structure learning algorithms. In the constraint identification phase of the algorithm, we start with a complete undirected graph G. For increasing conditioning set sizes, an independence test is performed on each remaining edge in the graph. If the variables sharing the edge are conditionally independent given the conditioning set, the edge is removed. Once the skeleton is identified, edges are oriented using a set of rules to ensure that the resulting structure is a DAG.

The constraint-identification portion of the PC algorithm (without edge orientation) is also known as Fast Adjacency Search (FAS). Rather than using a rule-based algorithm for orienting edges as in the PC algorithm, FAS can be followed by a search, such as greedy hillclimbing, to orient edges. This approach was taken by Fast (2010) and Hellman (2012). In this thesis, we use the max-min hillclimbing (MMHC) algorithm for structure learning (Tsamardinos et al. 2006). Like FAS, the first stage of the algorithm performs constraint-identification. Hillclimbing is then used to orient edges.

The other major class of structure learning algorithms is search-and-score. The basic approach here is to start with some network structure, then iteratively add, remove, or reverse edges until the final structure is obtained (Koller and Friedman 2009). The search is guided by the application of a chosen scoring metric to the structure after each modification. The most basic search-and-score algorithm is greedy hillclimbing. We start with an arbitrary structure G and define the set of possible actions to include the addition of any edge not in the graph; the removal of any edge in the graph; and the reversal of any edge in the graph such that the graph remains acyclic. On each iteration of the search, we consider each possible action and apply whichever one results in the greatest increase in the overall score of the network structure. The algorithm terminates when no action increases the score, i.e. a local maximum has been reached.

2.4.2 Parameter Estimation

In addition to learning a structure G for our Bayesian network, we must also estimate a set of parameters θ in order to encode a joint probability distribution. In this thesis, we use maximum likelihood estimation (MLE) (Koller and Friedman 2009) to estimate parameters.

Given a dataset D as described at the beginning of Section 2.4, MLE finds parameters θ such that the likelihood of θ is maximized with respect to D. Formally, we choose θ to maximize the liklelihood function:

$$L(\theta|D) = \prod_{i=0}^{m} P(D_i|\theta) \tag{2.3}$$

where D_i is the ith instance in D and m is the total number of instances (Koller and Friedman 2009). Thus, we want to select θ such that the product of probabilities for each instance is maximized given those parameters.

For Bayesian networks, we can exploit an important property of the likelihood function called *global decomposition* (Koller and Friedman 2009). This property allows us to decompose the likelihood function from Equation 2.3 into a product of terms where each term corresponds to the local likelihood of parameters at a node. The likelihood function becomes

$$L(\theta|D) = \prod_{i=0}^{m} \prod_{v \in \mathbf{V}} P(D_i|\theta_{v|\mathbf{Pa}_v}) \tag{2.4}$$

where $\mathbf{V}$ is the set of variables in the network and $\theta_{v|\mathbf{Pa}_v}$ is the set of parameters for a node given its parents.

2.5 Linear Gaussian Networks

In this thesis, our goal is to model continuous data using Bayesian networks. We will henceforth restrict our attention to the class of BNs where the CPD at each node is a Gaussian distribution whose mean is a linear combination of the values of its parent nodes. This type of network has been well-studied in the literature (Shachter and Kenley 1989; Geiger and Heckerman 1994; Heckerman and Geiger 1995) and is commonly known as a linear Gaussian network (LGN). For real-world datasets, the assumptions of Gaussianity and linear relationships between variables are quite strong, but the flexibility and mathematical conveniences resulting from this model sometimes justify its use even when the assumptions are violated. To demonstrate the validity of this assertion, we will next investigate some properties of LGNs and will later extend the work of Hellman to accurately model nonlinear functions using ensembles of LGNs.

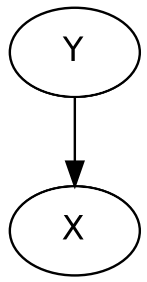
For a node Y with parents $\mathbf{X} = (X_1, ..., X_k)$, the conditional probability distribution of Y takes the form

$$P(Y|\mathbf{X}) = N(\beta_0 + \beta^T X; \sigma^2) \tag{2.5}$$

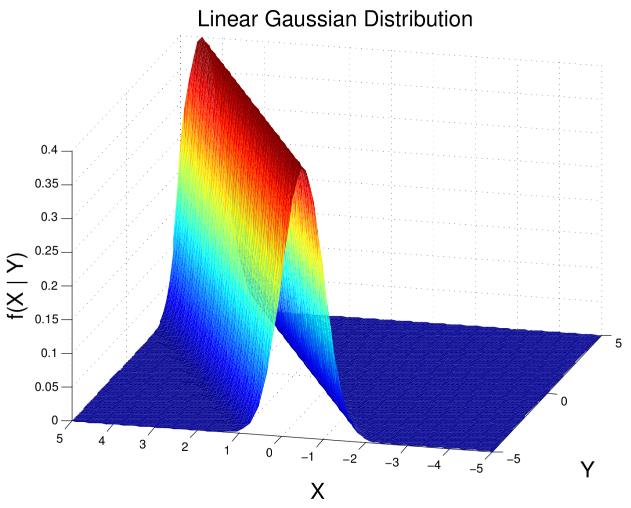
where β_0 is a constant scalar, β is a vector of constant weights, and σ^2 is the variance of Y. Notice that the mean of Y is strictly a linear function of the values of its parents. Conveniently, when each node in the network is linear Gaussian, it can be shown that the network taken as a whole is equivalent to a multivariate Gaussian distribution over the variables in the network (Shachter and Kenley 1989; Wermuth 1980). As we'll see in Section 2.5.5, this fact allows us to perform inference in closed form in polynomial time.

2.5.1 Conversion to Gaussian Distribution

The equivalence of LGNs with multivariate Gaussian distributions allows us to convert a parameterized network to a mean-covariance representation. To compute the covariance matrix, we use the algorithm presented by Shachter and Kenley (1989), shown in Algorithm 2.1. First, a topological sort of the graph is performed, resulting in an ordering $orderedNodes$. Next, we iterate over nodes according to the ordering and populate the covariance matrix by calculating the variance of each node and its covariance with all of its predecessors in



(a) Linear Gaussian Network



(b) CPD for X

Figure 2.2: *A linear Gaussian network consisting of a node X with a single parent Y. A plot of the CPD for X is shown where $P(X|Y) = N\left(\frac{1}{2}Y + 2.5; 1\right)$. Plot from Hellman (2012).*

$orderNodes$. The mean for each node is computed as a function of the means of its parents according to Equation 2.5.

2.5.2 Scoring

As with discrete Bayesian networks, a scoring metric is required to perform structure learning with search-and-score algorithms. In this thesis, we use the Bayesian metric for Gaussian networks having score $e$quivalence, known as the BGe score, developed by Geiger and Heckerman (1994). The BGe score of an LGN is proportional to the posterior probability of the structure given the data:

$$P(G|D) = \frac{P(D|G)P(G)}{\sum_{G^* \in \mathbf{G}} P(D|G^*)P(G^*)}. \tag{2.6}$$

Using Bayes' rule, Equation (2.6) gives the posterior in terms of the prior probability of the network structure and the marginal likelihood of the data given the

Algorithm 2.1: Computing the corresponding covariance matrix for a LGN

```
Input: $BN$, a Bayesian network with $n$ nodes
Output: $\mathbf{\Sigma}$, the covariance matrix
$orderedNodes \leftarrow$ topologicalSort($BN$)
for $i = 1, ..., n$ do
    $node = orderedNodes[i]$
    $\beta = node$.parentWeights
    $K =$ indices of $node$.parents in $orderedNodes$
    for $j = 1, ..., i$ do
        $\mathbf{\Sigma}_{ij} \leftarrow \mathbf{\Sigma}_{ji} \leftarrow \sum_{k \in K} \mathbf{\Sigma}_{jk}\beta_k$
    end
    $\mathbf{\Sigma}_{ii} \leftarrow node$.variance $+ \sum_{k \in K} \mathbf{\Sigma}_{jk}\beta_k$
end
```

network structure. The prior $P(G)$ encodes any prior knowledge of the network structure that we may possess. If no prior knowledge exists, the uniform distribution is often used, as is done in this thesis. The denominator of Equation (2.6) involves a summation over all possible network structures $\mathbf{G}$. Fortunately, this evaluates to a constant and can safely be ignored for scoring purposes.

A number of assumptions are made when using BGe as a scoring metric, outlined by Geiger and Heckerman (1994). Perhaps most importantly, it is assumed that each instance in the training dataset is drawn randomly from a multivariate Gaussian distribution and that the dataset is complete, i.e. there is no missing data. The mean vector μ and precision matrix $\mathbf{W} = \mathbf{\Sigma}^{-1}$ of the generating distribution are assumed unknown and distributed according to a normal-Wishart prior. The use of the normal-Wishart prior in this case is analagous to the use of a Dirichlet prior in the BDe score, which is a commonly used Bayesian scoring metric when the dataset consists of discrete variables (Heckerman et al. 1995).

Due to the mathematical verbosity involved in computing the BGe score, we refer the reader to the work of Geiger and Heckerman (1994) if a thorough analysis of the derivation and mathematical details of the BGe score is desired. One important point about the BGe score is that it requires us to specify a prior mean vector and covariance matrix. In this thesis, we assume no prior knowledge of our datasets, which leads us to use the identity matrix as our prior covariance matrix and the zero vector as our prior mean vector.

Bayesian scoring metrics, including BGe, have several convenient properties. As we saw previously in Equation 2.6, $P(G|D) \propto P(D|G)P(G)$. The $P(D|G)$ term is computed as follows:

$$P(D|G) = \int P(D|G, \theta_G)P(\theta_G|G)\mathrm{d}\theta_G \tag{2.7}$$

where θ_G is the set of parameters for a specific network structure G (Friedman and Koller 2000). In computing a Bayesian score, we are effectively averaging over all possible values of parameters given a specific network structure. This is in contrast to non-Bayesian scores, such as the Bayesian Information Criterion (BIC) (Schwarz et al. 1978), which use the MLE of parameters instead of averaging. The result is that Bayesian scores implicitly penalize model complexity and thereby reduce the potential for overfitting. In the BIC score, an explicit regularization term is required to penalize complexity.

Another useful property of BGe is decomposability, which results from the assumptions of parameter modularity and parameter independence. Parameter modularity, expressed as $P(\theta_{X_i|\mathbf{U}}|G) = P(\theta_{X_i|\mathbf{U}}|G')$, states that a node X_i with the same set of parents $\mathbf{U}$ in two different graph structures G and G' have equal likelihoods. Parameter independence, expressed as $P(\theta_G|G) = \prod_i P(\theta_{X_i|\mathbf{U}}|G)$, states that the likelihood of parameters given a specific graph structure decomposes into a product of likelihoods computed for the parameters at each node. These two properties allow us to compute the BGe score locally for a node and its parents without considering the rest of the network strucutre. This greatly reduces the computational resources required for search algorithms such as greedy hill climbing. For each action (addition, deletion, or reversal of an edge) in such an algorithm, the score need be recomputed only for the affected nodes.

2.5.3 Independence Tests

Constraint-based structure learning algorithms require tests of conditional independence — that is, for two variables X and Y and a conditioning set of variables $\mathbf{Z}$, we need to be able to establish whether X and Y are conditionally independent given $\mathbf{Z}$. Assuming that our variables are jointly distributed according to a multivariate Gaussian distribution, we can use partial correlation combined with Fisher's Z-transformation to perform hypothesis tests of independence.

The partial correlation of variables X and Y with a *controlling set* $\mathbf{Z}$, written $\rho_{XY\cdot\mathbf{Z}}$, is a measure of the linear dependence between X and Y when the effects of $\mathbf{Z}$ are removed. When the controlling set is empty, partial correlation reduces to Pearson correlation for two variables. In calculating partial correlation, we must assume that all pairs of variables are linearly related, which is consistent with the LGN framework. One of several methods to compute partial correlation is with the following recursive formula (Morrison 1967):

$$\rho_{XY\cdot\mathbf{Z}\cup\{Z_0\}} = \frac{\rho_{XY\cdot\mathbf{Z}} - \rho_{XZ_0\cdot\mathbf{Z}}\rho_{YZ_0\cdot\mathbf{Z}}}{\sqrt{1-\rho^2_{XZ_0\cdot\mathbf{Z}}}\sqrt{1-\rho^2_{YZ_0\cdot\mathbf{Z}}}}. \tag{2.8}$$

Using this formula with a dynamic programming technique, computation can be done in cubic time with respect to the number of variables.

Two Gaussian random variables are conditionally independent given a controlling set if and only if their partial correlation is zero (Baba et al. 2004). In order to test this, we apply Fisher's Z-transformation (Fisher et al. 1915) to the computed partial correlation coefficient, then perform a two-tailed hypothesis test using the normal distribution. We assume independence as the null hypothesis and reject this hypothesis if the resulting p-value is less than a specified threshold. To account for the uncertainty in our computed partial correlation resulting from a finite sample size, we compute Fisher's Z as done by Spirtes et al. (2000):

$$z(\rho_{XY\cdot\mathbf{Z}}, n) = \frac{1}{2}\sqrt{n - |\mathbf{Z}| - 3}\,\ln\frac{|1+\rho_{XY\cdot\mathbf{Z}}|}{|1-\rho_{XY\cdot\mathbf{Z}}|} \tag{2.9}$$

where n is the sample size.

2.5.4 Parameter Estimation

In section 2.4.2, we gave a general overview for using MLE to estimate the parameters of a Bayesian network once the structure has been learned. We will now examine parameter estimation for the specific case of LGNs. Recall from Equation 2.5 that the mean at each node is a linear linear combination of its parents' means. Thus for each node we must learn the parameters β_0 and β as well as the estimate the variance σ^2. We use the method presented by Koller and Friedman (2009).

For a node Y with parents $\mathbf{X}$, we first compute the covariance matrix $\mathbf{\Sigma}$ and mean vector μ for $\mathbf{X} \cup \{Y\}$. Next, we compute our MLE parameters as follows:

$$\beta_0 = \mu_Y - \mathbf{\Sigma}_{Y\mathbf{X}}{\mathbf{\Sigma}_{\mathbf{X}\mathbf{X}}}^{-1}\mu_{\mathbf{X}}, \tag{2.10}$$

$$\beta = \mathbf{\Sigma_{XX}}^{-1}\mathbf{\Sigma}_{\mathbf{X}Y}, \tag{2.11}$$

$$\sigma^2 = \mathbf{\Sigma}_{YY} - \mathbf{\Sigma}_{Y\mathbf{X}}\mathbf{\Sigma_{XX}}^{-1}\mathbf{\Sigma}_{\mathbf{X}Y}. \tag{2.12}$$

2.5.5 Inference

One extremely useful property of LGNs is that inference can be done in closed form, requiring only polynomial time. This is in contrast to the general case, where the time complexity of exact inference is exponential in the size of the network (Cooper 1990). To perform inference on an LGN containing variables $\mathbf{V}$, we first convert the network to its covariance representation as described in section 2.5.1. Next, given an assignment of evidence variables $\mathbf{E}$ and a set of query variables $\mathbf{Q}$, we condition on $\mathbf{E}$ and then marginalize over $\mathbf{V} - \mathbf{E} - \mathbf{Q}$ (the set of irrelevant variables). Formally, we can express a joint Gaussian distribution over sets of variables $\mathbf{X}$ and $\mathbf{Y}$ as follows:

$$P(\mathbf{X}, \mathbf{Y}) \sim N\left((\mu_{\mathbf{X}}, \mu_{\mathbf{Y}}); \begin{bmatrix} \mathbf{\Sigma_{XX}} & \mathbf{\Sigma_{XY}} \\ \mathbf{\Sigma_{YX}} & \mathbf{\Sigma_{YY}} \end{bmatrix}\right). \tag{2.13}$$

Here, we have simply partitioned the mean vector and covariance matrix.

From Mardia et al. (1979), we have the following conditional distribution for $\mathbf{Y}|\mathbf{X}$:

$$P(\mathbf{Y}|\mathbf{X}) \sim N(\mu_{\mathbf{Y}|\mathbf{X}}; \mathbf{\Sigma}_{\mathbf{Y}|\mathbf{X}}) \tag{2.14}$$

where

$$\mu_{\mathbf{Y}|\mathbf{X}} = \mu_{\mathbf{Y}} + \mathbf{\Sigma_{YX}}\mathbf{\Sigma_{XX}}^{-1}(\mathbf{X} - \mu_{\mathbf{X}}), \tag{2.15}$$

$$\mathbf{\Sigma}_{\mathbf{Y}|\mathbf{X}} = \mathbf{\Sigma_{YY}} - \mathbf{\Sigma_{YX}}\mathbf{\Sigma_{XX}}^{-1}\mathbf{\Sigma_{XY}}. \tag{2.16}$$

To marginalize over $\mathbf{Y}$ (starting from Equation 2.13), we simply remove the partitions of the mean vector and covariance matrix which involve $\mathbf{Y}$, resulting in the following distribution:

$$P(\mathbf{X}) \sim N(\mu_{\mathbf{X}}; \mathbf{\Sigma}_{\mathbf{XX}}). \tag{2.17}$$

Observe that both of the preceding operations result in a Gaussian distribution when the original distribution is Gaussian. Thus, after performing inference using conditioning and marginalization, we obtain a multivariate normal distribution over the set of query variables.

2.5.6 Ensembled Linear Gaussian Networks

Ensembling is a common method in machine learning that has been applied to both classification and regression problems. An ensemble consists of a collection of models and makes predictions based on a weighted vote of the component models. Ensembles often peform better in terms of prediction accuracy than a single model (Dietterich 2000).

Two standard techniques for learning ensembles are randomization (Breiman 2001) and bagging (Breiman 1996). These techniques can be used individually or together and result in variability among the models in an ensemble. With randomization, each model is trained on a random subset of variables from the dataset. Bagging is a technique that produces multiple datasets from the original dataset by sampling with replacement. Each model is trained on a different bagged dataset, allowing different models to specialize on different subsets of the data.

Ensembled LGNs (ELGNs) were first thoroughly analyzed by Hellman (2012), extending techniques described by Utz (2010) for ensembling discrete Bayesian networks. Bagging and randomization were employed by both Hellman and Utz. An important observation by Hellman, which is the basis for our work in Section 3.1, is that an ensemble of LGNs is equivalent to a Gaussian mixture model (GMM). This results from the fact that a single LGN is equivalent to a multivariate Gaussian distribution. A GMM, and thus ensemble of LGNs, has a probability density of the form

$$f(x) = \sum_{i=1}^{k} \pi_i N(x; \mu_i, \mathbf{\Sigma}_i)$$

subject to

$$\sum_{i=1}^{k} \pi_i = 1$$

where we have k components, μ_i and $\mathbf{\Sigma}_i$ are the mean and covariance of the ith component, and π_i is the prior probability of the ith component.

Although each component of an ELGN is only capable of representing linear relationships, Hellman demonstrated that Gaussian mixture regression (GMR) (Sung 2004) could be used to perform nonlinear regression with an ELGN. Rather than assigning a constant weight to each component, GMR computes each weight w_i based on the value of the probability density function of component i at the point $\mathbf{X} = \mathbf{x}$ where we want to make a prediction:

$$w_i(\mathbf{x}) = \frac{\pi_i N(\mathbf{x}; \mu_{i\mathbf{X}}, \mathbf{\Sigma}_{i\mathbf{X}})}{\sum_{j=1}^{k} \pi_i N(\mathbf{x}; \mu_{j\mathbf{X}}, \mathbf{\Sigma}_{j\mathbf{X}})}. \tag{2.18}$$

Given a set of query variables $\mathbf{Y}$ and a set of evidence $\mathbf{X} = \mathbf{x}$, the regression function $m(\mathbf{x})$ for an ELGN is thus

$$m(\mathbf{x}) = E[Y|\mathbf{X} = \mathbf{x}] = \sum_{i=1}^{k} w_i(\mathbf{x}) m_i(\mathbf{x}) \tag{2.19}$$

where $m_i(\mathbf{x})$ is the regression function for the ith component.

To account for prediction biases that may be introduced by the use of randomization, Hellman learns an additional constant weight ω_i for each component, resulting in the final regression function for an ELGN:

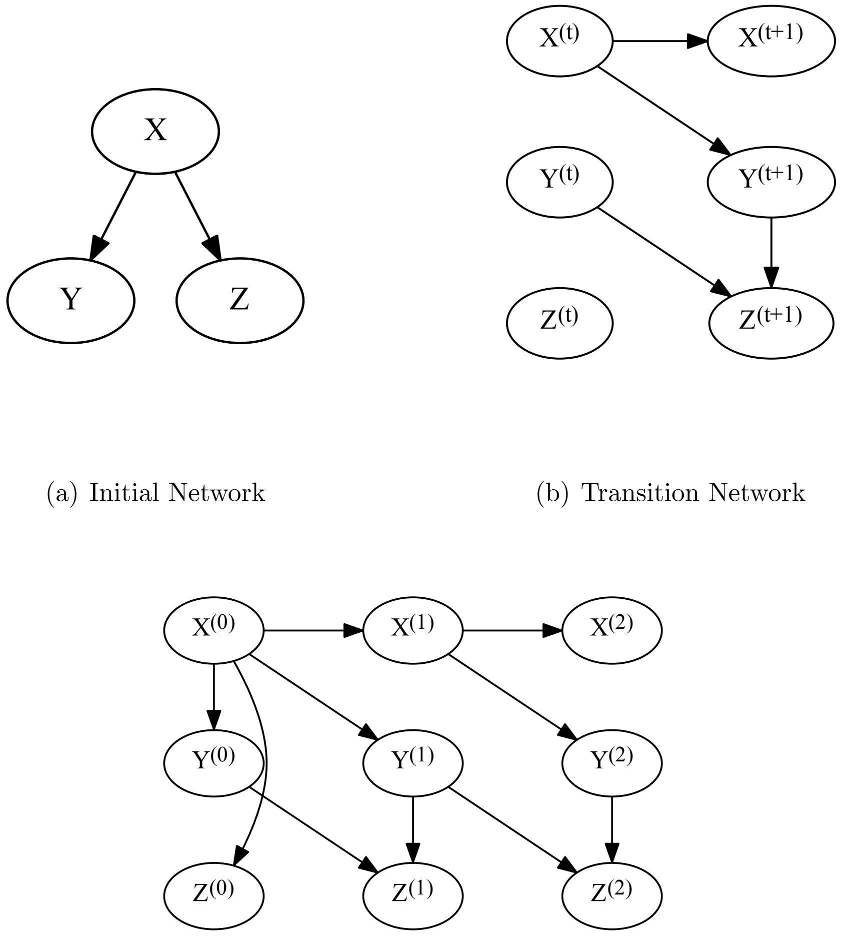
$$m(\mathbf{x}) = E[Y|\mathbf{X} = \mathbf{x}] = \sum_{i=1}^{k} w_i(\mathbf{x}) m_i(\mathbf{x}) \omega_i. \tag{2.20}$$

With randomization, each component of the ensemble is trained on a random subset of variables. Some combinations of variables may be better predictors for the set of query variables than others, so it makes intuitive sense to learn the additional weight ω_i for each component. As is recommended by Hellman et al. (2012), we use ridge regression for learning these weights.

2.6 Dynamic Bayesian Networks

Many domains include a temporal aspect. In such cases, we may be interested in modeling how a system changes over time. Dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1989) encode a joint probability distribution over a set of variables across a number of discrete time slices. In most cases, we make a first-order Markov assumption, which implies that the state of the system depends completely on the state of the system at the previous time step. More formally, for a set of variables $\mathbf{X}$, we have that $Ind(\mathbf{X}^{(t+1)}, \mathbf{X}^{(0:t-1)}|\mathbf{X}^{(t)})$ where $\mathbf{X}^{(T)}$ denotes the set of variables $\mathbf{X}$ at a time slice or sequence of time slices T. Given a set of observations of $\mathbf{X}$ at time t, the state of the system for $T > t$ is independent of the state of the system at $T < t$.

A DBN is specified by two components: a network representing the initial distribution over $\mathbf{X}$ and a transition network representing the conditional distribution of $\mathbf{X}^{(t+1)}|\mathbf{X}^{(t)}$. To represent the distribution of $\mathbf{X}$ over multiple time slices, we *unroll* the network. To perform unrolling, we start with the initial network, then apply the transition network for as many time steps as we wish to model. Figure 2.3 shows an example of a DBN consisting of variables $\mathbf{X} = (X, Y, Z)$. The unrolled network for two timesteps is shown in Figure 2.3(c).



(a) Initial Network

(b) Transition Network

(c) Unrolled Network

Figure 2.3: *The initial and transition networks specifying a DBN over three variables, and the unrolled network over three timesteps.*

Chapter 3

Improving Predictions for Nonlinear and Spatial Data

As we have seen, ensembled linear Gaussian networks have the ability to perform nonlinear regression when Gaussian mixture regression is used to compute component weights during inference. Unfortunately, when traditional bagging techniques are used to learn ELGNs, they are unable to accurately model nonlinear functions. In Section 3.1, we consider why this is the case and offer a solution that allows us to better model local relationships within the data.

We have also seen how dynamic Bayesian networks can be used to model temporal correlations within a dataset and the evolution of a system over time. In Section 3.2, we demonstrate a new technique for taking into account spatial correlations when performing inference.

3.1 Weighted Bagging

In this section we aim to improve the prediction accuracy of ELGNs for datasets that exhibit nonlinear relationships between variables. To accomplish this, we introduce the use of weighted bagging during the learning process.

By applying the ELGN framework to a given regression problem, we are assuming that our dataset $D = (\mathbf{x}_1, ..., \mathbf{x}_m)$ with m instances can be modeled with

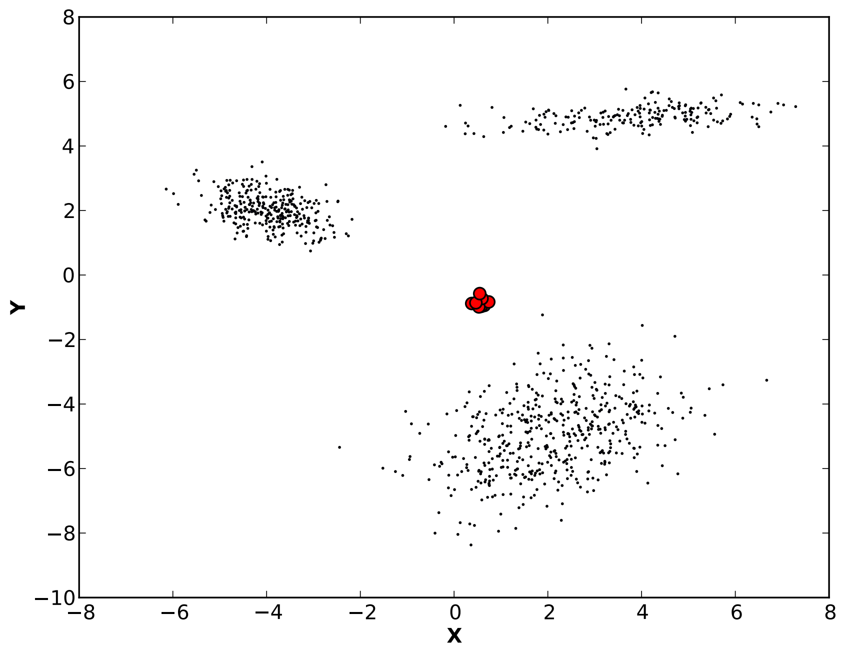


Figure 3.1: *A dataset generated from a three-component GMM and the means of a 10-network ELGN learned with traditional bagging.*

sufficient accuracy by a Gaussian mixture model. Thus we assume that each instance $\mathbf{x}_i \in D$ is an independent sample produced by some underlying mixture of Gaussians. When learning an ELGN, we wish to capture information about the individual components of the generating distribution, including their means, covariances, and prior probabilities. Unfortunately, when traditional bagging is used, we cannot distinguish between these different components, resulting in a model that is incapable of truly representing the underlying multicomponent distribution.

An an example, consider the dataset shown in Figure 3.1. The data was generated by a mixture of bivariate Gaussian distributions with three components. If we apply traditional bagging to learn an ensemble of LGNs, each component network is trained on a random subset of the entire dataset, without regard for the clusters of datapoints that represent the components of the underlying distribution. In the plot, the filled circles represent the mean of the equivalent bivariate normal distribution for each LGN in the ensemble. Observe that these means tend toward the global mean of the dataset rather than fitting to the clusters.

In order to accurately model a mixture of Gaussians, we use weighted bagging to generate bagged datasets that are representative of the underlying distribution's individual components. A form of weighted bagging was first employed by Shieh and Kamm (2009), where kernel density estimation was used to assign weights to training examples in order to reduce the impact of outliers in training one-class classifiers. This work was applied to classification tasks by Biggio et al. (2011) and Seguí et al. (2010). The context here is rather different, as we aim to faithfully model the components of the generating distribution to perform regression rather than eliminate noise in classification problems.

In our method, we first use expectation maximization (EM) to estimate the parameters of a k-component mixture of Gaussians (Dempster et al. 1977; Bilmes et al. 1998). When the algorithm terminates, we have a m by k weight matrix $\mathbf{w}$ where each weight w_{ij} corresponds to the probability that instance $\mathbf{x}_i$ was generated by the jth multivariate Gaussian component. In addition to data weights, EM also returns a set of weights $\pi = (\pi_1, ..., \pi_k)$ where each π_j is the prior probability of the jth component. In estimating the parameters of the mixture, the mean and covariance for each component are computed, but these are not needed to perform weighted bagging and can be discarded.

Once we have estimated the data weights and component priors, we are ready to learn an ensemble of LGNs. To train each component of the ensemble, we randomly select a column $\mathbf{w}_{:j}$ from the weight matrix $\mathbf{w}$ with probability π_j for $1 \leq j \leq k$. Next, we perform weighted sampling with replacement on the original dataset using $\mathbf{w}_{:j}$ to obtain a bagged dataset which is representative of the jth component of the generating distribution. The procedure for learning an ELGN with weighted bagging is summarized in Algorithm 3.1.

Algorithm 3.1: Learning an ELGN with weighted bagging

```
Input: $data$, $k$ = number of clusters, $nNetworks$ = number of networks
       in the ensemble
Output: $ensemble$: an ELGN
$ensemble \leftarrow$ Empty ensemble
for $i = 1, ..., nNetworks$ do
    $means, covariances, \mathbf{w}, \pi \leftarrow$ gaussianEM($data$, $k$)
    $weights \leftarrow$ choose column $\mathbf{w}_{:j}$ with probability $\pi_j$
    $bag \leftarrow$ sampleWithReplacement($data$, $weights$)
    $network \leftarrow$ learnLGN($bag$)
    $ensemble$.addModel($network$)
end
```

The effect of learning with weighted bagging is to better model local relationships within the data. By running the EM algorithm, we find clusters that can be modeled well by Gaussian distributions. Then, for each cluster, we learn one or more LGNs, each of which is equivalent to a Gaussian distribution. Weighted bagging allows the components of the ensemble to spread throughout the data, each focusing its learning efforts on a particular cluster. Besides spreading our models throughout the data, weighted bagging allows us to implicitly represent the prior probability of each component in the underlying distribution. Each time a model in the ensemble is learned, we select a column of weights $\mathbf{w}_{:j}$ from the weight matrix with the corresponding prior probability π_j. Thus, more networks are learned around clusters for which the dataset contains a larger number of instances.

Figure 3.2 again shows the dataset from Figure 3.1, but this time we have learned a 10-network ELGN using weighted bagging. The filled circles represent the means of the equivalent Gaussian distributions for the networks in the ELGN. Observe that rather than tending toward the global mean of the data, the means of the networks are close to one of the cluster centers.

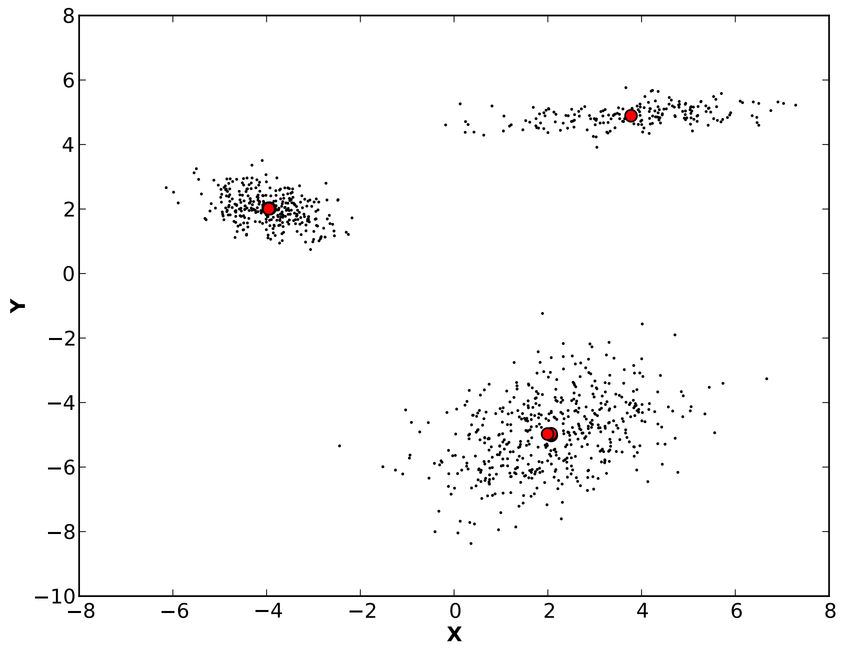


Figure 3.2: *A dataset generated from a three-component GMM and the means of a 10-network ELGN learned with weighted bagging.*

In addition to modeling data generated from a GMM, weighted bagging allows us to accurately model arbitrary nonlinear functions with Gaussian noise. Figure 3.3 shows a dataset generated from a noisy sine function. Values for X were sampled uniformly from the interval (0, 10), then values for Y were computed as $Y = 10\sin(X) + \epsilon$ where $\epsilon \sim N(0; \sigma = 2)$. An ELGN consisting of 32 networks was then learned with EM estimating $k = 16$ cluster centers. The mean of each network is plotted as a filled circle. We can see that even though this dataset was generated from a noisy sine function rather than a mixture of Gaussians, our networks are still spread throughout the data, allowing us to accurately model local relationships. Using Gaussian mixture regression to predict Y given values of X, we obtain the regression curve shown in the plot. From this and the previous example, we can see that learning ELGNs with weighted bagging can be a powerful means of modeling nonlinear functions. In Chapter 4, we conduct formal experiments to demonstrate this claim.

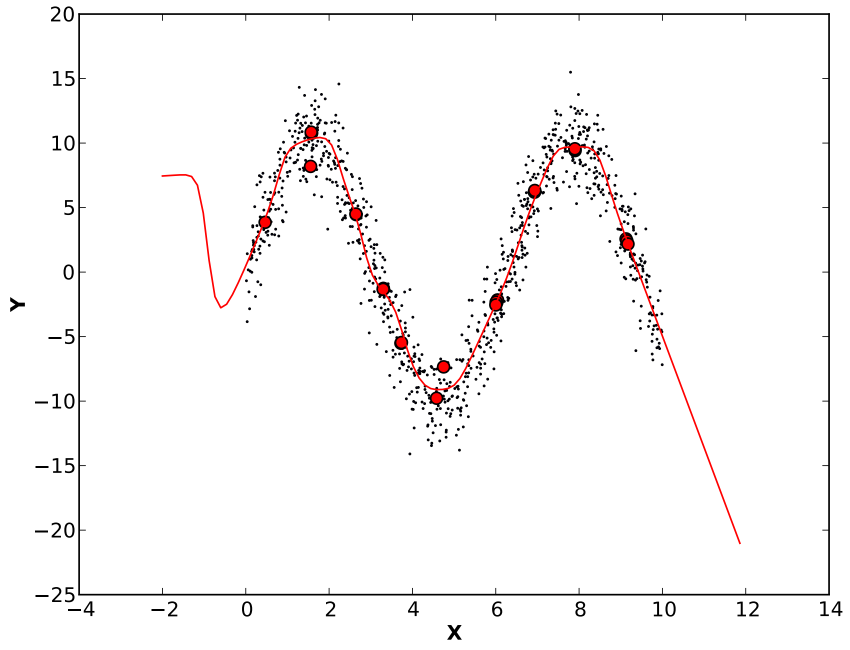


Figure 3.3: *ELGN trained with weighted bagging on a dataset generated from a sine function with added Gaussian noise. The regression curve is shown for predicting Y given X.*

3.2 Spatial Inference

In Section 2.6, we saw that dynamic Bayesian networks can be used to model the evolution of a system over time. Since many domains also contain a spatial aspect, we wish to develop a framework that incorporates spatial correlations between variables. In contrast to time, space can be multidimensional and influence can flow in multiple directions. Thus, modeling space is not straightforward. Work has been done in using Bayesian networks to model spatiotemporal relationships by Dereszynski and Dietterich (2011) and Tucker et al. (2005), but these methods have disadvantages. The method presented by Dereszynski and Dietterich (2011) learns an explicit network structure over all pertinent locations, and is therefore not scalable for domains with many locations. The method presented by Tucker et al. (2005) requires the use of a cardinal (gridded) coordinate system, and cannot model correlations between locations at arbitrary points in space.

In this section, we present a framework for performing inference that takes into account spatial correlations between variables. Our method makes no assumptions about the choice of coordinate system and can incorporate information from arbitrary locations at arbitrary distances when making predictions. In addition, our method can be used to make predictions at a target location when observations are made only at other locations and not the target location itself.

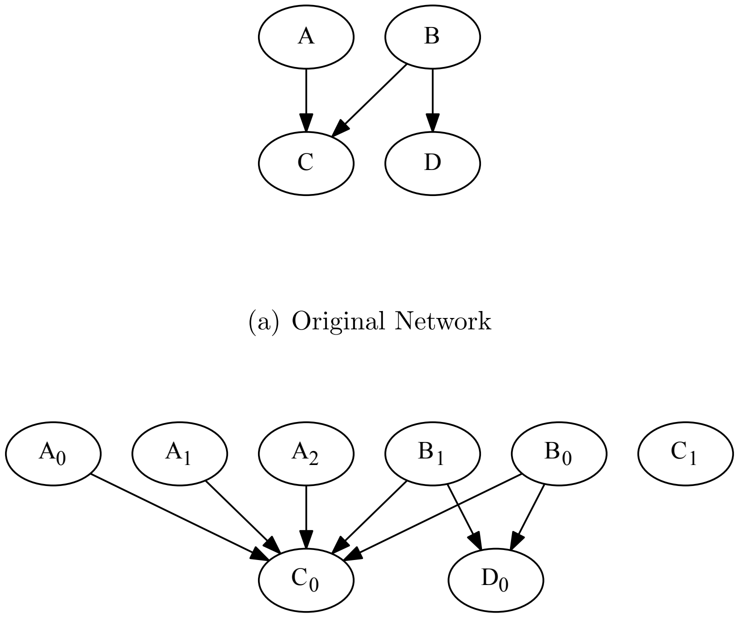
In domains with a spatial aspect, variables can possess correlations with variables at other points in space. Taking meteorological data as an example, it is reasonable to expect that variables such as temperature, pressure, and reflectivity are highly correlated at nearby points in space. As the distance between two points increases, we would expect the correlation to decrease. In a Bayesian network, direct correlations between variables are represented by a directed edge from one variable to another. To account for these spatial correlations, our method adds nodes corresponding to observations to the network prior to inference. To weight observations from different locations, we employ kernel functions.

3.2.1 A Graphical Perspective of Spatial Inference

To provide an intuitive understanding of how we perform spatial inference, we will take this section to show how the network is modified given observations of variables at various points in space. The next section provides the mathematical details of our method.

Suppose that we have learned a LGN over a set of variables $\mathbf{V}$ and now wish to perform inference for a set of query variables $\mathbf{Q}$ at a spatial location l_0 given a set of evidence $\mathbf{E}$. Each piece of evidence $E \in \mathbf{E}$ must correspond to a variable $V \in \mathbf{V}$, but may be from a location other than l_0. For each piece of evidence E corresponding to a variable V, we add a node V' to the network structure as well as a directed edge from V' to the children of V. Nodes added in this way have a mean and variance equal to that of the corresponding variable.

As an example, consider the network shown in Figure 3.4(a). Suppose we wish to perform inference at location l_0 and that our set of evidence consists of observations of A at locations l_0, l_1, and l_2; B at location l_1; and C at location l_1. If we add nodes and edges corresponding to the set of evidence as described above, we obtain the network structure shown in Figure 3.4(b). The subscripts for variables shown in the resulting network correspond to the locations at which the variables were observed. Note that because C has no children in the original network, C_1 is unconnected and therefore has no impact during inference. Also note that although A is observed at l_0, no additional node is added for this piece of evidence because A_0 is already in the network. Of course, we will still condition on this evidence when performing inference.



(a) Original Network

(b) Network with Spatial Evidence

Figure 3.4: *Adding evidence nodes to a network to perform spatial inference.*

Now that we have an augmented network structure that incorporates observations from other points in space, we can simply use the algorithm from (Shacter and Kenley) (described in section 2.5.1) to convert the network to a multivariate Gaussian distribution. We then condition on the set of evidence **E** and marginalize over any irrelevant variables to obtain predictions for the query set **Q**.

3.2.2 Mathematical Considerations of Spatial Inference

As we have seen previously, the CPD for a node Y with a set of parents $\mathbf{X} = (X_1, ..., X_k)$ takes the following form in a LGN:

$$P(Y|\mathbf{X}) = N\left(\mu_Y; {\sigma_Y}^2\right) \tag{3.1}$$

where

$$\mu_Y = \beta_0 + \beta^T \mathbf{X} = \beta_0 + \sum_{i=1}^{k} \beta_i X_i. \tag{3.2}$$

For each parent variable X_i for which we have observations at one or more points in space, we modify the equation for μ_Y so that

$$\mu_Y = \beta_0 + \sum_{i=1}^{k} \beta_i \tilde{X}_i \tag{3.3}$$

where

$$\tilde{X}_i = \frac{\sum\limits_{j=0}^{n} X_{ij} K(X_{i0}, X_{ij})}{\sum\limits_{j=0}^{n} K(X_{i0}, X_{ij})}. \tag{3.4}$$

In Equation (3.4), X_{i0} is defined as variable X_i at the location where inference is being performed, X_{ij} refers to an observation of variable X_i at location j, and K is a kernel function whose value depends on the spatial locations of the two input variables. The above equation for $\tilde{X}_i$ takes the form of what is commonly known as the Nadaraya-Watson kernel estimator (Nadaraya 1964), (Watson 1964). This is essentially a weighted average of observed values of X_i at various points in space.

Expanding (3.3) gives us the following:

$$\mu_Y = \beta_0 + \sum_{i=1}^{k} \sum_{j=0}^{n_i} \beta_{ij} X_{ij} \tag{3.5}$$

where

$$\beta_{ij} = \frac{K(X_{i0}, X_{ij})}{\sum\limits_{l=0}^{n_i} K(X_{i0}, X_{il})} \tag{3.6}$$

and n_i is the number of observations at other points for X_i. Collecting the β_{ij} and X_{ij} terms into vectors $\tilde{\beta}$ and $\tilde{\mathbf{X}}$, respectively, we have

$$\mu_Y = \beta_0 + \tilde{\beta}^T \tilde{\mathbf{X}} \tag{3.7}$$

which is of the form required for the mean of a variable in a LGN.

It is worth noting that Equation 3.4 is closely related to the concept of kernel regression. In kernel regression, we make a prediction by taking a weighted sum of values, where the weights are computed with a kernel function. In our case, the values in the weighted sum correspond to observations at spatial locations. As an alternative to the simple kernel estimator used here, locally weighted regression (LWR) (Cleveland 1979; Atkeson et al. 1999) could offer the ability to model complex nonlinear relationships between locations. However, LWR is a nonparametric method that requires use of the training data to perform regression. By using Equation 3.4 with observations to make predictions, we avoid the need to store and reference the training data.

Just as we added a term corresponding to each observation in our computation of μ_Y, we must also add entries to the covariance matrix $\mathbf{\Sigma_{XX}}$ for each observation. Given the independence properties of Bayesian networks and the algorithm to convert a network to a multivariate Gaussian distribution, this turns out to be straightforward. In the previous section, we only added edges between evidence nodes and their children. Thus, all evidence nodes are conditionally independent of the ancestors of their children until the children are conditioned upon. The corresponding entries in the augmented version of $\mathbf{\Sigma_{XX}}$, which we refer to as $\mathbf{\Sigma_{\tilde{X}\tilde{X}}}$, are thus zero. We set the variance for an evidence node corresponding to a variable V equal to ${\sigma_V}^2$. After taking into account observations at other points, the variance of Y with parents $\tilde{\mathbf{X}}$ is then

$${\sigma_Y}^2 = \sigma^2 + \tilde{\beta}^T \Sigma_{\tilde{X}\tilde{X}} \tilde{\beta}. \tag{3.8}$$

Chapter 4

Empirical Evaluation of Weighted Bagging and Spatial Inference

As discussed in Section 3.1, the use of weighted bagging to learn ELGNs should allow us to accurately model arbitrary nonlinear functions. In this chapter, we empirically demonstrate the efficacy of weighted bagging in achieving this end. We begin by analyzing the performance of ELGN on a synthetic dataset with clear nonlinear relationships, then move on to a very different real-world dataset obtained from the UCI Machine Learning Repository (Bache and Lichman 2013). For each experiment, we compare ELGN with weighted bagging to Hellman's ELGN (which uses traditional bagging) and to least-squares linear regression or persistence. In all cases, we find that ELGN with weighted bagging is able to perform as well as or better than the other models. In addition to evaluating ELGN with weighted bagging, we also assess spatial inference in the weather forecasting domain.

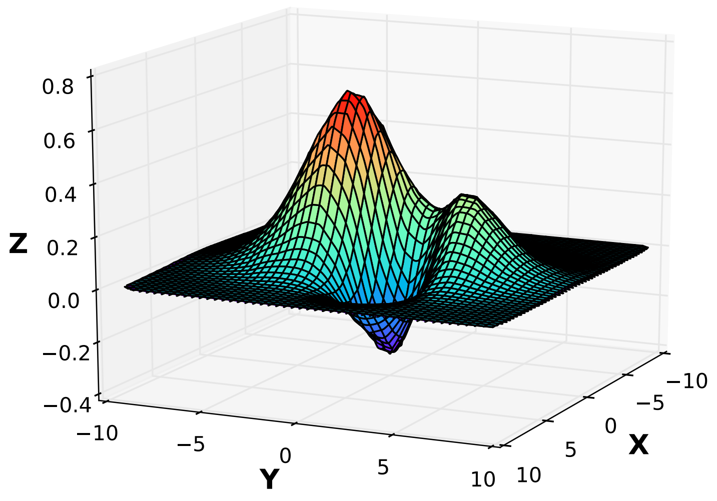
4.1 The Curvy Dataset

Our first experiment is intended to convince the reader that ELGNs are indeed capable of accurately modeling arbitrary nonlinear functions. The dataset for this experiment, which we call the Curvy dataset, is a set of instances that we artificially generated and consists of three variables, X, Y, and Z. To produce the dataset, we uniformly sampled X and Y in the range $(-10, 10)$, then computed

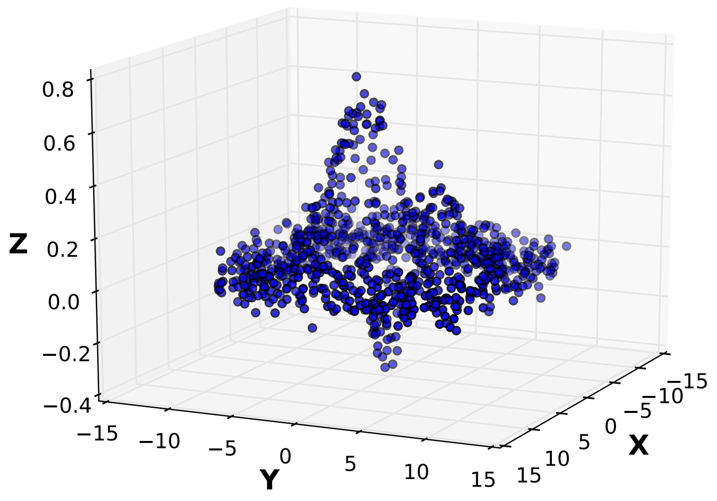
$$f(X,Y) = e^{-\frac{X^2}{10}-\frac{Y^2}{20}} - e^{-\frac{(X-1)^2}{10}-\frac{(Y-1)^2}{5}}. \tag{4.1}$$

We then added Gaussian noise to produce Z so that $Z = f(X,Y) + \epsilon$ where $\epsilon \sim N(0; \sigma = 0.05)$. The plot of $f(X,Y)$ is shown in Figure 4.1(a). The Curvy dataset consists of 2000 instances. In our experiments, we randomly select 1000 instances from Curvy for training and use the remaining instances for testing. For ELGN, we vary the number of clusters found by EM (k), the number of networks per ensemble ($nNetworks$), and the number of variables per network ($nVariables$). For Hellman's ELGN, we vary $nNetworks$ and $nVariables$. Each experiment is run 30 times and performance is evaluated using root mean squared error (RMSE).

Figure 4.2 shows the performance of ELGN on the Curvy dataset. RMSE, indicated by color, is plotted against the number of networks per ensemble and the number of clusters. Figure 4.2(a) shows results for two variables per network while 4.2(b) shows results for three variables per network. In both plots, we see that performance is worst when k or $nNetworks$ is small and improves as both k and $nNetworks$ increase. For $nNetworks < k$, we expect poor performance since there is not a network learned for every cluster found; therefore, it makes little sense to set $nNetworks < k$. When $k = 1$, we are only able to represent linear relationships, so performance is poor on the Curvy dataset. As we increase k, we have more flexibility to model nonlinear relationships, so performance improves. When k becomes very large, we expect performance to worsen due to overfitting, but this has not occurred within the range of parameters that we considered. Increasing $nNetworks$ has the effect of increasing the number of

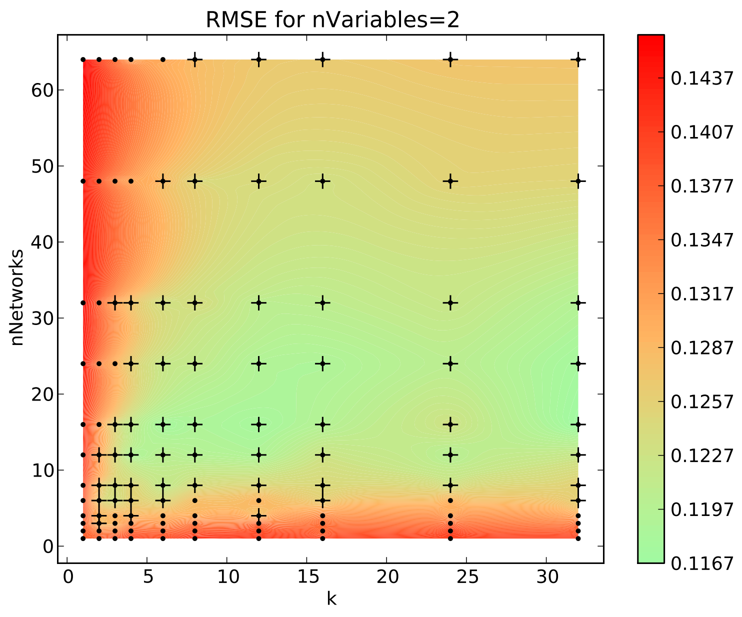


(a) Underlying function of the Curvy dataset



(b) The Curvy training data

Figure 4.1: *The Curvy dataset*



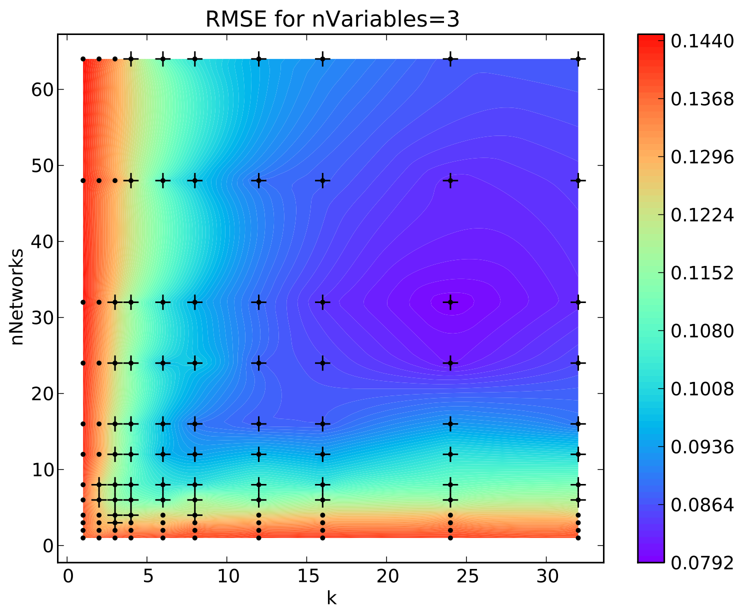


Figure 4.2: *RMSE for the Curvy dataset as the number of clusters and number of networks per ensemble are varied. Dots in the plot correspond to parameter combinations that were tested. Crosses show statistically significant parameter settings.*

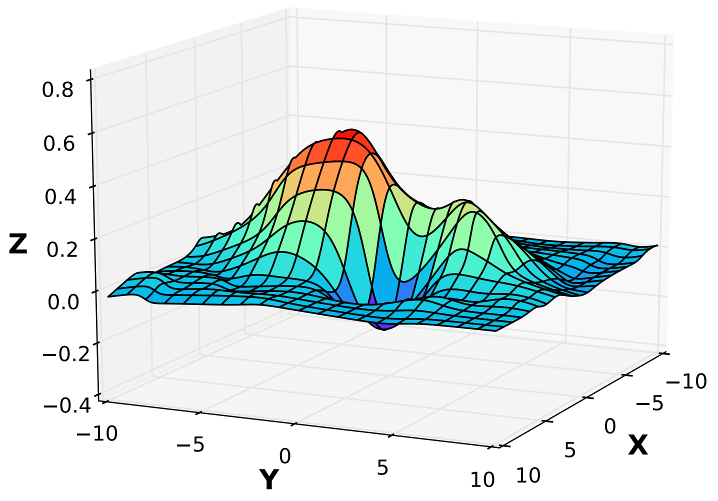


Figure 4.3: *Regression surface for the Curvy dataset produced by ELGN with weighted bagging (k=24, nNetworks=32).*

models learned around each cluster, which we expect to produce a smoothing effect and help to reduce overfitting.

In Figure 4.3 we show a plot of the regression surface produced by an ELGN with $k = 24$ and $nNetworks = 32$. By visual inspection, it appears that ELGN provides a relatively accurate model for $f(X, Y)$, the underlying function that produced our dataset.

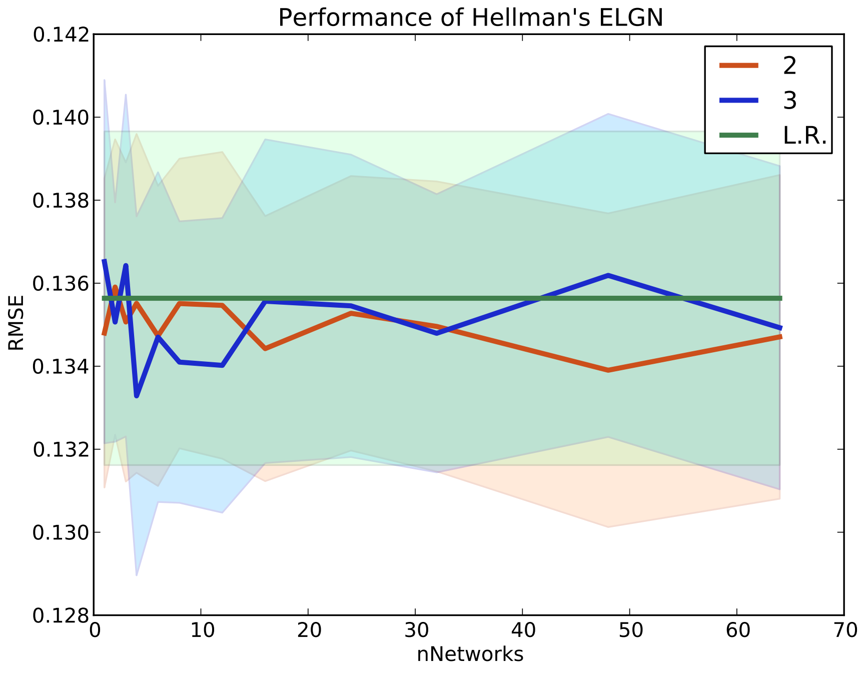


Figure 4.4: *RMSE of Hellman's ELGN on the Curvy dataset as the number of networks per ensemble is varied. Each curve corresponds to a different value for* $nVariables$. *The performance of linear regression is also shown for comparision. Standard error is represented by the shaded areas.*

Figure 4.4 shows the performance of Hellman's ELGN on the Curvy dataset. Since traditional bagging is used, the only parameters that we vary are $nNetworks$ and $nVariables$. To provide a comparison, we also used linear regression to model the Curvy dataset. We see from the plot that Hellman's ELGN performs similarly to linear regression, indicating that ELGN has difficulty capturing nonlinear relationships when traditional bagging is used.

In contrast to Hellman's ELGN, Figure 4.2 provides evidence that ELGN with weighted bagging has the ability to capture nonlinear relationships. To formally compare the use of traditional and weighted bagging for learning ELGN, we use a two-sample one-tailed t-test with a significance level of $\alpha = 0.05$. We use Bonferroni correction to reduce the significance level in order to account for statistical anomalies that could result from testing multiple sets of parameters. With this method, we find that ELGN with weighted bagging performs significantly better than Hellman's ELGN for most parameter settings when $k > 2$ and $nNetworks > 4$. Significant parameter settings are marked with a cross in Figure 4.2.

4.2 The Wine Quality Dataset

In our next experiment, we use ELGN to learn on the Wine Quality dataset (Cortez et al. 2009) obtained from the UCI Repository. We chose to focus on the red wine portion of the dataset, which contains 1599 total instances. Each instance consists of eleven real-valued attributes describing physical characteristics of a particular wine and a quality attribute, which is a score in the range 0.0 to 10.0. We wish to predict the quality of a wine given its physical attributes. As in the previous experiment, we compare ELGN with weighted bagging to Hellman's ELGN and linear regression. In each of 30 runs, we train on 1000 randomly selected instances and use the remainder for testing.

Figure 4.5 shows results for ELGN with weighted bagging. As we expect for any dataset, we have worse performance in the lower right corner of the plot, when $nNetworks < k$. However, we see fairly consistent performance across the rest of each plot. This seems to indicate that the dataset contains nearly linear relationships between variables, as increasing k does not provide any additional power to model the data. For fixed values of k and $nNetworks$, we see that performance generally improves as the number of variables per network is increased. This is similar to what we observed with the Curvy dataset.

Results for Hellman's ELGN and linear regression are shown in Figure 4.6. Both performed similarly to ELGN with weighted bagging, supporting the hypothesis that the relationships in the dataset are mostly of a linear nature. Like ELGN with weighted bagging, we find that the performance of Hellman's ELGN improves as the number of variables per network is increased.

For the Wine Quality dataset, ELGN with weighted bagging offers no advantage in terms of prediction accuracy over Hellman's ELGN or linear regression. If the relationships within the data are truly linear, we cannot expect to have significantly better performance than a linear model. However, the results of this experiment are still important because they demonstrate the robustness of ELGN. Provided that the number of networks per ensemble is sufficiently large, it seems that ELGN is resistant to overfitting as k is increased. Thus, ELGN with weighted bagging can provide an accurate model for linear and nonlinear datasets alike.

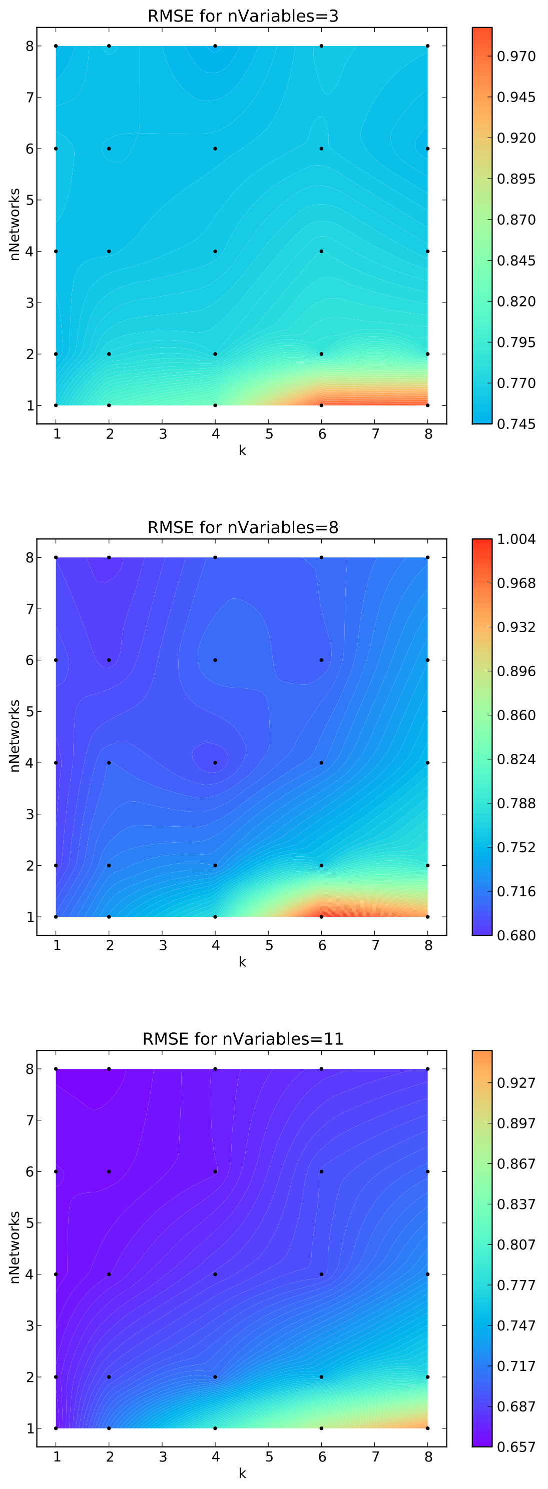


Figure 4.5: *RMSE for the Wine Quality dataset as the number of clusters and number of networks per ensemble are varied.*

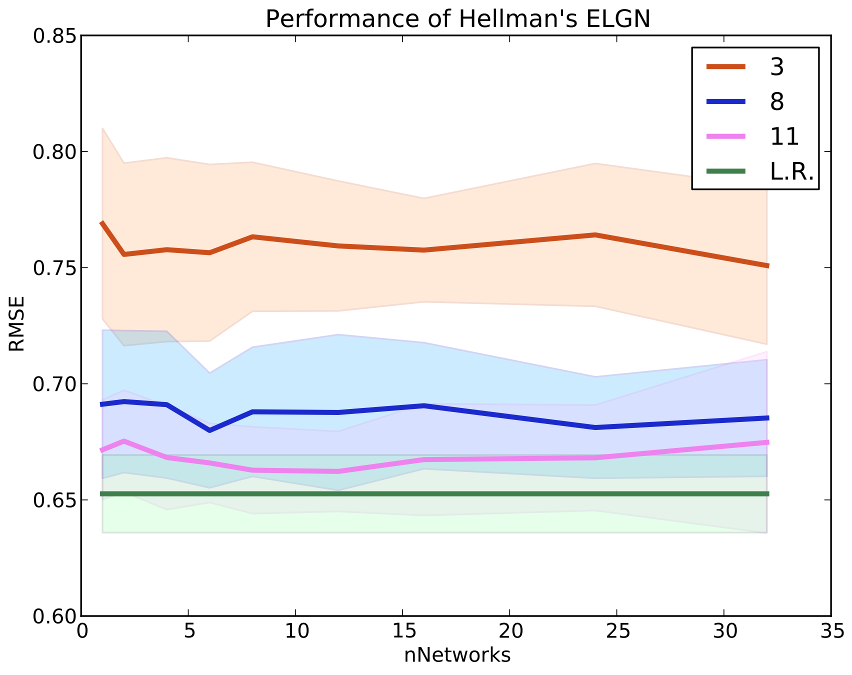


Figure 4.6: *RMSE of Hellman's ELGN on the Wine Quality dataset as the number of networks per ensemble is varied. Each curve corresponds to a different value for* $nVariables$. *The performance of linear regression is also shown for comparision.*

4.3 The Weather Research and Forecasting (WRF) Dataset

Weather forecasting is a complex domain that contains both spatial and temporal relationships among variables. The ability to predict reflectivity is crucial in helping meteorologists determine how a storm will develop over time. This can allow meteorologists to warn the public about dangerous storms, preventing property damage and the loss of life. In this section, we apply ELGN to predict reflectivity for data obtained from the Center for Analysis and Prediction of Storms (CAPS) (Clark et al. 2012). The dataset is from May 20, 2013 and is centered over the state of Oklahoma. Storms that day produced several tornadoes, including a destructive EF5 tornado that passed through the city of Moore, Oklahoma. The entire dataset consists of observations of approximately 200 variables at time slices spaced five minutes apart over the course of 24 hours. The data correspond to gridded spatial locations 500 meters apart.

Our task for the following experiments is to predict maximum composite reflectivity (*CREFD_MAX*) for the next time slice given observations of variables at the current time slice. Since we must model temporal relationships, we learn a dynamic ensemble of LGNs (DELGN). Dynamic Bayesian networks, discussed in Section 2.6, can be trivially combined with LGNs. Since the dataset contains a large number of variables, we chose eight in addition to *CREFD_MAX* to train our networks. To aid us in selecting variables, we examined partial and Spearman rank correlations with *CREFD_MAX* at the following time step and chose variables that were well-correlated. Selected variables with brief descriptions are shown in Table 4.1.

Variable	**Description**
CREFD_MAX	Maximum derived composite radar reflectivity
REFD_MAX	Maximum derived radar reflectivity
REFDM10C_MAX	Maximum derived radar reflectivity at -10C
REFL_10CM	Radar reflectivity with lambda=10cm
W_DN_MAX	Maximum downdraft wind speed
W_UP_MAX	Maximum updraft wind speed
GRPL05_MAX	Graupel mass 0-5 km above ground
PRATE	Precipitation rate
LTG3_MAX	Maximum lightning threat

Table 4.1: *Variables from the WRF dataset used to train DELGN.*

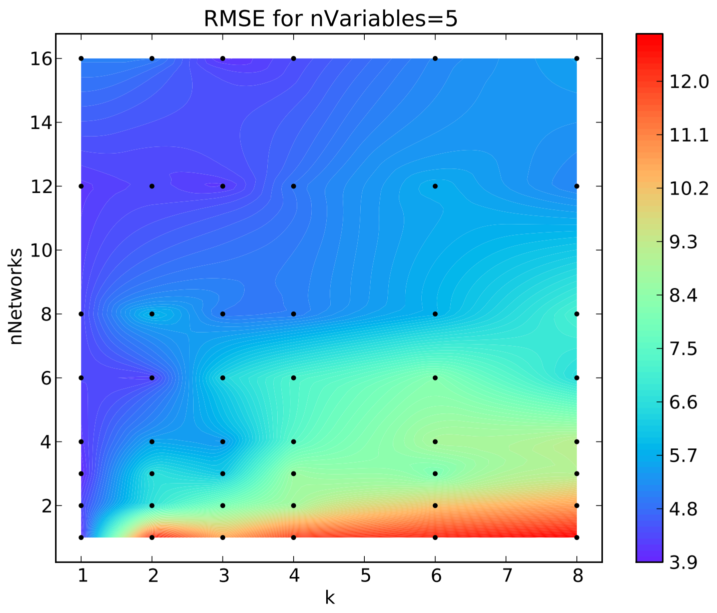
Due to the large size of the dataset, we created a training set by sampling from multiple time slices throughout the day at random spatial locations. Reflectivity is highly skewed in the original dataset, with approximately 90% of the instances having a *CREFD_MAX* value at zero. For this reason, we used a biased sampling technique to produce the training set so that approximately 50% of the instances have *CREFD_MAX* > 0. In total, the training set contains 2000 paired instances (where each pair consists of instances from neighboring time slices).

4.3.1 Evaluation of DELGN with Weighted Bagging

In this experiment, we evaluate the performance of DELGN with weighted bagging in the WRF domain. Our test set consists of 1000 paired instances sampled from time slices at 20:10 GMT and 20:15 GMT at random spatial locations. We compare DELGN with weighted bagging to a persistence predictor and Hellman's DELGN. The persistence predictor simply uses the value of $CREFD\_MAX$ at the current timestep as its prediction for $CREFD\_MAX$ at the next timestep. In Figure 4.7 we can see that the performance of DELGN generally seems to improve as we increase the number of variables per network. Interestingly, it also appears that we are modeling somewhat nonlinear relationships within the data. If we look specifically at the plot for $nVariables = 8$, we can see that some of the best performances occur when k is set to two, three, or four. This is an indication that a more flexible nonlinear model is capable of providing more accurate predictions than a strictly linear model.

The results of Hellman's DELGN and a persistence predictor are shown in Figure 4.8. From the plot, it appears that DELGN with traditional bagging performs similarly to or slightly better than persistence. To compare DELGN with weighted bagging to Hellman's DELGN and persistence, we again use a two-sample one-tailed t-test with an initial significance level $\alpha = 0.05$ and Bonferroni correction. We find that DELGN with weighted bagging performs significantly better than persistence for five of our parameter settings, but not significantly better than Hellman's DELGN. Statisitically significant parameter settings are marked with a cross in Figure 4.7.

To provide an intuitive idea of how DELGN with weighted bagging performs, Figure 4.9 shows plots of composite reflectivity at 20:15 GMT. Figure 4.9(a) shows the actual reflectivity at that time while Figure 4.9(b) shows the predicted reflectivity for DELGN with weighted bagging given observations for the previous time step. By visual inspection, DELGN appears to provide reasonable predictions for reflectivity. One important thing to note in this example is that DELGN rarely provides predictions of zero reflectivity. At locations where



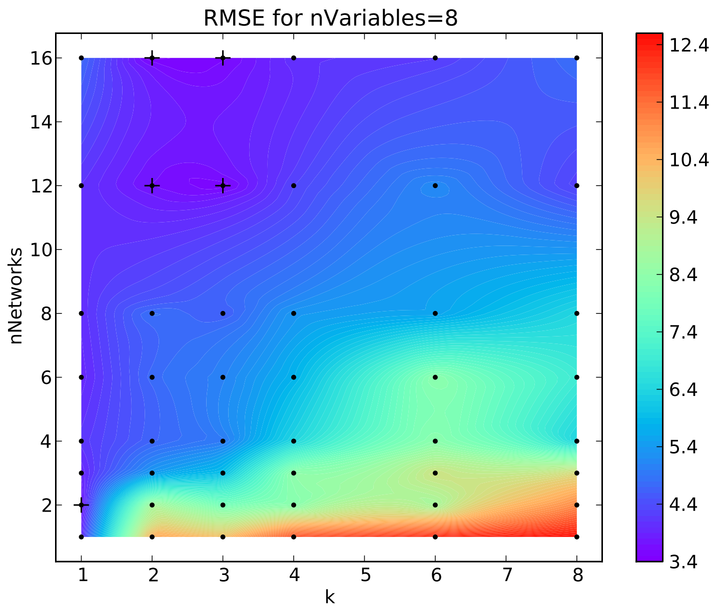


Figure 4.7: *RMSE for the WRF dataset as the number of clusters and number of networks per ensemble are varied. Crosses show parameter settings that are statistically significant compared to persistence.*

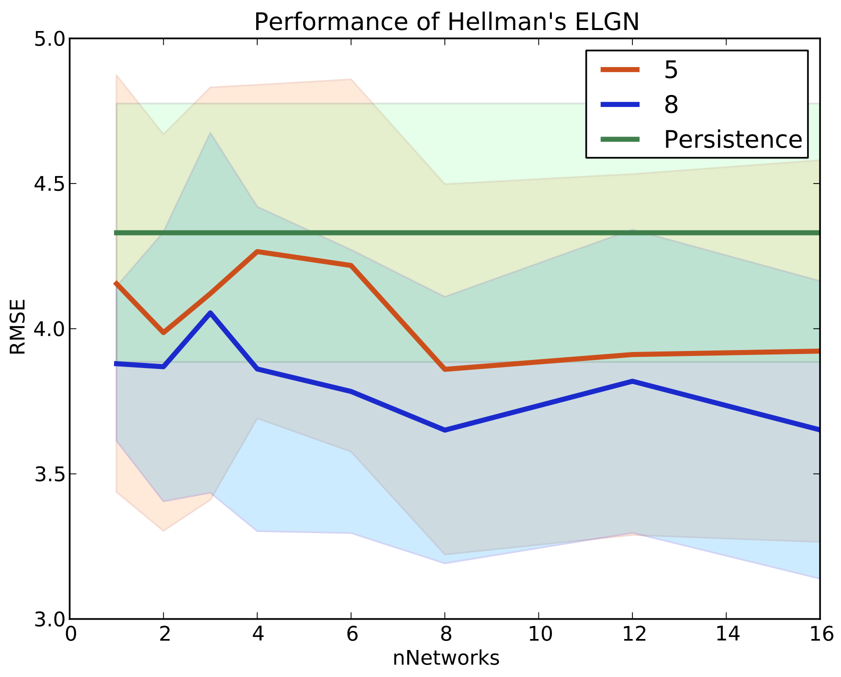
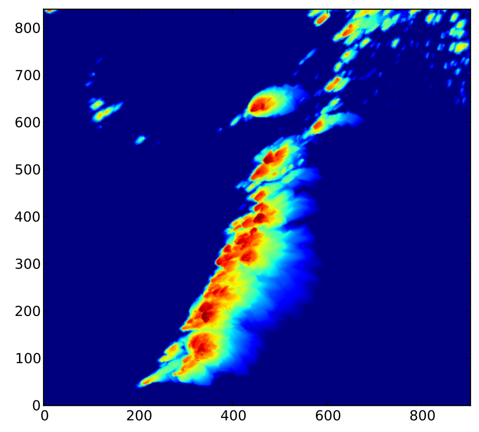
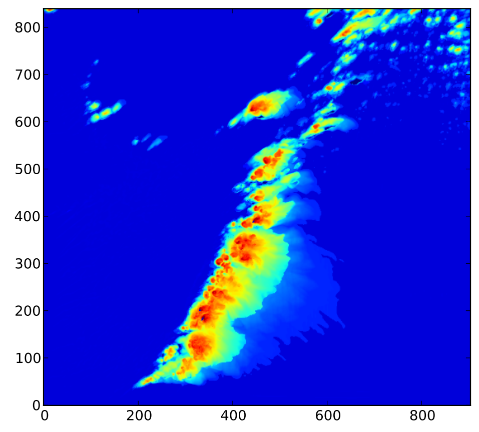


Figure 4.8: *RMSE of Hellman's DELGN on the WRF dataset as the number of networks per ensemble is varied. Each curve corresponds to a different value for* $nVariables$*. The performance of persistence is also shown for comparision.*

the actual reflectivity is zero, DELGN predicts small nonzero values. For future work, a two-part prediction algorithm could apply a classifier to determine whether the reflectivity is nonzero, and if so, apply DELGN; otherwise, it would predict zero.



(a) Actual Reflectivity



(b) Predicted Reflectivity

Figure 4.9: *DELGN's prediction of CREFD_MAX for k=4 and nNetworks=8.*

4.3.2 Evaluation of Spatial Inference

In this experiment, we investigate whether the use of spatial inference improves prediction accuracy in the WRF domain. As in the previous experiment, we learn a DELGN using weighted bagging on the WRF training set described earlier. Given observations at 20:20 GMT, we wish to predict composite reflectivity at 20:25 GMT at gridded locations spaced 8 km apart. We make predictions using both normal and spatial inference and compare the two to determine if spatial inference offers an advantage.

Given our spatial inference framework described in Section 3.2, we must select a kernel function to weight evidence from nearby locations. For this experiment, we choose the squared exponential kernel, which takes the following form (Rasmussen 2006):

$$K(d) = \exp\left(-\frac{d^2}{2l^2}\right)$$

where d is the distance between two locations and l is the *characteristic length-scale*, a user-defined parameter. We set l to 500 m for this experiment, which is the distance between adjacent grid points in the WRF dataset. This choice for l seems reasonable since reflectivity values can be substantially different between locations a few kilometers apart. Due to the computational resources required, we use only the observations at the target location and four adjacent grid points to perform spatial inference.

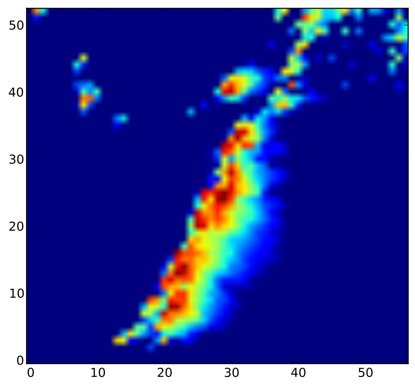
For this experiment, we select the four sets of parameters with the best performance from the previous experiment for training DELGN with weighted bagging. Figure 4.10 shows the average predicted reflectivity at grid points spaced 8 km apart for 20:25 GMT when DELGN was trained with parameters $k = 2$, $nNetworks = 16$, and $nVariables = 8$. By visual inspection, there does

k	**nNetworks**	**nVariables**	**RMSE (Normal)**	**RMSE (Spatial)**	**Statistically significant?**
2	16	8	4.27	4.15	No
3	12	8	4.47	4.68	No
3	16	8	4.26	4.33	No
2	12	8	4.88	4.17	No

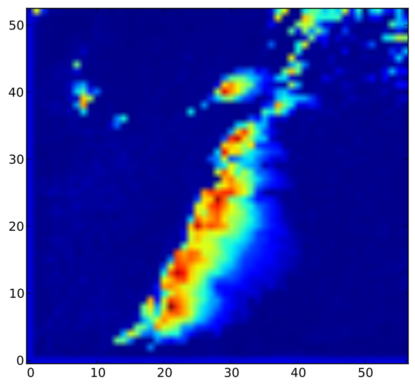
Table 4.2: *Results for predicting composite reflectivity with and without spatial inference.*

not appear to be much difference in the predictions made with and without spatial inference.

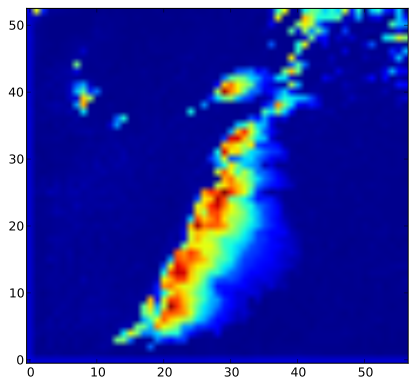
Results of our spatial inference experiments are summarized in Table 4.2. Compared to normal inference, our results for spatial inference are not significantly better. The mean RMSE of a persistence predictor for this test set is 4.50. We hypothesize that the choice of kernel and kernel parameters have a nontrivial impact on the performance of spatial inference. For our experiments, we chose a commonly used kernel function and manually set the length-scale parameter. Further experimentation is required to determine whether other kernel and parameter choices can improve the prediction accuracy of spatial inference.



(a) Actual Reflectivity



(b) Normal Inference



(c) Spatial Inference

Figure 4.10: *Visual comparison of normal and spatial inference.*

Chapter 5

Conclusions and Future Work

We presented weighted bagging for learning ELGNs, which allows us to better model local relationships for clusters within the data. When we use weighted bagging for learning and Gaussian mixture regression for inference, ELGNs are capable of accurately modeling nonlinear relationships. We also presented spatial inference for Bayesian networks, which allows us to model spatial correlations by taking into account observations at a nearby locations.

Once potentially interesting avenue for research would be the application of boosting (Freund and Schapire 1995) as an alternative to weighted bagging for learning ELGNs. For each component learned in the ensemble, boosting re-weights the training instances according to the performance of previously learned models. Though AdaBoost was originally developed for classification tasks, training examples could be re-weighted based on an error metric such as RMSE. One disadvantage of boosting is that it only applies to supervised learning problems and is only applicable if we have a specific target variable in mind when learning an ELGN. Nevertheless, boosting would provide an interesting comparison.

For our spatial inference experiments, we chose a kernel along with a length scale parameter that seemed reasonable for the data we were working with. Further investigation of different kernel functions and parameter settings is needed to take full advantage of our spatial inference framework. For example, a kernel that takes direction into account may be useful in the weather forecasting domain since storms tend to move in certain patterns. Rather than choosing parameters, it would be advantageous to devise algorithms for automatically learning kernel parameters from training data.

Gaussian processes (Rasmussen 2006) are a powerful means to modeling nonlinear data over random fields, including time and space. The unification of Bayesian networks with Gaussian processes would give us a model that is both easy to interpret in terms of dependence relationships and capable of performing nonlinear regression in spatiotemporal domains. In contrast to linear Gaussian networks, however, Gaussian processes are a nonparameteric kernel method and require storage of the training data to perform inference. In domains where Gaussian processes are already commonly used, the Bayesian network aspect of this hybrid model could provide important insights about how variables in the domain are related.

Reference List

Atkeson, C. G., A. W. Moore, and S. Schaal, 1999: Locally weighted learning. *Artificial Intelligence Review*.

Baba, K., R. Shibata, and M. Sibuya, 2004: Partial correlation and conditional correlation as measures of conditional independence. *Australian & New Zealand Journal of Statistics*, **46**, 657–664.

Bache, K. and M. Lichman, 2013: UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

Biggio, B., I. Corona, G. Fumera, G. Giacinto, and F. Roli, 2011: Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. *Multiple Classifier Systems*, Springer, 350–359.

Bilmes, J. A. et al., 1998: A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, **4**, 126.

Breiman, L., 1996: Bagging predictors. *Machine Learning*, **24**, 123–140.

—, 2001: Random forests. *Machine Learning*, **45**, 5–32.

Clark, A. J., S. J. Weiss, J. S. Kain, I. L. Jirak, M. Coniglio, C. J. Melick, C. Siewert, R. A. Sobash, P. T. Marsh, A. R. Dean, et al., 2012: An overview of the 2010 hazardous weather testbed experimental forecast program spring experiment. *Bulletin of the American Meteorological Society*, **93**.

Cleveland, W. S., 1979: Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, **74**, 829–836.

Cooper, G. F., 1990: The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, **42**, 393–405.

Cortez, P., A. Cerdeira, F. Almeida, T. Matos, and J. Reis, 2009: Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, **47**, 547–553.

Dean, T. and K. Kanazawa, 1989: A model for reasoning about persistence and causation. *Computational Intelligence*, **5**, 142–150.

Dempster, A. P., N. M. Laird, D. B. Rubin, et al., 1977: Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, **39**, 1–38.

Dereszynski, E. W. and T. G. Dietterich, 2011: Spatiotemporal models for data-anomaly detection in dynamic environmental monitoring campaigns. *ACM Transactions on Sensor Networks (TOSN)*, **8**, 3.

Dietterich, T. G., 2000: Ensemble methods in machine learning. *Multiple Classifier Systems*, Springer, 1–15.

Fast, A. S., 2010: Learning the structure of Bayesian networks with constraint satisfaction. *Open Access Dissertations*, 182.

Fisher, R. A. et al., 1915: Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, **10**, 507–521.

Freund, Y. and R. E. Schapire, 1995: A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory*, Springer, 23–37.

Friedman, N. and D. Koller, 2000: Being Bayesian about network structure. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 201–210.

Geiger, D. and D. Heckerman, 1994: Learning gaussian networks. *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 235–243.

Geiger, D., T. Verma, and J. Pearl, 1990: Identifying independence in Bayesian networks. *Networks*, **20**, 507–534.

Heckerman, D. and D. Geiger, 1995: Learning Bayesian networks: a unification for discrete and gaussian domains. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 274–284.

Heckerman, D., D. Geiger, and D. M. Chickering, 1995: Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, **20**, 197–243.

Hellman, S., 2012: *Learning ensembled dynamic Bayesian networks*. Master's thesis, University of Oklahoma.

Hellman, S., A. McGovern, and M. Xue, 2012: Learning ensembles of continuous Bayesian networks: An application to rainfall prediction. *Intelligent Data Understanding (CIDU), 2012 Conference on Intelligent Data Understanding*, IEEE, 112–117.

Koller, D. and N. Friedman, 2009: *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Mardia, K., J. Kent, and J. Bibby, 1979: *Multivariate Analysis*. Probability and Mathematical Statistics, Academic Press.

Morrison, D. F., 1967: *Multivariate Statistical Methods*. McGraw-Hill Book.

Pearl, J., 1985: Bayesian networks: A model of self-activated memory for evidential reasoning.

—, 1986: Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, **29**, 241 – 288.

—, 1988: *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Rasmussen, C. E., 2006: *Gaussian Processes for Machine Learning*. MIT Press.

Schwarz, G. et al., 1978: Estimating the dimension of a model. *The Annals of Statistics*, **6**, 461–464.

Seguí, S., L. Igual, and J. Vitrià, 2010: Weighted bagging for graph based one-class classifiers. *Multiple Classifier Systems*, Springer, 1–10.

Shachter, R. D. and C. R. Kenley, 1989: Gaussian influence diagrams. *Management Science*, **35**, 527–550.

Shieh, A. D. and D. F. Kamm, 2009: Ensembles of one class support vector machines. *Multiple Classifier Systems*, Springer, 181–190.

Spirtes, P., C. N. Glymour, and R. Scheines, 2000: *Causation, Prediction, and Search*, volume 81. MIT Press.

Sung, H. G., 2004: *Gaussian mixture regression and classification*. Ph.D. thesis, Rice University.

Tsamardinos, I., L. E. Brown, and C. F. Aliferis, 2006: The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, **65**, 31–78.

Tucker, A., V. Vinciotti, X. Liu, and D. Garway-Heath, 2005: A spatio-temporal Bayesian network classifier for understanding visual field deterioration. *Artificial Intelligence in Medicine*, **34**, 163–177.

Utz, C. M., 2010: *Learning ensembles of Bayesian network structures using random forest techniques*. Master's thesis, University of Oklahoma.

Wermuth, N., 1980: Linear recursive equations, covariance selection, and path analysis. *Journal of the American Statistical Association*, **75**, 963–972.