A Method for Scalable First-Order Rule Learning on Twitter Data

Monica Senapati
University of Missouri-Kansas City
msenapati@mail.umkc.edu

Laurent Njilla
Air Force Research Lab (AFRL)
laurent.njilla@us.af.mil

Praveen Rao
University of Missouri-Kansas City
raopr@umkc.edu

***Abstract*—We propose a method for scalable first-order rule learning on large-scale Twitter data. By learning rules, probabilistic inference queries can be executed to reason over the data to ascertain its veracity. Our method employs a *divide-and-conquer* approach, graph-based modeling, and data parallel processing during rule learning using a commodity cluster to overcome the problem of slow structure learning on large-scale Twitter data. The first-order predicates (constructed on the posts) are first partitioned in a balanced way by pivoting around users to reduce the chance of missing relevant rules. By constructing a weighted graph and applying graph partitioning, balanced partitions of the ground predicates can be created. Each partition is then processed using an existing structure learning approach to get the set of rules for that partition. We report a preliminary evaluation of our method to show that it offers a promising solution for scalable first-order rule learning on Twitter data.**

I. Introduction

Social media sites have been widely used for political campaigns, marketing and advertising, sharing breaking news, and during a catastrophic event like an earthquake or a tsunami. Unfortunately, they have been exploited by adversaries to launch cyberattacks against users and their organizations. More recently, the spread of fake news has become a serious problem for Internet users [1]. Thus, ascertaining the veracity (or trustworthiness) of social media posts is a critical challenge for enterprises and users.

In statistical relational learning, a Markov logic network (MLN) [10] is regarded as one of the most flexible representations as it combines first-order logic and probabilistic graphical models. First-order logic enables the complexity of the data to be modeled; and probability allows expressing the uncertainty in the data. An MLN is a knowledge base (KB) and can model the complex, diverse nature of Twitter posts (a.k.a. tweets) containing 100+ attributes. Once the KB is learned, which includes the first-order rules (or formulas) and their weights, probabilistic inference can be conducted on the KB to reason about the content in the posts and users' behavior. For example, one can compute the marginal probability of a URL being malicious, a tweet being sensitive, a user account being an adversary or social bot [11], and so on. One approach is to use handcrafted first-order rules based on prior observations/knowledge [9]. Alternatively, the first-order rules can be learned over the data, which is appealing when the significance of a rule changes over time.

DISTRIBUTION A. Approved for public release: Distribution unlimited. Case Number 88ABW-2019-0278; Dated 24 Jan 2019

Although a number of methods have been proposed for MLN structure learning to learn first-order rules [4]–[7], [12], they have been tested only on small datasets using a single machine. Ontological pathfinding [2] is a recent work on scalable first-order rule mining using cluster computing. However, it learns only Horn clauses [2], whereas MLNs use general first-order rules, which are more expressive than Horn clauses. Thus, our goal is to scale MLN structure learning on large-scale Twitter data by (a) leveraging cluster computing and (b) exploiting the capability of existing structure learning tools [7] for learning rules on small datasets.

We propose a novel method called SRLearn (**S**calable **R**ule **Learn**ing) to scale the learning of relevant first-order rules over a large number of tweets using a commodity cluster. The salient features of SRLearn are as follows: (a) a *divide-and-conquer* approach to rule learning by first employing graph-based modeling of ground predicates and users, and applying graph partitioning to create partitions of ground predicates to minimize the chance of missing interesting rules; (b) exploiting data parallelism at different stages during rule learning using a commodity cluster; and (c) leveraging the power of Alchemy's structure learning [7] on small datasets. We report a preliminary evaluation of SRLearn on Twitter data to show that it is a better solution than running Alchemy in a centralized setting.

II. Background

A. Twitter Data

Tweets are short messages posted by users on Twitter. Each tweet collected from Twitter is assigned a unique ID; each user account is also assigned a unique ID. There are attributes whose values embed the actual text of a tweet, the URLs contained in a tweet, hashtags used in a tweet, users mentioned in a tweet, who retweeted a tweet, and so on. There are attributes that provide counts about the number of friends of a user, the number of followers of a user, the number of tweets liked/favorited by a user (i.e., favourites count), and the number of posts of a user (i.e., statuses count). Note that a tweet does not contain the list of friends or followers of a user. Nor does it contain information about hashtags that are trending. These pieces of information, however, can be obtained using Twitter APIs.

TABLE I
A SET OF FIRST-ORDER PREDICATES ON TWEETS

tweeted(userID, tweetID)	containsLink(tweetID, link)
containsHashtag(tweetID, hashtag)	malicious(link)
trending(hashtag)	verified(userID)
attacker(userID)	mentions(tweetID, userID)
retweeted(userID, tweetID)	isPossiblySensitive(tweetID)
retweetCount(tweetID, count)	friend(userID1, userID2)
isFollowedBy(userID1, userID2)	friendsCount(userID, count)
followersCount(userID, count)	statusesCount(userID, count)
favouritesCount(userID, count)	isBot(userID)

Table I shows a list of possible first-order predicates on tweets. (Other predicates can be defined.) The variables in the predicates are (a) *tweetID* denoting the ID of a tweet; (b) *userID* denoting the ID of a user; (c) *link* denoting a URL; (d) *hashtag* denoting a topic/word prefixed by the '#' symbol; and (e) *count* denoting a non-negative integer.

B. Markov Logic Networks

Formally, an MLN is a KB defined by a set of pairs (F,w), where F is a first-order rule that denotes a constraint and w is a real-valued weight of the rule. Higher the weight, more likely is the constraint believed to be satisfied in a possible world. A rule with a positive weight is more likely to be satisfied in a possible world; a rule with a negative weight is more likely to be unsatisfied. Two predicates that share at least one common variable can appear in an MLN's rule. For instance, $\forall u\ verified(u) \Rightarrow\ !isBot(u)$ is a candidate rule of size 2 on Twitter data that states every verified user u (as determined by Twitter) is not a social bot. *Thus, there are many candidates rules to test during MLN structure learning.* A grounding of a rule (or predicate) is obtained by replacing all its variables by constants resulting in a ground rule (or ground predicate). Once the rules and weights are learned [10], probabilistic inference queries can be posed.

III. OUR APPROACH

MLN structure learning [7] on a large number of ground predicates is time consuming especially in a centralized setting. To speed up and scale the learning of first-order rules on large datasets, our method, SRLearn, employs a *divide-and-conquer* approach, graph-based modeling of ground predicates constructed on tweets and users' actions, and data parallelism at different stages during structure learning.

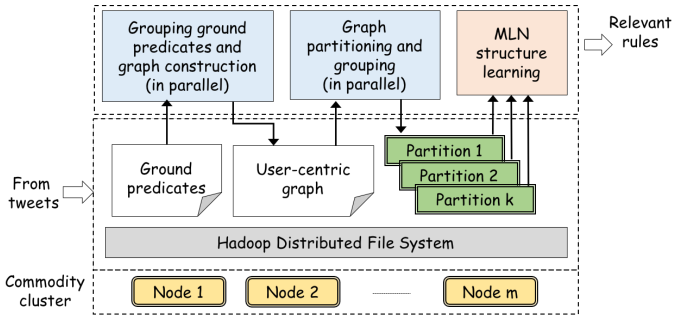

Fig. 1. Design of SRLearn

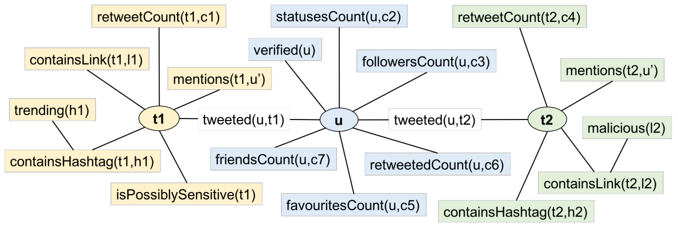

Fig. 2. An example of a ground predicate graph

Figure 1 shows the design of SRLearn, which operates on a commodity cluster, wherein the input dataset containing ground predicates is stored in the Hadoop Distributed File System (HDFS). The first challenge is to group the ground predicates by users. Essentially, we model the ground predicates associated with a user as a *ground predicate graph*. Suppose a user posted two tweets containing a link flagged as malicious and a hashtag that is trending. Figure 2 shows an example ground predicate graph constructed on 19 predicates. These graphs form the *building blocks* for the next step to generate the *user-centric graph*, which facilitates the divide-and-conquer approach and serves three purposes. First, a vertex of the user-centric graph denotes the entire ground predicate graph of a user and therefore, provides a natural way to cluster relevant ground predicates in the data. Its weight is based on the number of ground predicates in the ground predicate graph. Second, an edge in the user-centric graph indicates social interaction between two users, and its weight captures the extent of social interaction based on certain ground predicates that are present in the dataset. Third, by assigning weights to the vertices and edges of the user-centric graph, we can formulate the task of finding the optimal way to produce balanced partitions of ground predicates (for structure learning) as a graph partitioning task on a weighted graph by pivoting around the users. Another key advantage is that we can reduce the chance of missing relevant rules that can span across partitions during structure learning. As a result, the time taken for rule learning can be significantly reduced by operating on smaller sets of ground predicates (in parallel).

The key processing steps of SRLearn are described in Algorithms 1, 2, and 3, which are explained next. Algorithm 1 shows the steps involved in grouping the ground predicates for each user. To enable data parallelism, the MapReduce paradigm is employed [3]. First, an associative array is generated on the input dataset containing (tweetID, userID) pairs, and broadcast to all the nodes (Line 3). This is required for associating ground predicates that do not contain userID as a variable, to the right user. This is followed by the map phase (Lines 5-6) and the reduce phase (Lines 7-8). In the map phase, each block of ground predicates is processed by invoking MapPredicates$(\cdot)$ (Lines 11-27). Those predicates whose first argument is not of the type userID (e.g., $malicious(l)$, $containsHashtag(t,h)$) are processed specially in order to associate them with the right user in the reduce phase (Lines 15-24). The reduce phase invokes ReducePredicates$(\cdot)$ and generates key-value pairs where userID is the key, and the value is a group of predicates belonging to the ground predicate graph of that user (Lines 28-34). At this stage, there can be more than one key-value pair with the same key. So a reduce-by-key operation is executed to group all the ground predicates for a particular user into a key-value pair (Line 9).

Algorithm 1 Grouping the ground predicates by users

```
1: Let $F$ denote the HDFS file containing ground predicates
2: Let $G$, $H$, and $I$ denote distributed collections
3: Using map and reduce operations on $F$, construct an associate array $A$
   containing key-value pairs $(t, u)$, where $t$ is tweetID and $u$ is userID
4: Broadcast $A$ to all workers on nodes
   {/* Map phase */}
5: for each block $B_i$ of $F$ in HDFS do
6:     Invoke MapPredicates($B_i$)
   {/* Reduce phase */}
7: for each $(K, V)$ in $G$ do
8:     Invoke ReducePredicates($K, V$)
9: Run a reduce-by-key operation to process $H$ by grouping the tuples by
   userID and store the result into $I$
10: return $I$
   {/* Map function */}
11: begin func MapPredicates(block $B$)
12: for each ground predicate $p$ in $B$ do
13:     if $p$ is a predicate with userID $u$ as the first parameter then
14:         Output ($u$, $p$) to $G$
15:     if $p$ is of type malicious(link) or trending(hashtag) then
16:         Output ($p$, "EXISTS") to $G$
17:     if $p$ is of type containsLink(t, link)) then
18:         Output ("malicious(link)", t) to $G$
19:         Find $(t, \overline{u})$ in $A$; Output ($\overline{u}$, $p$) to $G$
20:     if $p$ is of type containsHashtag(t, hashtag)) then
21:         Output ("trending(hashtag)", t) to $G$
22:         Find $(t, \overline{u})$ in $A$; Output ($\overline{u}$, $p$) to $G$
23:     if $p$ is of type mentions(t, u') or isPossiblySensitive(t) or retweet-
        Count(t, c) then
24:         Find $(t, \overline{u})$ in $A$; Output ($\overline{u}$, $p$) to $G$
25:     if $p$ is of type retweeted(u, t) and $(t, \overline{u})$ exists in $A$ then
26:         Output ($\overline{u}$, "retweetedBy(t, u)") to $G$
27: end func
   {/* Reduce function */}
28: begin func ReducePredicates($K, V$)
29: if $K$ is a userID then
30:     Output($K, V$) to $H$
31: else if any item in $V$ = "EXISTS" then
32:     for each item $t' \neq$ "EXISTS" in $V$ do
33:         Find $(t', \overline{u})$ in $A$; Output($\overline{u}, K$) to $H$
34: end func
```

Algorithm 2 shows the steps involved in constructing the user-centric graph. It has a map phase (Lines 3-4) and a reduce phase (Lines 5-6). In the map phase, MapWts($\cdot$) is invoked on each (K, V), where K is a userID and V denotes all the ground predicates for that userID (Lines 9-23). First, the vertex ID and its weight are output (Lines 10-11). (The vertex weight is the number of predicates in V.) Next, the social interaction between two users is determined based on the rules in Table II. That is, we define an edge between two users u and u' in the graph if (a) u mentions u' or vice-versa, (b) u has a friend relationship with u' or vice-versa, (c) u is followed by u' or vice-versa, or (d) u retweets a tweet of u' or vice-versa. Each of the above conditions defines an edge with unit weight. Partial edge weights are computed and output along with the edge (Lines 12-22). The partial edge weights for the same edge from different map operations are combined in the reduce phase by ReduceWts($\cdot$) (Lines 24-30). The weighted graph is then stored in HDFS (Line 7).

Algorithm 2 Constructing the user-centric graph

```
1: Let $I$ denote the output of Algorithm 1
2: Let $J$ and $L$ denote distributed collections
   {/* Map phase */}
3: for each $(K, V)$ in $I$ do
4:     Invoke MapWts($K, V$)
   {/* Reduce phase */}
5: for each $(K, V)$ in $J$ do
6:     Invoke ReduceWts($K, V$)
7: Store $L$ as a file to HDFS
8: return
   {/* Map function */}
9: begin func MapWts($K, V$)
10: Let $c$ denote the no. of predicates in $V$ excluding retweetedBy
11: Output $(K, c)$ to $J$
12: $edgeList \leftarrow \emptyset$
13: for each $p$ in $V$ do
14:     if $p$ is of type friend(u, u') or isFollowedBy(u, u') or mentions(t, u')
        or retweetedBy(t, u') then
15:         $x \leftarrow min(u, u')$; $y \leftarrow max(u, u')$
16:         Create edge $(x, y)$
17:         if $(x, y) \in edgeList$ then
18:             Increase edge weight $w_{x,y}$ by 1
19:         else
20:             Add $(x, y)$ to $edgeList$ with $w_{x,y} = 1$
21: for each $(x, y)$ in $edgeList$ do
22:     Output $((x, y), w_{x,y})$ as the edge to $J$
23: end func
   {/* Reduce function */}
24: begin func ReduceWts($K, V$)
25: if $K$ is a userID then
26:     Output $(K, V)$ to $L$ {/* weighted vertex */}
27: else
28:     Let $\overline{w}$ denote the sum of all wt. values in $V$
29:     Output $(K, \overline{w})$ to $L$ {/* weighted edge */}
30: end func
```

TABLE II
SOCIAL RELATIONSHIPS BETWEEN USERS

Condition	Edge(u, u')
tweeted(u, t) $\wedge$ mentions(t, u')	TRUE
tweeted(u', t') $\wedge$ mentions(t', u)	TRUE
friend(u, u') $\vee$ friend(u', u)	TRUE
isFollowedBy(u, u') $\vee$ isFollowedBy(u', u)	TRUE
tweeted(u, t) $\wedge$ retweeted(u', t)	TRUE
tweeted(u', t) $\wedge$ retweeted(u, t)	TRUE

Algorithm 3 shows the divide-and-conquer approach of SRLearn. The user-centric graph is partitioned into some number of partitions by minimizing the total weight of the cut edges (Line 4). The intuition is to group predicates of users that have high degree of social interaction, in the same partition so that interesting rules can be discovered between them. Alternatively, users with low degree of social interaction can be placed in different partitions without missing interesting rules spanning across partitions. The partitions are assigned to different machines (e.g., round-robin assignment), and structure learning [7] is executed independently on each partition to learn the rules (Lines 5-9).

IV. Preliminary Evaluation

For our implementation, we used Scala 2.11.8, Apache Spark 2.3.1[1], and Apache Hadoop 2.7.6[2]. We conducted the

[1] http://spark.apache.org [2] http://hadoop.apache.org

Algorithm 3 Graph partitioning and rule learning in parallel

```
1: Let $U$ denote the user-centric graph file output by Algorithm 2
2: Let $m$ denote the number of required partitions
3: Let $n$ denote the number of machines
4: Run graph partitioning (in parallel) on $U$ to generate partitions $p_1, \ldots, p_m$ of the ground predicates (in $F$) by minimizing the total weight of the cut edges in $U$
5: Assign partitions in a round-robin fashion to the $n$ machines
6: In parallel, invoke StructureLearning() on each node
7: Wait for all the nodes to finish
8: Let $R$ denote the union of all the learned rules
9: return $R$
10: begin func StructureLearning()
11: for each partition $p_i$ on the node do
12:     Run Alchemy's structure learning algorithm [7] on $p_i$
13: end func
```

experiments on CloudLab[3] by setting up a cluster of physical nodes. Each node had a 10-core Intel processor with 64 GB RAM and 480 GB SSD. We collected 20,000 tweets during 2016-2017 and created an input file with 22.4 million ground predicates containing 17 unique predicates (except $isBot()$) as shown in Table I. The file size was 741 MB.

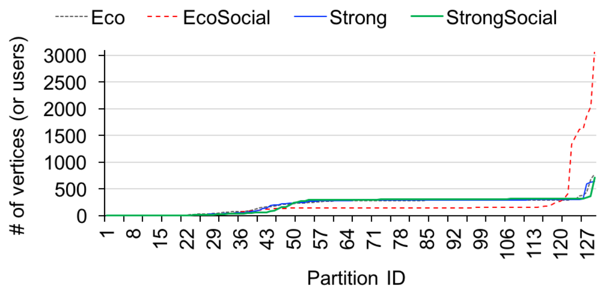

Fig. 3. Quality of graph partitioning

We first ran Alchemy's structure learning approach [7] on a single machine. Unfortunately, it was very slow even on a small number of ground predicates. It ran for 138 hours on a meager 15,417 ground predicates and eventually failed due to a memory allocation error. *This clearly underscores the problem of MLN structure learning in a centralized setting.*

Next, we present insights on SRLearn's performance on 22.4 million ground predicates. We first ran SRLearn on a 16-node cluster to produce 128 partitions of the ground predicates. This required a total of 2 hours and 30 minutes. For graph partitioning, we used KaHIP [8] and tested different strategies that provide a tradeoff between partition quality and time for different types of graphs. Figure 3 shows the quality of partitioning in terms of the number of vertices (i.e., users) of the user-centric graph per partition in sorted order. Compared to the *Eco*, *EcoStrong*, and *Strong* settings in KaHIP, *StrongSocial* produced the most balanced partitions.

Next, we ran Alchemy's structure learning on each partition to generate rules using a 32-node cluster. The 128 partitions were distributed across 32 nodes. It took a total of 66 hours and 33 minutes to run the structure learning on the partitions. (Although a node had many cores, we ran Alchemy on the partitions serially on the node.) Some of the learned rules

[3] http://cloudlab.us

TABLE III
SOME RULES LEARNED BY SRLEARN

First-order rules (in clausal form)
`friend(v0,v1) ∨ !verified(v0)`
`!mentions(v0,v1) ∨ !retweeted(v2,v3) ∨ !tweeted(v1,v3) ∨ !tweeted(v2,v0)`
`!containsHashtag(v0,v1) ∨ containsHashtag(v2,v1) ∨ !mentions(v0,v3) ∨ !mentions(v2,v3)`

are shown in Table III. Thus, SRLearn provides a promising solution to reduce the computational complexity of first-order rule learning on large-scale Twitter data.

V. Conclusion and Future Work

We presented SRLearn for scalable first-order rule learning on Twitter data by employing a divide-and-conquer approach, graph-based modeling of tweets and users' actions, and data parallelism. Our method can be adapted to other social media sites by generating a different set of first-order predicates based on the posts and users, and defining social interactions. In the future, we plan generalize the algorithms for any type of predicate and develop a method to combine the rules from different partitions (e.g., by learning their weights on the ground predicates) and evaluate their quality for inference.

Acknowledgments

Praveen Rao would like to thank the support of AFRL's Visiting Faculty Research Program & Extension Grant Program.

References

[1] H. Allcott and M. Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–36, May 2017.

[2] Y. Chen, S. Goldberg, D. Z. Wang, and S. S. Johri. Ontological Pathfinding: Mining First-Order Knowledge from Large Knowledge Bases. In *Proc. of the 2016 SIGMOD Conference*, pages 835–846, San Francisco, California, USA, 2016.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.

[4] T. N. Huynh and R. J. Mooney. Online Structure Learning for Markov Logic Networks. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, pages 81–96, Athens, Greece, 2011.

[5] H. Khosravi, O. Schulte, T. Man, X. Xu, and B. Bina. Structure Learning for Markov Logic Networks with Many Descriptive Attributes. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 487–493, Atlanta, Georgia, 2010.

[6] S. Kok and P. M. Domingos. Learning Markov Logic Network Structure via Hypergraph Lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 505–512, Montreal, 2009.

[7] S. Kok and P. M. Domingos. Learning Markov Logic Networks Using Structural Motifs. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 551–558, Haifa, Israel, 2010.

[8] H. Meyerhenke, P. Sanders, and C. Schulz. Parallel Graph Partitioning for Complex Networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2017.

[9] P. Rao, A. Katib, C. Kamhoua, K. Kwiat, and L. Njilla. Probabilistic Inference on Twitter Data to Discover Suspicious Users and Malicious Content. In *Proc. of the 2nd IEEE Intl. Symp. on Security and Privacy in Social Networks and Big Data*, pages 407–414, Nadi, Fiji, 2016.

[10] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, Feb. 2006.

[11] C. Shao, G. L. Ciampaglia, O. Varol, A. Flammini, and F. Menczer. The Spread of Fake News by Social Bots. *CoRR*, abs/1707.07592, 2017.

[12] Z. Sun, Y. Zhao, Z. Wei, W. Zhang, and J. Wang. Scalable Learning and Inference in Markov Logic Networks. *Int. J. Approx. Reasoning*, 82(C):39–55, Mar. 2017.