# Stream Mining Using Statistical Relational Learning

Swarup Chandra, Justin Sahs, Latifur Khan and Bhavani Thuraisingham
*Department of Computer Science*
*The University of Texas at Dallas, Richardson, TX, USA.*
*Email: (src093020, jcs074000, lkhan, bxt043000)@utdallas.edu*

Charu Aggarwal
*IBM T. J. Watson Research Center*
*Yorktown, NY, USA.*
*Email: charu@us.ibm.com*

***Abstract*—Stream mining has gained popularity in recent years due to the availability of numerous data streams from sources such as social media and sensor networks. Data mining on such continuous streams possess a variety of challenges including concept drift and unbounded stream length. Traditional data mining approaches to these problems have difficulty incorporating relational domain knowledge and feature relationships, which can be used to improve the accuracy of a classifier. In this work, we model large data streams using statistical relational learning techniques for classification; in particular, we use a Markov Logic Network to capture relational features in structured data and show that this approach performs better for supervised learning than current state-of-the-art approaches. Additionally, we evaluate our approach with semi-supervised learning scenarios, where class labels are only partially available during training.**

***Keywords*-Stream Mining; Statistical Relational Learning; Classification;**

## I. Introduction

Data mining on continuously occurring data instances in streams such as web clicks for advertising, telecommunications, and sensor data cannot use traditional algorithms that assume the availability of complete datasets prior to computation. In particular, data stream classification possess unique challenges including change in data distribution over time (called *Concept Drift*), and hardware limitations that may prevent storage and processing of very large datasets. These challenges are typically addressed using a single or an ensemble of classifiers [1], with computations performed on a subset of data instances at a time.

A desirable stream classifier would represent a generative data streaming process, adapting to changes in distribution as the process evolves over time, and providing an efficient manner to perform probabilistic inference and learning. This includes the ability to compactly represent relationships between attributes. Traditionally, stream classification models (or classifiers) are not designed to incorporate explicit relational structures of a data generative process they represent. The characteristic of representing relationships between attributes, in a stochastic model, can be expressed using languages from the field of Statistical Relational Learning. With the use of an expressive relational language that aid in the specification of known relational features, the classifier performance can be improved. We propose a solution to the data stream classification problem that captures inherent relational structures, and expert domain knowledge, using a statistical relational language called Markov Logic Networks [2] (MLN). Previous studies [3], [4] have shown the usage of MLNs for data classification. However, these applications do not address the challenges of stream mining.

Classification of streaming data may require a quick and accurate algorithm to process all data instances. However, in certain cases, a highly accurate classifier may be prudent. For example, data obtained after processing a set of images captured by a satellite, for classifying them into different pollution levels, can be used as an input to a stream classifier. Data can be periodically collected to determine small shifts in pollution level, which may gravely affect the ecological system in a particular geographic area. In such cases, a highly accurate classifier is imperative. A relational model can incorporate inter-related pertinent factors directly affecting pollution levels, and expert domain knowledge, to provide acceptable results.

In this paper, our intuition is to embed domain knowledge and feature relationships into an MLN, which can be trained and used for classification within a stream mining framework. We perform chunk-based classification, with incremental training of the MLN model. Using an MLN on a streaming data possess multiple challenges including incremental weight learning, data discretization, and execution time. We explore various approaches to address these challenges. A selective training method is used to determine the need for model training. We compare the performance of our algorithm with existing state-of-the-art systems (which use supervised learning) on relational datasets, and show that our approach performs better, in terms of accuracy, in most of the cases. Additionally, we provide an evaluation of our approach using semi-supervised learning, where ground truth on training data are only partially available.

The remainder of this paper is organized as follows. First we review some background information in Section II, on stream mining and MLN. We then discuss various challenges in using an MLN as a classifier, and describe our methods to overcome these challenges in Section III. In Section IV, we describe the datasets, experimental setup and results. Finally in Section V, we conclude our paper.

## II. Background

In this section, we briefly describe relevant background information on stream mining and Markov logic networks.

*Stream Mining:* An ordered set of data instances, forming a stream of infinite length, may evolve over time. This induces a non-stationary data distribution, popularly known as concept drift. A stochastic model, or a classifier, representing such a distribution will have to adapt to these changes in order to reduce misclassification errors. A drift can be a change in conditional probability distribution of the concept (called real drift), or a change in feature boundary (called virtual drift) [5]. In this paper, we propose a methodology addressing virtual drift.

*Markov Networks:* A Markov network is an undirected graph denoted as $\mathcal{G}(\mathbf{X}, \mathbf{E})$, where $\mathbf{X}$ is a set of random variables and $\mathbf{E}$ is a set of edges representing relationships between random variables [6]. The relationships are captured using a set of $k$ potential functions $\boldsymbol{\phi_k}$ associated with cliques in the graph. The joint probability distribution is given by-

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

where $x_{\{k\}}$ is the state of $k^{th}$ clique, and $Z$ is the partition function given by $Z = \sum_{x \epsilon X} \prod_k \phi_k(x_{\{k\}})$. An equivalent log-linear model represents the product of potential functions with an exponential sum of weighted features (binary or real-valued). Exact inference over large and complex networks is known to be intractable [7]. Therefore, approximate methods are used to perform inference and learning. Sampling based inference algorithms utilize samples to asymptotically approximate the probability distribution represented by the model.

Equivalently, a Markov network can be represented by a set of weighted propositional formulas over propositions corresponding to the nodes in the undirected graph. First Order Logic provides a more powerful and expressive way to capture various relationships among propositions using predicates and quantifiers. A set of formula in First Order Logic (FOL) are constructed using constants, variables, functions, predicates, and quantifiers. An atomic formula or atom is a predicate symbol applied to a tuple of propositions. A world, or grounding, is a set of assignments to atoms. A formula is satisfiable if there exists at least one world in which it is true.

*Markov Logic Networks:* Markov Logic Networks consist of weighted first order logic formulas over which learning and inference are performed [2]. Nodes in the network are formed from groundings of predicates. Nodes corresponding to predicates occurring in a FOL formula are connected to each other to form a clique. A binary feature is associated with each possible grounding of the formula. It has value 1 if its ground formula is true, and 0 otherwise. Each formula $f_i$ is associated with a weight $w_i$. The weight allows non-zero probability for worlds that fail to satisfy the formula. The log-linear probability distribution over possible worlds $x$ is given by-

$$P(X = x) = \frac{1}{Z} exp(\sum_i w_i n_i(x)) \tag{2}$$

where, $n_i(x)$ is the number of true groundings of $f_i$ in $x$. Inference in MLN, i.e. finding the most likely world given a set of observations, can be performed using MaxWalkSAT or MC-SAT [8]. Weight learning can be performed generatively by learning the joint probability $p(x, y)$, or discriminatively by learning the conditional probability $p(y|x)$ directly, using a variety of methods [9]. Here, $x$ is the input and $y$ is the label.

*Hybrid Markov Logic Networks:* In the case of real valued predicates, a Hybrid Markov Logic Network (HMLN) [10] with real valued features can be used. The value of a feature in the network is a numeric value of its corresponding ground term. In this case, $n_i$ in Equation 2 is replaced by $s_i$, which is the sum of the values of all groundings of formula $f_i$ in $x$.

## III. Relational Stream Classification

In this section, we describe the challenges involved when using MLNs as classifiers for stream mining, and provide an approach to overcome these challenges.

### *Challenges*

Challenges in designing a data stream classifier using MLN includes choice of graph structure, computational time, concept drift and storage limitation.

Current methods for inference and learning on an MLN requires a finite domain [2]. Further, large domain size inputs may create a large grounded network. In such cases, attributes with large or continuous domain need to be discretized for limiting its domain size. The choice of discretization method may affect the classifier accuracy.

Computational time and space complexity is largely influenced by the size of cliques, formed by predicates in a formula, in the graph. Large formulas create cliques requiring greater computational power, but a smaller formula may not sufficiently capture data relationships in the domain. Additionally, formulas with hard constraints have zero probability for a world violating it. Whereas, soft constraints are weighted worlds having non-zero probability if it violates a formula. An appropriate choice of FOL formulas affects the efficiency of the MLN classifier.

Concept drift is addressed by periodically re-training the classifier parameters while processing streaming data. In case of an MLN, re-training a model using approximate weight-optimization algorithms may be slow compared to the streaming data rate. The overall algorithm efficiency can be largely improved by reducing the re-training time.

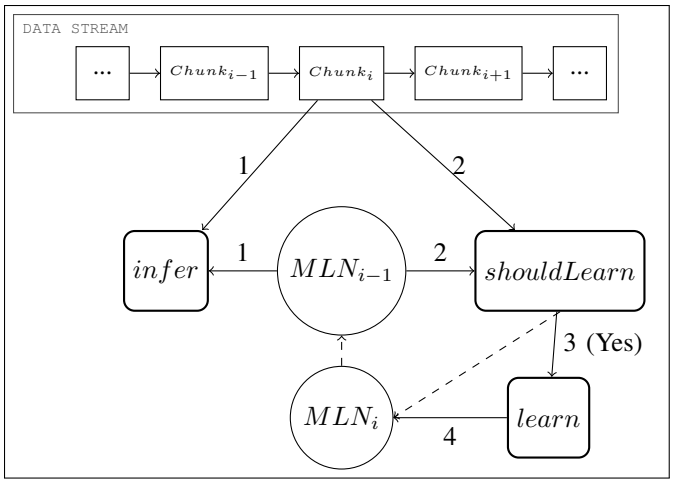


Figure 1: Block Diagram for Stream Mining using MLN

Finally, space constraint for storing all data instances in a stream is addressed by processing a small set of sequential data instances (called a data chunk) in an iterative manner. Number of data instances considered in each chunk (chunk size) may affect the classifier's ability to adequately capture changes in data distribution, increasing its misclassification rate on test data. Larger chunk size may increase execution time for weight learning in MLN. Whereas, a smaller chuck may not appropriately represent changes in the data distribution.

*Approach*

We address these challenges by designing a method involving a single incremental MLN classifier. Expert domain knowledge and feature relationships are incorporated into the MLN, which is used for classification. Data chunks are considered sequentially for classification and weight learning, both of which are performed iteratively. Weight learning is performed in an incremental manner, where the MLN used to process the next chunk in the sequence, is initialized with the weights learned using the current chunk.

*Classification and Training:* A chunk is formed using a small set of sequential data instances. Each chunk is assumed to have a fixed size. An initial MLN is manually constructed, using expert domain knowledge, and trained using the first chunk ($dChunk_1$) of data instances (preprocessed) as shown in Line 3 of Algorithm 1. A chunk is discretized, when necessary, as shown in Line 2. The resultant MLN ($MLN_1$) is then used to test the next data chunk in the sequence (Line 7). Weights are updated by re-training the current MLN ($MLN_{i-1}$) to produce a new MLN ($MLN_i$) as shown in Line 13 of the algorithm. This process of classification and re-training is iteratively performed for all chunks, as depicted in Figure 1. Here, the classification process, depicted as Step 1, uses $MLN_{i-1}$ whose weights are trained using $Chunk_{i-1}$. Steps 3 and 4 depicts the re-training process,

**Algorithm 1** Stream Mining using MLN

1: **Input:** Initial $MLN_0$; $N$ data chunks of size $k$.
2: $dChunk_1 = discretizeChunk(Chunk_1)$
3: $MLN_1 = learn(dChunk_1, MLN_0, Label_1)$
4: Let Q be the data distribution
5: $Q = getDistribution(Q, dChunk_1)$
6: **for** each chunk $i = 2 : N$ **do**
7: $Result_i = infer(dChunk_i, MLN_{i-1})$
8: $getError(Result_i, Label_i)$
9: $P = getDistribution(Q, dChunk_i)$
10: **if** $i < N$ **then**
11: $dChunk_i = discretizeChunk(Chunk_i)$
12: **if** $shouldLearn(Q, P)$ **then**
13: $MLN_i = learn(dChunk_i, MLN_{i-1}, Label_i)$
14: Update $Q$ with $dChunk_i$
15: **end if**
16: **end if**
17: **end for**
18: **Output:** Classification Error.

where weights of $MLN_{i-1}$ is used to initialize the formula weights for a weight-learning algorithm, resulting in $MLN_i$ with updated weights. $MLN_i$ is then used in the next iteration for classification of $Chunk_{i+1}$.

*Preprocessing:* In the process of stream mining using MLN, data discretization is performed on attributes with real-valued domain, or large domain size. We now describe two methods to perform discretization.

First, we assume that the largest and smallest values for an attribute in the dataset is known. The width of this domain is the difference between the largest and the smallest value. In this case, every real-valued attribute $a$ in a data instance is mapped to at most one of the $k$ discrete-valued element in the co-domain. These discrete values (or bins) correspond to an integer in the range $\{1 \dots k\}$. The discrete co-domain is formed by dividing the real-valued domain width into $k$ parts of equal width. This is given by $y_a = \lfloor \frac{(x_a - x_a^s) * k}{(x_a^l - x_a^s)} \rfloor$, where $x_a$ is the data value of a real-valued continuous attribute $a$, $y_a$ is the corresponding discrete value obtained, $x_a^l$ is the largest known value for attribute $a$ in the dataset, and $x_a^s$ is the smallest known value for that attribute. This binning strategy is known as equal-width binning since each bin has a domain of equal width.

In many cases, it may not be possible to know the largest and the smallest value of a real-valued attribute in a data stream, especially if the stream length is assumed to be unbounded. An adaptive binning strategy may be used, where the current largest ($x_a^l$) and the smallest ($x_a^s$) value of attribute $a$, observed in the stream, can be considered for binning. Here, each chuck is used to update the largest and the smallest value observed for a particular attribute. This update is performed when the largest (or smallest) value

observed for that attribute, in the current chunk, is greater (or smaller) than the largest (or smallest) value used for discretizing the previous data chunk.

*Selective Learning:* We observe that training a model with every new chunk may not be necessary if the overall data distribution of the stream, obtained by processing the current chunk, does not significantly differ from the data distribution of the chunk. The overall computational time can then be reduced by selectively re-training the model using only a subset of data chunks. This selective training procedure is shown in Line 12 for Algorithm 1. Here, $M$ and $Q$ are probability distributions of data instances in the current chunk, and the overall data distribution in the observed stream respectively. We use the Kullback-Leibler distance measure (KL), in the $shouldLearn$ procedure, as a distance measure between distributions for each attribute $a$. This is given by-

$$KL_a = \sum_{z \in \mathcal{D}} (P_M(z) - P_Q(z)) * \log \frac{P_M(z)}{P_Q(z)} \tag{3}$$

The KL measure is obtained over each attribute whose domain is $\mathcal{D}$. In order to evaluate the need for MLN re-training, we measure the relative difference in KL distance between two consecutive iterations ($KL^{prev}, KL^{curr}$), given by-

$$d_a = \frac{|KL_a^{prev} - KL_a^{curr}|}{KL_a^{prev}} \begin{cases} True\colon d_a > T \\ False\colon d_a \leq T, \forall a \end{cases} \tag{4}$$

Here, $T$ is an empirical threshold value. If $d_a \leq T$ for all attributes, the MLN is not re-trained. Lines 13 and 14 are skipped in the current iteration. Consequently, the current MLN weights are used in the next iteration for classification. However, if $d_a > T$ for any attribute $a$, then the current MLN is re-trained and $Q$ is updated. This is illustrated in the Figure 1 as Steps 2, 3 and 4.

## IV. Experiments and Results

In this section, we present the results of our analysis comparing MLN and HMLN classifiers with a few competing classifiers, using publicly available datasets.

### *DataSet*

We evaluate the proposed MLN classifier on streaming data using publicly available datasets [11]. A list of these datasets, along with relevant information, is given in Table I. The *SyntheticLED* dataset is generated, with 10% noise and concept drift, using the MOA toolkit [12]. A total of 100,000 data points are considered in each. A data stream is simulated by dividing the dataset into multiple data chunks.

### *Experiments*

The MLN and HMLN classifiers are evaluated by measuring its classification accuracy on the datasets. We use both supervised and semi-supervised learning methods for training the classifier with a set of data chunks. Supervised method assumes complete availability of class labels during weight learning. On the other hand, semi-supervised method uses class labels from only a small random set of data instances in each chunk during weight learning. We also perform various experiments using techniques discussed in Section III, including adaptive binning and selective weight learning.

| Dataset | Attributes Count | | | Classification Problem? |
|---|---|---|---|---|
| | Total | Discrete | Real-valued | |
| ForestCover | 55 | 45 | 10 | forest cover type |
| Airline | 8 | 6 | 2 | schedule delay |
| Poker | 11 | 11 | 0 | poker hand |
| SyntheticLED | 8 | 8 | 0 | digit displayed |

Table I: Publicly available Datasets

We use the Alchemy implementation [13] of inference and learning on MLN and HMLN. MC-SAT [8] and MAP (MaxWalkSAT) inference methods are used. Discriminative weight learning (using gradient descent), with the class attribute as non-evidence predicate, is used during training.

### *MLN Formula*

The MLN formula provided by a domain expert significantly affects the operational time of inference and learning in an MLN. Since we iteratively perform these operations, the overall time linearly increases with number of iterations. Classification of data instances involves prediction of a class label, given all other attributes as evidence. Therefore, we focus on MLN formulas that directly relates to the class attribute.

Table II lists examples of different sets of MLN formulas we use in our experiments. Note that the $+$ sign indicates that different weights are assigned for each grounding of the formula. Representing each data instance as an object $o$, the first formula relates an attribute value of each object to the class label for that object. In case of an HMLN classifier, this is represented by formula 2. Further, we use a hard constraint that specifies that an object can only belong to one class (listed as formula 3 in the table). These general formulas apply to all datasets. However, our empirical results showed that these formulas negatively affect the classification accuracy on some datasets. The table shows the datasets associated with each formula in order to obtain the best accuracy.

In addition, we also use domain-specific formulas that reflect the domain knowledge for each dataset, in an MLN. Table II lists an example of a domain-specific formula (No. 4 to 7) used for each dataset. For instance, the domain-specific formulas for the Poker dataset consists of all formulas that related to each poker hand (class label), similar to that shown in the table. These rules are derived from publicly available information. Similarly, for the SyntheticLED dataset, formulas are constructed for each of the 10 digits that can be displayed on an 7-segment LED.

| No. | Dataset | MLN Formula | Relation |
|---|---|---|---|
| 1 | All except Airline | $attr_j(o,+v) \implies Class(o,+c)$ | Attribute j of object o, having value v, related to class label c. |
| 2 | ForestCover & Airline | $Class(o,+c) \times (attr_j(v) = T)$ | Real-valued attribute j of object o, having value v, related to class label c. |
| 3 | ForestCover & Poker | $Class(o,c) \wedge c \neq c' \implies !Class(o,c').$ | Object o has one and only one class label (Hard constraint) |
| 4 | ForestCover | $Wilderness(o,3) \implies (Class(o,2) \vee Class(o,5))$ | Wilderness area of type 3 has cover of type 2 or type 5. |
| 5 | Airline | $AirportFrom(o,a) \wedge Distance(o,l) \wedge AirportTo(o,d) \wedge AirportFrom(o',a) \wedge Distance(o',l) \implies AirportTo(o',d)$ | Two airlines departing from an airport, traveling the same distance, have the same destination airport. |
| 6 | Poker | $SOne(o,s) \wedge STwo(o,s) \wedge SThree(o,s) \wedge SFour(o,s) \wedge SFive(o,s) \implies PokerHand(o,5)$ | Class label "Flush = 5" have 5 cards of the same suit |
| 7 | SyntheticLED | $Att3(o,1) \wedge Att6(o,1) \wedge Att1(o,0) \wedge Att4(o,0) \wedge Att7(o,0) \implies Class(o,1)$ | Display segments lit representing the numeric digit "1" are 3 and 6, with segments 1, 4 and 7 off. |

Table II: Examples of MLN Formulas

| | | Dataset (Chunk Size) | | | |
|---|---|---|---|---|---|
| | | ForestCover (500) | Airline (500) | Poker (750) | SyntheticLED (750) |
| Classifier | MLN | 13.59 | **35.04** | **8.86** | **25.94** |
| | HMLN | 26.87 | 37.4 | - | - |
| | SluiceBox [14] | 26.9 | 45.2 | 49.9 | 89.9 |
| | Hoeffding Tree | 13.6 | 40.97 | 48.27 | 28.4 |
| | NaïveBayes | 22.0 | 41.07 | 50.2 | 27.6 |
| | Perceptron | 21.4 | 46.43 | 48.27 | 26.8 |
| | SGD | 6.6 | 44.13 | 47.87 | 87.73 |
| | SingleClassifierDrift | 12.8 | 41.33 | 48.27 | 29.07 |
| | OzaBoostAdwin | **6.0** | 41.27 | 51.6 | 28.53 |
| | Accuracy-Updated-Ensemble2 [15] | 7.8 | 35.7 | 48.13 | 27.33 |

Table III: Comparison of Classification Error (%)

*Results*

Table III compares the average classification error obtained on all datasets when using MLN and HMLN classifiers, with other state-of-the-art classifiers. Here, SluiceBox [14] is an ensemble based classifier that combines Adaboost and Naïve Bayes for data classification, assuming complete availability of ground truth. We use the Massive Online Analysis framework (MOA) [12] for performing stream classification using other classifiers listed in the table, including a recent method [15] that uses weighted increment ensemble of classifiers. For each of these classifiers in MOA, we use a window performance evaluator with a window size equal to the chunk size for corresponding datasets, and with default settings. The results indicates that the MLN classifier performs significantly better than other classifiers on 3 out of 4 datasets. In particular, the MLN classifier outperforms all classifiers in case of the Poker dataset. This can be attributed to specific formulas provided as domain knowledge regarding each class label. Excluding any particular domain knowledge, and using only the general MLN formula on each dataset, the MLN classifier provided an average error of **16.88%**, **35.6%** and **50.1%** for the ForestCover, Airline and Poker dataset respectively. This shows that domain-specific formulas significantly impact the classification accuracy.

As the chunk size remains fixed throughout the evaluation process, its value may affect the accuracy of the classifier since an optimum size would represent a concept drift in the stream. This can then be captured in the model during the re-training process. We perform a grid search, starting from 100 data instances in each chunk, in an increment of 100 using the MLN and HMLN classifiers. The optimal chunk size of each dataset was found as given in Table III. For instance, a chunk size of 500 was found to be optimal in the case of the ForestCover and the Airline datasets since it yielded the highest accuracy.

Further, we evaluated the average error obtained when using an adaptive or standard binning techniques for datasets having real-valued or large domain size features. Here, standard binning technique refers to the assumption of prior knowledge of the largest and the smallest value for an attribute in the dataset. We obtained an error of **13.76%** when using the standard binning technique on the ForestCover dataset, and an error of **35.11%** when using the adaptive binning technique on the Airline dataset. When comparing these with Table III for MLN classifier, which uses adaptive binning for ForestCover and standard binning for Airline, the difference is negligible.

All experiments were performed using both MC-SAT and MAP inference techniques provided in Alchemy. The empirical evaluation suggested that MC-SAT performed better in the case of the Poker and SyntheticLED datasets, and MAP on the ForestCover and Airline datasets. The results shown in this section corresponds to the usage of inference technique that yields the best classification accuracy for each dataset.

Finally, we use the MLN classifier to perform stream classification in a semi-supervised setting. This can be used for reducing expensive manual annotation of data instances. Using only 10% of labels in the training data, a classification error of **36.06%**, **40.19%**, **32.62%** and **32.86%** was achieved on the ForestCover, Airline, Poker and SyntheticLED datasets respectively. In particular, note that this error obtained on the Poker dataset is better than any competing classifier shown in Table III, which uses a supervised training method. With increase in the number of labeled training data, we observe that the accuracy increases as expected. This indicates the effectiveness of using feature relationships for stream classification.

| Dataset | Without SL | | With Selective Learning (SL) | | |
|---|---|---|---|---|---|
| | Time (s) | Error (%) | Threshold (%) | Time (s) | Error (%) |
| ForestCover | 72.28 | 13.59 | 5 | 66.6 | 13.64 |
| | | | 10 | 61.4 | 13.81 |
| Airline | 8.09 | 35.04 | 5 | 7.94 | 34.96 |
| | | | 10 | 7.32 | 35.09 |
| Poker | 278.6 | 8.86 | 5 | 189.77 | 9.06 |
| | | | 10 | 189.95 | 9.06 |
| SyntheticLED | 85.03 | 25.94 | 5 | 60.19 | 25.99 |
| | | | 10 | 59.9 | 25.99 |

Table IV: Using Selective Learning (SL) on MLN

A major limitation of using MLNs for stream classification is the computational time required for weight learning and classification on each chunk. Table IV shows the time taken to process each chunk by an MLN classifier. Since MLNs are highly expressive, the size of the network can negatively affect the total time taken to perform classification on a data stream. We address this limitation, to a certain extent, by using the Selective Learning (SL) process with an empirically determined KL threshold. The table shows the average classification time for each data chunk when trained with and without the selective learning technique. It also provides the threshold used, and the corresponding classification error obtained on each dataset. The column "Time without SL" in the table can be viewed as the time taken for inference and learning while setting corresponding KL threshold to 0%. These results show substantial improvement in computational time elapsed for each chunk when using SL, with negligible change in classification error. The MLN classifier may be further improved by selecting good training data, online computations, and by learning the MLN structure from data. We leave these for future work.

## V. Conclusion

In this paper, we have shown an adaptation of MLNs for stream mining on data, using an incremental supervised and semi-supervised learning approach, leveraging feature relationships and domain knowledge to build the structure of a classifier. The use of MLNs to capture both implicit and explicit relationship between the attributes, using domain and relational knowledge, provides an advantage over other algorithms modeling relational data. We provide a detailed analysis and extensive experimental results comparing the performance of stream classification using MLN with state-of-the-art classifiers. In addition to the supervised learning approach, we also evaluate a semi-supervised learning approach where only a portion of the incoming stream data have true class labels. The experiment shows promising results of using a relational structure for classification.

## Acknowledgement

We thank anonymous reviewers for their invaluable feedback. This research is partially supported by the NSF Award No. CNS-1229652, and the Air Force Office of Scientific Research under Award No. FA-9550-12-1-0077.

## References

[1] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 226–235.

[2] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[3] R. Crane and L. K. McDowell, "Evaluating markov logic networks for collective classification," in *Proceedings of the 9th MLG Workshop at the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2011.

[4] L. Mihalkova and W. E. Moustafa, "Structure selection from streaming relational data," Tech. Rep., 2011.

[5] G. Widmer and M. Kubat, "Effective learning in dynamic environments by explicit context tracking," in *Machine Learning: ECML-93*. Springer, 1993, pp. 227–243.

[6] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[7] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, no. 1, pp. 273–302, 1996.

[8] H. Poon and P. Domingos, "Sound and efficient inference with probabilistic and deterministic dependencies," in *AAAI*, vol. 6, 2006, pp. 458–463.

[9] D. Lowd and P. Domingos, "Efficient weight learning for markov logic networks," in *Knowledge Discovery in Databases: PKDD 2007*. Springer, 2007, pp. 200–211.

[10] J. Wang and P. Domingos, "Hybrid markov logic networks." in *AAAI*, vol. 8, 2008, pp. 1106–1111.

[11] C. for Open Software Innovation, "Datasets in moa massive online analysis-real time analytics for data streams," http://moa.cms.waikato.ac.nz/datasets/.

[12] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[13] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, and D. Lowd, "The alchemy system for statistical relational ai," *University of Washington, Seattle*, 2005.

[14] B. Parker, A. M. Mustafa, and L. Khan, "Novel class detection and feature via a tiered ensemble approach for stream mining," in *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, vol. 1. IEEE, 2012, pp. 1171–1178.

[15] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 81–94, 2014.