2015 IEEE Symposium Series on Computational Intelligence

A Comparative Study of Markov Network Structure Learning Methods Over Data Streams

Swarup Chandra, Vishal Karande and Latifur Khan
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75080
Email: (swarup.chandra, vishal.karande, lkhan)@utdallas.edu

***Abstract*—Markov network is a widely used graphical representation of data in applications such as natural language and computational biology. This undirected graph consists of nodes and edges as attributes and its dependencies respectively. One major challenge in a learning task involving Markov network is to learn its structure, i.e. attribute dependencies, from data. This has been the subject of various studies in the recent past, which uses heuristics to estimate dependencies from data. In this paper, we highlight the challenges of Markov network structure learning, and review existing methods addressing these challenges. In particular, we study the scalability of these heuristics over streaming data where data instances are assumed to occur continuously. Furthermore, we propose a new heuristic based on clustering of features, consisting of attribute dependencies, that can seamlessly update the model structure as new data arrive in a stream. This clustering technique effectively reduces search space and uses fewer number of features to generate a single model. Weight learning and inference is performed at the end of each data chunk consisting of data instances arriving within a fixed time frame. We empirically evaluate the proposed heuristic by comparing the CMLL score, on various datasets (both streaming and non-streaming), with other state-of-the-art methods.**

I. Introduction

Representation of classical machine learning models such as SVM ignore dependence information among random variables in the data. Probabilistic graphical model is a framework in which conditional independence among random variables are represented by a graph. Markov network is one such graph with undirected edges that compactly represents the joint probability distribution specified by functions over cliques in the graph. They are widely used in a variety of applications including image processing [5], medical diagnosis [10], bioinformatics [25], social networks [21] and natural language processing [23]. Important machine learning problems when using graphical models are inference, weight learning, and structure learning. In this paper, we concentrate on structure learning of a Markov network over a given dataset.

Most statistical techniques using a Markov network require a domain expert to provide relevant network structure [2]. However, appropriate domain knowledge may not always be available. Additionally, experts may make incorrect assumptions on interactions between random variables, which may increase errors during inference. These challenges can be addressed by learning the network structure, i.e. discovering conditional independencies (or dependencies) between random variables, from data. This problem is known as *Structure Learning*.

Learning the structure of a Markov network can be seen as a feature selection problem where each feature represents the dependencies between a subset of random variables. In particular, features are real-valued functions with variable assignments. The log-linear model of a Markov network enables a feature selection algorithm where necessary features are selected using the dataset. However, the search space for feature selection is exponential in the number of random variables and corresponding domain size [17]. When using a dataset with large number of random variables, a scalable strategy which cohesively reduce search complexity is warranted. Typically, search strategies use heuristic scores between variables to approximate the dependencies. These include pseudo-likelihood, information gain and support [24]. Scalability of a structure learning algorithm depends on the search heuristic considered.

A traditional algorithm that requires a complete set of training data in memory for learning cannot be used when the size of a dataset is much larger (or asymptotically unbounded) than the memory size. Such datasets are seen in the form of a data stream where data instances are continuously produced from a data generation process. Examples of data streams include telecommunication network, Twitter feed etc [1]. The problem of limited computational space is typically addressed by using online learning mechanisms which use disjoint sequential subsets of data instances called *data chunks* that can easily fit in memory [14]. These mechanisms use an incremental model that is updated periodically to reflect the characteristics of the data generation process learned from newer data instances. Furthermore, recent studies [2] have shown superior performance in inference when graphical models are used over data streams. The absence of domain experts and need for scalable learning heuristics motivates the problem of structure learning over continuous data where a graphical model is periodically updated by newer features learned from data chunks.

One major challenge in performing structure learning on a series of data chunks is the ability to efficiently combine features selected at each chunk to reflect the joint probability of the data generation process. Here, we assume that the feature generation and selection heuristic needs to be applied at every chunk. If the heuristic explores a large number of features at every chunk, this effectively slows down the data stream. Therefore, these challenges need to be addressed while designing a feature selection strategy.

Dependencies (or independencies) between random variables, learned from data, may not form an easily interpretable model. However, they may capture the independencies accurately for performing inference. A typical inference query used in traditional data stream model is to perform classification [6], [14], [15]. Here, a single query variable (namely a class variable) is considered, with the rest as evidence variables. In a similar scenario, a structure learning algorithm may be designed to perform inference with a specific set of known query variables. In these cases, a simple structure learning mechanism such as probabilistic decision tree [12] can be efficiently used to obtain features. On the contrary, the algorithm scalability is affected when a large number of query variables are present. In particular, a local model is built for each query variable, and then merged to form global model of features. If the query variables are unknown prior to model construction, then local models are built for each variable in the data. This can be computationally expensive, especially when using data streams where this operation is performed on every data chunk. Therefore, we need a computationally inexpensive method that can methodically discover the network structure.

The aim of this paper is to study the behavior of existing structure learning approaches when streaming data is used, and compare their complexity while handling multiple data chunks while performing model update with newly learned features. Here, we assume that the feature selection process is applied on each data chunk independently. Furthermore, we propose a scalable feature selection strategy to perform Markov network structure learning over streaming data. First, we propose a new structure learning method designed to minimize computational cost of discovering new independences between random variables. Second, we inculcate well-known methods to deal with data streams for structure learning. The main idea is to perform clustering of features in order to generate more relevant features over small sets of data chunks, and incrementally learn features along the data stream. In particular, we first cluster the set of training data instances over each data chunk. This produces cluster centroids that represent a set of data instances having similar features. A data instance in each cluster is considered as a feature. A set of features are uniformly selected from each cluster proportional to their size. The variable with largest (or smallest) support within a corresponding cluster is dropped to form a new feature, which is later combined with a global feature set to simulate model update. This process is iteratively performed to allow smaller length features to be eventually created. We call this method as *ClusterNet*. The weighted features learned from current data chunk are used as initial features to perform weight learning while process subsequent chunks in the data stream.

It is evident that *ClusterNet* performs a bottom-up feature generation by considered complete data instances initially as features, and dropping appropriate variables at each iteration, similar to GSSL [24]. However, the major difference is that the features are constructed based on support for a variable within the cluster rather than support from conjunction of variables across all clusters. The intuition is to enable creation of generalized features having a structure supported by features with similar structures (features within a cluster is assumed to have similar dependency structure). New features are created within a cluster using a combination of previously generated feature set and the new data instances of the current data chunk under consideration. Therefore, each cluster represents a local model. The set of features in each cluster is combined to form a global model at the end of processing a data chunk. We evaluate our method on various datasets by computing the conditional marginal log-likelihood (CMLL) score since we assume that the query variables are unknown prior to model learning. This also forms a basis of anytime query model during a streaming process where a learned model can be used to answer any query given at a particular time.

The contributions of this paper are as follows:

1) We perform a comparative study of existing Markov network structure learning approaches over streaming data. To the best of our knowledge, this is the first study performed in this setting.
2) We propose a new scalable structure learning approach, called *ClusterNet*, by using feature clusters and randomized feature generation. We show a mechanism to incrementally update the learned model over data streams.
3) We empirically evaluate *ClusterNet* by calculating the CMLL score over various datasets typically used in structure learning studies, and compare our results with other state-of-the-art methods.

The paper is organized as follows. In Section II, we discuss relevant background knowledge on Markov network and structure learning. We then discuss the behavior of various existing structure learning algorithms over a data stream in Section III. We describe our proposed clustering based structure learning method (called *ClusterNet*) in Section IV with detailed illustration. *ClusterNet* is evaluated on multiple datasets in Section V, where we present the empirical results. Finally, we conclude our paper in Section VI.

II. Preliminaries

A. Markov Network

A Markov network is an undirected graph $\mathcal{G}$ representing the probability distribution of a dataset. Each random variable X is denoted as a node in the graph, with edges representing a relationship between two variables. A non-negative real-valued function (called *potential* ϕ) is associated with each clique in the graph [18]. The joint probability distribution is represented by

$$P(X = x) = \frac{1}{Z} \prod_{k} \phi_k(x_{\{k\}})$$

where $x_{\{k\}}$ is the state of the variable $x \in \mathbb{X}$ appearing in the k^{th} clique, and Z is the partition function given by $Z = \sum_{x \epsilon X} \prod_k \phi_k(x_{\{k\}})$.

Conditional probability of a set of variables $\mathbb{X}_q$ (query), given the value of another disjoint set of variables $\mathbb{X}_e$ (evidence), is evaluated accurately by methods such as *Variable Elimination*. However, these algorithms have exponential complexity. In particular, the computation of Z (also called *partition function*) is known to be #P-Complete. Instead, an approximate algorithm is employed to approximate the probability distribution represented by the model. Sampling methods such as MCMC (Markov Chain Monte Carlo) [8], in particular Gibbs sampling, is typically used.

Weight learning optimizes a given log-likelihood or pseudo-log-likelihood objective function by associating appropriate weights for each feature. This is an optimization function whose optimal value is obtained by iterative techniques such as gradient descent. Maximum-a-posteriori (MAP) estimates are performed using gradient based or quasi-Newton optimization methods.

B. Structure Learning

Weight learning is performed over a given Markov network that encodes the independence (dependence) between variables. However, the network structure may be unknown. Characteristics of independencies between random variables may be displayed by a dataset generated in an i.i.d fashion from an unknown probability distribution. The process of discovering the knowledge about such independencies (or dependencies) between variables from a set of data instances is known as *Structure Learning*. The goal may be either to specifically answer a certain type of queries such as in classification, or a general structure that could answer any relevant MAP query. In this paper, we develop a structure learning scheme that can answer any MAP query for a dataset.

Structure learning can be seen as a two step process i.e. feature generation and selection. Typically, the feature generation process generates candidate features that can be potentially included to form a Markov network. These features are generated from the dataset in order to generalize the structure of the network. On the other hand, feature selection process determines the necessity to select these candidate features to be used to construct the network. The necessity is measured by a score i.e. either by improvement in log-likelihood, independence tests or other heuristics such as *support*. In particular, for every new feature considered for selection, the log-likelihood score is calculated for the resulting model. If it improves the score, then the feature is selected. In case of the *support* heuristic, those features that have similar attribute values from data are selected. The former scoring mechanism is computationally intensive and inhibits the scalability of structure learning algorithms. This makes the support scoring mechanism attractive.

Most studies that follow this search-based technique address the feature generation step using a *top-down* approach [16], [19] or a *bottom-up* approach [4], [17]. In a top-down approach [19], the algorithm initially considers atomic features (single variables). New candidate features are then created by conjoining two existing features. Feature selection is performed by checking whether the new feature improves the model's log-likelihood score. In particular, the candidate feature with largest gain is added to the set of features. This is iteratively performed till the model score does not improve upon creation of new candidate features. On the other hand, a typical bottom-up algorithm such as BLM [4] starts by considering a complete data instance as a feature. New candidate features are created by dropping attributes (variables) in an existing feature. This generalized set of candidate features are then selected by checking if it improves the model's score. The algorithm is iteratively performed till no improvement in score is observed. These approaches have large search space and are computationally intensive.

Yet another technique of structure learning involves the use of local models. Studies such as [20] proposed a method that learn one model per variable (local model), and then combine them to form a single global model. Recently, a decision tree based structure learning (DTSL) [12] was proposed. Here, a probabilistic decision tree is constructed for each variable to represent its conditional probability. Features are constructed by traversing the variables from the root of the tree to its leaves, and conjoining them. A union of all features are taken to form a global model by ignoring the order of variables. [13] proposes to combine BLM with DTSL.

Recently, Jan Van Haaren et al. [24] proposed a randomized mechanism to learn the structure of the network. This bottom-up learning mechanism randomly selects existing features (starting from a list of complete data instances), and then randomly drops a few variables in the feature to form a new feature. Such a mechanism has been shown to perform better than scoring based feature selection mechanisms on some datasets. Further, the randomization is shown to reduce computational time to a large extent. However, the algorithm uses only a global model, which is iteratively grown by addition of new features. In addition, it also uses a large number of features.

In this paper, we discuss the operation of above techniques while using a data stream. Furthermore, we propose a new mechanism that combines the idea of local models and randomized feature generation for developing a scalable model suitable to perform inference over streaming data. In particular, we utilize the clustering mechanism to generate multiple clusters, each operating as a local model. This is combined to form a global model at every iteration, and aids in generating new features incrementally as new data instances arrive in a stream. Unlike DTSL, we only generate clusters initially. These are maintained throughout the algorithm. We inculcate the idea of randomized selection of features from each cluster. However, the selection is performed proportional the cluster size unlike the uniform selection performed in GSSL.

C. Stream Mining

A data stream typically consists of an ordered set of data instances assumed to be generated in an i.i.d fashion. Most data streams [6] such as sensor networks, telecommunication networks and social networks generate data continuously. This characteristic of streams can generate a large dataset within a short period of time. In the case of a data mining application, a machine learning algorithm initially trains a model, and then uses it to perform inference on previously unseen data instances. When considering a data stream, a desired prediction algorithm should be incremental, low latency, one pass, fixed memory, and have easy parameter adaptation [7]. Statistical summaries are maintained using appropriate data structures. Multiple methods have been proposed to maintain statistical summaries [3], [6]. These are applied to solve problems such as frequent counts, finding quantiles etc. In our proposed approach, the summaries are the features generated in the algorithm. For each chunk, we generate features in local models, and combine them to form a set of global features. This global feature set is maintained throughout the streaming process. Typical stream mining techniques perform classification in which the class label of each data instance is predicted by the model. In this paper, we perform a generalized inference using conditional marginal log-likelihood (CMLL) for comparative purposes.

A non-stationary streaming process may exhibit drifts in data distribution at certain time periods, affecting the quality of a model [9]. These are typically known as concept drifts in terms of data classification. Due to the nature of data streams, data instances of unseen distribution may evolve at a later time period. This requires model retraining using newer data. Naturally, storing all data instance is practically impossible. Therefore, an incremental learning model is used whose parameters are periodically trained using a set of newer data instances [22]. In our study, assume that newer structures evolve over time while processing a data stream. These are captured into a global model containing features learned from previous data chunks. We address the need for incremental learning by first periodically updating the feature set of the global model, and learning weights from newer data chunks by initializing the feature weights (features excluding those learned from the current chunk) to those learned from previous chunks.

III. Structure Learning over a Data Stream

In this section, we discuss the effects of using existing structure learning methods over a data stream. In general, a simple way to adopt the structure learning mechanism over a data stream is perform feature generation and selection on each data chunk. Features generated in each chunk are combined together to form a global set of model features. These features are used inference over the next chunk. Note that the feature weights are learned using the current data chunk. In this paper, we use this mechanism to discuss some structure learning methods over streaming data.

A. Top-Down Approach

A top-down approach starts with atomic features. Since each chunk can be assumed to contain all non-query variables, the feature generation and selection mechanism will have to explore the whole feature space. Chunk based technique only reduces the size of the dataset, without reducing the number of random variables in each data instance. Hence, for data having large number of random variables, this approach will not be suitable to perform stream mining.

B. Bottom-Up Approach (BLM)

Similar to top-down approach, the search mechanism for bottom-up approach will also not scale enough to handle streaming data since the search space is not reduced within each data chunk. The feature generation and selection method performs similar to that of a non-stream dataset within each chunk of the streaming dataset since each complete example is used to generate new features, and a scoring mechanism to select an appropriate feature for the global model.

C. DTSL

A probabilistic decision tree streamlines the search criteria in structure learning. However, they require one tree per variable in the dataset. Within each chunk, the *DTSL* mechanism builds a set of trees whose size is equal to the number of query variables, i.e. to represent the query variable X_i, given all other variables, in $P(X_i|\mathbb{X} - X_i)$. Let n is the number of variables, m is the number of training examples within each chunk, and l is the domain size of each variable. If each split has at least $\frac{1}{k}$ examples in its child, then the total complexity for one tree is $O(lmn\log_k(m))$, and for the entire structure is $O(lmn^2\log_k(m))$ [12]. This is repeated at every chunk, and is quadratic in its variable size.

D. GSSL

This method uniformly chooses a feature from the existing feature set, and drops an arbitrary number of variables to form a new feature. The new feature is added to the global feature set. At each chunk, this operation is repeated a finite number of times (set by a threshold max). The new features generated are added back to the global set at the end of each chuck. Therefore, this technique can be efficiently used scale structure learning over data streams. However, the uniform sampling of features during feature generation can select features having similar properties. Thus, new features generated may be rejected by the feature selection algorithm if the global set already contains the newly generated features. We address this by using data clustering to group similar features and sample according to their relative size.

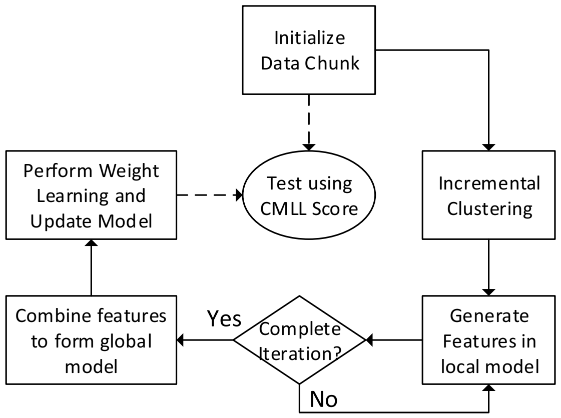
Initialize Data Chunk
Test using CMLL Score
Perform Weight Learning and Update Model
Incremental Clustering
Combine features to form global model
Yes
Complete Iteration?
No
Generate Features in local model

Fig. 1: Illustration of *ClusterNet* overview.

IV. ClusterNet : Clustering Based Markov Network Structure Learning

In this section, we describe the process of feature generation and feature selection using clustering.

The *ClusterNet* structure learning process is illustrated in Figure 1. A *data chunk* is formed using m data instances occurring sequentially in the stream. The first chunk is used to create a set of k clusters using complete data instances. Centroids obtained from clustering is then fixed. After this initialization, the *ClusterNet* algorithm converts the feature vector of each data instance into a propositional form, and assigns it to the cluster having the closest centroid (according

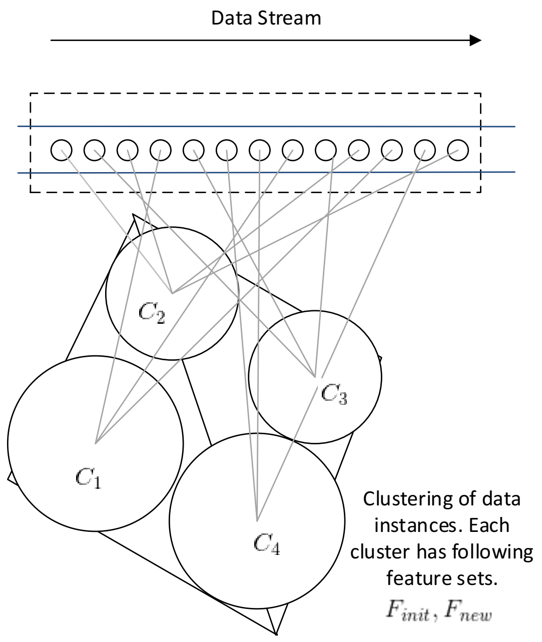
Data Stream
C_1
C_2
C_3
C_4
Clustering of data instances. Each cluster has following feature sets.
F_{init}, F_{new}

Fig. 2: Initialization process of *ClusterNet* where data streams are considered in chunks, and k-means is applied over it.

to Euclidean distance). These steps form the incremental clustering at each cluster. It then generates a set of features within each cluster by iteratively selecting an existing feature and dropping one of its variable. Each newly created feature is then assigned back to the same cluster. The set of features within each cluster form a local model. Therefore, *ClusterNet* combines the efficiency of randomized feature generation and capture localized structural properties within each cluster. This algorithm is performed iteratively, with replacement, to generate more generalized features. Finally, feature weight learning is incrementally performed using the current data chunk. The learned weights are used to compute the CMLL score using the data instances from the next chunk as test dataset. This is repeated throughout at every instance when a new data chunk is encountered.

We now discuss the initialization, feature generation and overall algorithm execution in detail.

A. Initialization

Figure 2 illustrates the process of initialization. Data instances from the first data chunk is considered. k-means clustering is employed over these instances, within its feature space, to create k clusters. Here, C_i is shown to be the corresponding cluster centroid of the i^{th} cluster, where $i \in \{1 \cdots k\}$, with $k = 4$.

Each data instance in a cluster is converted to a propositional form, which is used a feature. In this paper, we assume binary variables. Therefore, the value of the variable denoted by 1 or 0, indicates the presence or absence of the variable

Instances	V_1	V_2	V_3	V_4
1	0	1	1	0
2	1	1	0	1
3	1	0	0	1
4	0	1	0	1

TABLE I: Example dataset illustrating variables and corresponding values

Instances	Feature
1	$V_2 \wedge V_3$
2	$V_1 \wedge V_2 \wedge V_4$
3	$V_1 \wedge V_4$
4	$V_2 \wedge V_4$

TABLE II: Initial set of features (propositional form) generated from the example dataset.

in that feature. As mentioned in [24], we use the conjunction of positive variables (having "true" value) as a feature. This aids in specifying sparse domains effectively. For example, consider the dataset shown in Table I as an example. There are 4 variables denoted as V_1, V_2, V_3 and V_4. A data instance has a value of 0 or 1 for each variable (denoted by a row). If a domain is sparse, then majority of variables have a value of 0 for most data instances. Therefore, a feature corresponding to each data instance can be expressed as given in Table II. We consider all training examples from each chunk and convert them into the propositional form. These features are then assigned to the appropriate cluster in the model, corresponding to the data instance through which it is formed. The initial set of features are stored as a separate list in these clusters.

B. Feature Generation

Algorithm 1 illustrates the process of feature generation ($genFeatures$) within each cluster. First, random variables with highest and least *support* are calculated in a given cluster. Support for every variable is calculated by counting the number of data instances having true value for that variable in the initial feature set. A variable with the largest count indicates highest support, denoted as V_H. This is calculated using $calcSupportHigh()$. Similarly, a variable with the smallest count indicates least support, denoted as V_L. This is calculated using $calcSupportLow()$ For example, from Table II, $V_H = V_2$ since most data instances have $V_2 = 1$, and $V_L = V_3$ since most data instances have $V_3 = 0$.

Each cluster maintains two sets of feature lists (or sets). One set of features contain the initial feature set generated by the initialization procedure. The next set of features are those created during feature generation process. We refer to the first set as *Initial Feature Set* (F_{init}), and the latter set as *New Feature Set* (F_{new}). The $selectFeatureSet$ procedure probabilistically chooses a feature set (F_{init} or F_{new}) from which a feature is to be selected for feature generation, using a weight threshold denoted by w. We generate a fixed number of features (or samples) across all clusters at this stage. This is denoted as α. However, the number of features generated per cluster may vary. For a given cluster, s indicates the number of samples needed to be generated. This value is proportional to the cluster size (which is given by $|F_{init}| + |F_{new}|$) with respect to the fixed number of features.

Algorithm 1: $genFeatures$: Feature Generation Algorithm

```
Data: Data stream Chunk
Input: Cluster Initial Features $F_{init}$, Cluster New Features $F_{new}$, Number of Samples $s$, Weight $w$
Result: List of new features $F_{cand}$
1  begin
2      $V_H = calcSupportHigh()$
3      $V_L = calcSupportLow()$
4      for $i \in (1 \ldots s)$ do
5          $F = selectFeatureSet(F_{init}, F_{new}, w)$
6          $f \leftarrow selectFeature(F)$
7          $f_h \leftarrow dropSupportHigh(f, V_H)$
8          $f_l \leftarrow dropSupportLow(f, V_L)$
9          $F_{cand} \leftarrow \{f_h, f_l\}$
```

Next, a feature within the chosen feature set is selected uniformly at random. If $V_H = 1$ exists in the chosen feature, then the variable is dropped to create a new feature (performed using $dropSupportHigh(\cdot)$). Similarly, if $V_L = 1$ exists, then another feature is created by dropping V_L (performed using $dropSupportLow(\cdot)$). This creates two new features. For example, let the randomly chosen feature be $V_1 = 1 \wedge V_2 = 1 \wedge V_3 = 1$. If $V_H = V_2$, then a new feature $V_1 = 1 \wedge V_3 = 1$ is created by dropping V_2. Similarly, if $V_L = V_3$, then another feature $V_1 = 1 \wedge V_2 = 1$ is created. This generates the candidate features, denoted by F_{cand}.

C. Approach

Algorithm 2 illustrates the overall structure learning process in *ClusterNet*. After the initialization step (shown in lines 7), the cluster centroids are fixed throughout the algorithm (while processing subsequent chunks). The data from each chunk is converted to the propositional form to represent a set of features. For each feature, cluster having the closest centroid to the corresponding data instance is computed. The feature is then added to the cluster's F_{init} feature set. The feature generation and step is performed as shown in lines 15 through 22. Here, new features are generated using the $genFeatures(\cdot)$ procedure (Algorithm 1) within each cluster. First, the proportion of samples to be generated within a given cluster c is computed using $compProportion$. This provides the number of samples s_c to be generated as a proportion of α. The weight w is computed as shown in line 20. The value of the weight increases with the number of iterations. This aids in sampling F_{init} more in the initial stages, and then F_{new} more at the later stages in the iteration in $genFeatures$. Such a scheme enables the formation of smaller features as new features are generated at every iteration. The candidate feature F^c_{cand} is obtained, $F^c_{cand} \cup F^c_{new}$ is performed to update F^c_{new}. This feature generation process is repeated multiple times. In particular, if there are n random variables in the dataset, the iteration is performed n times so that more generalized features having lesser variables are generated. Finally, the global feature set F_{global} is updated using $updateGlobalFeatures$ procedure. This updated global feature set is used for weight learning using the data instances from the current chunk. Moreover, we learn weights incrementally by using the previously learned weights of existing features as their corresponding

Algorithm 2: Overview of *ClusterNet* algorithm

```
Data: Data stream
Input: Number of Clusters $k$, Number of Features $\alpha$
Result: CMLL Score
1  begin
2      Initialize data chunks
3      $i = 0$
4      $subScore = 0$
5      while $Chunk_{i+1}$ exists do
6          if $i = 0$ then
7              Perform $k$-means on $Chunk_i$
8          else
9              $F_{global} \leftarrow getGlobalFeatures()$
10             $cmll \leftarrow compCMLL(Chunk_i, F_{global})$
11             $subScore \leftarrow subScore + cmll$
12         $F_{all} \leftarrow toPropositionForm(Chunk_i)$
13         for $f \in F_{all}$ do
14             Assign $f$ to closest Cluster's $F_{init}$
15         for $j \in \{1 \ldots n\}$ do
16             for each cluster $c$ do
17                 if $|F^c_{init}| \neq 0$ then
18                     $s_c \leftarrow \alpha \times compProportion(c)$
19                     $\{F^c_{init}, F^c_{new}\} \leftarrow c$
20                     $w \leftarrow \frac{n-j}{n}$
21                     $F^c_{cand} \leftarrow genFeatures(F_{init}, F_{new}, s_c, w)$
22                     $F^c_{new} \leftarrow F^c_{new} \cup F^c_{cand}$
23         $F_{global} \leftarrow updateGlobalFeatures()$
24         $F_{global} \leftarrow learnWeights(F_{global}, Chunk_i)$
25         $i = i + 1$
26     Output $Score = \frac{subScore}{i}$
```

initial weights in the current weight learning process. The initial weights of newly learned features, i.e. features from F_{new} added to F_{global}, is assigned 0. This essentially forms a incremental learning mechanism. The weighted global feature list is used to compute the CMLL score using $computeCMLL$ while using the data from next chunk. The algorithm outputs average CMLL score over all chunks processed.

Overall, the algorithm performs a fixed number of iterations to generate new features when dropping a single variable. However, the number of iterations within each cluster depends on its feature size. The newly created features are added to the clusters at the end of the inner *for*-loop. Therefore, this may increase the feature count within the cluster for further creating newer features. Unlike [24], the feature selection process is embedded within the generation process as no duplicate features are retained.

D. Asymptotic Complexity

When the first chunk $Chunk_0$ is encountered, the k-means clustering algorithm is performed over the data instance within the chunk. The overall complexity of this algorithm is $O(mknr)$ where m is the number of training examples, n is the number of variables, k is the number of clusters

Dataset	Number of variables	Number of instances
MSNBC	17	388,434
KDDCup 2000	64	234,954
Plants	69	23,215
MSWeb	294	37,711
WebKB	839	4,199
Reuters-52	889	9,100
20-Newsgroups	1058	18,821

TABLE III: Dataset Characteristics

and r is the number of iterations required to converge. The feature generation process within each cluster generates a maximum of α number of features. The support calculation takes $O(M_p)$ times, where M_p is the number of features within cluster $p \in \{1 \dots k\}$. The total computation within each cluster is $O(\alpha + M_p)$. This is asymptotically equal to $O(|F_{max}|)$, where $|F_{max}| = max(\alpha + M_p, |F_{global}|)$. Therefore, the complexity for all clusters is $O(k|F_{max}|)$. The complexity of assigning each candidate feature to closest cluster is $O(k|F_{max}|)$. Therefore, the overall complexity of the algorithm is $O(mknr + (n * k * |F_{max}|))$. It should be noted that the size of F_{max}, i.e. $|F_{max}|$, can be significantly less than the feature set size of GSSL algorithm since we do not keep all generated features in memory, and the features are spread across multiple clusters. At the end of each chunk, the feature selection is performed.

V. Empirical Results

In this section, we show our empirical evaluation of *ClusterNet* on multiple datasets, and compare the results with other existing approaches.

A. Experiments

We perform multiple experiments using the same datasets as [24]. Table III lists the real-world datasets.

All previous structure learning methods are performed over static data (non-streaming). In order to provide a fair comparison, we perform experiments using various values of k, i.e. using different number of clusters, using both streaming and non-streaming settings. In the non-streaming case, only one chunk is considered to contain all data instances in the data set. On the other hand, we divide the dataset into 10 chunks in the streaming case. We also fix the number of samples per iteration to be 1000. Depending on the size of each cluster, samples or features are selected from each cluster for generating candidate features. If a candidate feature is unique, it is added to the new feature list.

The CMLL score is calculated as follows [24]. A chunk is considered from the data stream. This is used to generate features. Weight learning is performed using this chunk to obtain feature weights. The next chunk of data instances are considered as test set. Conditional marginal log-likelihood (CMLL) is calculated by evaluating the best model using this test set. The test set is divided into four disjoint sets of variables. While one set a query set Q, the rest is considered as evidence set E. CMLL is computed using

$$CMLL(X = x) = \sum_{i \in Q} logP(X_i = x_i|E) \qquad (1)$$

Dataset	CMLL Score			
	GSSL (Non-Stream)	DTSL (Non-Stream)	ClusterNet (Non-Stream)	ClusterNet (Stream)
MSNBC	**-1.29**	-4.30	-1.29	-1.77
KDDCup 2000	-1.61	-2.19	**-0.59**	-2.54
Plants	**-2.58**	-9.06	-2.60	-3.25
MSWeb	-11.50	**-8.37**	-14.31	-61.89
WebKB	**-140.66**	-151.18	-146.38	-152.55
Reuters-52	-109.62	**-83.29**	-130.33	-137.48
20-Newsgroups	-157.54	**-140.72**	-157.89	-162.76

TABLE IV: CMLL Score on various datasets

Dataset	Number of features		
	GSSL (A)	ClusterNet (B)	% Fraction (B/A)
MSNBC	491,885	35,367	7.19
KDDCup 2000	373,041	37,329	10
Plants	690,390	54,551	7.9
MSWeb	149,689	32,104	21.44
WebKB	2,738,782	52,054	1.9
Reuters-52	2,514,634	55,189	2.19
20-Newsgroups	3,037,720	61,483	2.02

TABLE V: Number of features obtained

The overall score is computed by averaging the the score of four iteration, each consisting of a set considered as query variables once. We use the Libra toolkit's [11] Gibbs sampler to compute these marginal probabilities. A burn-in of 100 samples are employed, and the probability if computed using the next 1000 samples. We compare the result of our experiment with GSSL [24] and DTSL [12] structure learning methods only since [24] show that these two methods perform superior to BLM [4] and other top-down approaches.

B. Results

The results of the empirical evaluation are given in Table IV. The scores show that the non-stream version of *ClusterNet* performs equivalently to other existing approaches in most datasets. In particular, the *ClusterNet* CMLL score on the KDDCup 2000 dataset performs the best. This dataset contains moderate number of variables with large number of data instances. This provides sufficiently large number of iterations that can create generalized features, and have more number data instances per chunks. This may provide better clusters and feature localization than uniformly random selection of feature over the entire dataset as performed by GSSL. Naturally, the approach is better than DTSL in 4 out of 7 cases since the combination of clustering with randomization provides a platform for broader search and localization criteria. Further, Table V shows the number of features obtained using *ClusterNet*. When compared to GSSL, *ClusterNet* uses significantly less number of features resulting in similar scores. The table shows the fraction of features *ClusterNet* uses when compared to GSSL, to illustrate the significance. For example, consider the 20-Newsgroup dataset. Table IV shows that the CMLL scores of GSSL and *ClusterNet* are -157.54 and -157.89 respectively. The difference between these two scores is 0.35. However, Table V indicates that *ClusterNet* only uses 2.02% of features in comparison with GSSL to obtain a similar CMLL score.

Finally, all scores presented here on *ClusterNet* are obtained with $k = 100$. We found that with higher magnitude of k value, the score does not vary significantly. However, with lower value of k, the score significantly varies. This may be due to the number of variables in each dataset. With lower k, the number of clusters is significantly less than the number of variables. This may restrict the formation of varied features in the feature generation stage. On the other hand, the *ClusterNet* approach under-performs compared to DTSL and GSSL on datasets such as Reuters-52. This may be due to large number of variables and small dataset size.

In the case of the data stream setting, *ClusterNet* score obtained is lower than that of the batch (non-stream) approach. This is expected since we evaluate the stream by using chunks and average all chunk scores. This decreases the over model score on each dataset.

It is important to note that the computational complexity of *ClusterNet* is greater than that of GSSL since GSSL only rely on randomization. However, since we do not perform independence tests or compute conditional probabilities at every iteration for feature selection, the algorithm is faster than BLM and DTSL. Furthermore, during streaming, we greedily generate new features at every chunk. This is added to the global feature list. However, we found that the number of new features added at the later part of the stream is significantly less compared to the initial part of the stream. This is expected as features with different combinations of variable values could occur initially, and might repeat after a few chunks. The time of feature generation can be reduced by selectively determining if newer features need to be created. We leave this for future work.

VI. Conclusion

In this paper, we perform a comparative study of various Markov network structure learning methods. In particular, we first study the behavior of existing structure learning techniques while operating with a data stream. We discuss various issues and complexities that would affect the streaming process while using these heuristic approaches. Furthermore, we propose a scalable structure learning approach for easy model update in a data stream. Instead of using a single global model for feature generation, we propose a new method (called *ClusterNet*) to perform structure learning by combining clustering of features with randomized feature generation. We argue that clustering aids in generating localized structure using similar data instances. Our empirical evaluation shows similar score as compared to competing methods, while using significantly less number of features. This is illustrated on various datasets.

Acknowledgement

This research is partially supported by the NSF Award No. CNS-1229652, DMS-1322353 and the Air Force Office of Scientific Research under Award No. FA-9550-12-1-0077. Any opinions, findings and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] A. Bifet and E. Frank, "Sentiment knowledge discovery in twitter streaming data," in *Discovery Science*. Springer, 2010, pp. 1–15.

[2] S. Chandra, J. Sahs, L. Khan, B. Thuraisingham, and C. Aggarwal, "Stream mining using statistical relational learning," in *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 743–748.

[3] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," in *LATIN 2004: Theoretical Informatics*. Springer, 2004, pp. 29–38.

[4] J. Davis and P. Domingos, "Bottom-up learning of markov network structure," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 271–278.

[5] N. Friedman and S. Russell, "Image segmentation in video sequences: A probabilistic approach," in *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1997, pp. 175–181.

[6] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, 2005.

[7] J. Gama, R. Sebastiao, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.

[8] W. R. Gilks, *Markov chain monte carlo*. Wiley Online Library, 2005.

[9] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[10] H. Li, M. Kallergi, L. Clarke, V. Jain, and R. Clark, "Markov random field for tumor detection in digital mammography," *Medical Imaging, IEEE Transactions on*, vol. 14, no. 3, pp. 565–576, 1995.

[11] D. Lowd, "The Libra Toolkit," http://libra.cs.uoregon.edu/.

[12] D. Lowd and J. Davis, "Learning markov network structure with decision trees," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 334–343.

[13] ——, "Improving markov network structure learning using decision trees," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 501–532, 2014.

[14] M. M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Addressing concept-evolution in concept-drifting data streams," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 929–934.

[15] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 6, pp. 859–874, 2011.

[16] A. McCallum, "Efficiently inducing features of conditional random fields," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 403–410.

[17] L. Mihalkova and R. J. Mooney, "Bottom-up learning of markov logic network structure," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 625–632.

[18] J. Pearl, "Probabilistic reasoning in intelligent systems: Networks of plausible reasoning," 1988.

[19] S. D. Pietra, V. D. Pietra, and J. Lafferty, "Inducing features of random fields," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 4, pp. 380–393, 1997.

[20] P. Ravikumar, M. J. Wainwright, J. D. Lafferty *et al.*, "High-dimensional ising model selection using 1-regularized logistic regression," *The Annals of Statistics*, vol. 38, no. 3, pp. 1287–1319, 2010.

[21] T. A. Snijders, "The statistical evaluation of social network dynamics," *Sociological methodology*, vol. 31, no. 1, pp. 361–395, 2001.

[22] N. A. Syed, H. Liu, and K. K. Sung, "Handling concept drifts in incremental learning with support vector machines," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 317–321.

[23] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation." in *AAAI*, 2011.

[24] J. Van Haaren and J. Davis, "Markov network structure learning: A randomized feature generation approach," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[25] Z. Wei and H. Li, "A markov random field model for network-based analysis of genomic data," *Bioinformatics*, vol. 23, no. 12, pp. 1537–1544, 2007.