

Learning Markov Network Structure with Decision Trees

Daniel Lowd* and Jesse Davis†

*Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
Email: lowd@cs.uoregon.edu

† Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, POBox 2402, 3001 Heverlee, Belgium
Email: jesse.davis@cs.kuleuven.be

Abstract—Traditional Markov network structure learning algorithms perform a search for globally useful features. However, these algorithms are often slow and prone to finding local optima due to the large space of possible structures. Ravikumar et al. [1] recently proposed the alternative idea of applying L1 logistic regression to learn a set of pairwise features for each variable, which are then combined into a global model. This paper presents the DTSL algorithm, which uses probabilistic decision trees as the local model. Our approach has two significant advantages: it is more efficient, and it is able to discover features that capture more complex interactions among the variables. Our approach can also be seen as a method for converting a dependency network into a consistent probabilistic model. In an extensive empirical evaluation on 13 datasets, our algorithm obtains comparable accuracy to three standard structure learning algorithms while running 1-4 orders of magnitude faster.

Keywords-Markov networks; structure learning; decision trees; probabilistic methods

I. INTRODUCTION

Markov networks are an undirected, probabilistic graphical model for compactly representing a joint probability distribution over set of variables. Traditional Markov network structure learning algorithms perform a global search to learn a set of features that accurately captures high-probability regions of the instance space. These approaches are often slow for two reasons. First, the space of possible structures is exponential in the number of variables. Second, evaluating candidate structures requires assigning a weight to each feature in the model. Weight learning requires performing inference over the model, which is often intractable.

Recently, Ravikumar et al. [1] proposed the alternative idea of learning a local model for each variable and then combining these models into a global model. Their method builds an L1 logistic regression model to predict the value of each variable in terms of all other variables. Next, it constructs a pairwise feature between the target variable and each other variable with non-zero weight in the L1 logistic regression model. Finally, it adds all constructed features to the model and learns their associated weights using any standard weight learning algorithm. While this approach greatly improves the tractability of structure learning, it

is limited to modeling pairwise interactions, ignoring all higher-order effects. Furthermore, it still exhibits long run times for domains that have large numbers of variables.

In this paper, we propose DTSL (Decision Tree Structure Learner), which builds on the approach of Ravikumar et al. by substituting a probabilistic decision tree learner for L1 logistic regression. Probabilistic decision trees can represent much richer structures that model interactions among large sets of variables. DTSL learns probabilistic decision trees to predict the value of each variable and then converts the trees into sets of conjunctive features. We propose and evaluate several different methods for performing the conversion. Finally, DTSL merges all learned features into a global model. Weights for these features can be learned using any standard Markov network weight learning method.

We conducted an extensive empirical evaluation on 13 real-world datasets. We found that DTSL is 1-4 orders of magnitude faster than alternative structure learning algorithms while still achieving equivalent accuracy.

The remainder of our paper is organized as follows. Section 2 provides background on Markov networks. Section 3 describes our method for learning Markov networks using decision trees. Section 4 presents the experimental results and analysis and Section 5 contains conclusions and future work.

II. MARKOV NETWORKS

A. Representation

A *Markov network* is a model for the joint distribution of a set of variables $X = (X_1, X_2, \dots, X_n)$ [2]. It is composed of an undirected graph G and a set of potential functions ϕ_k . The graph has a node for each variable, and the model has a potential function for each clique in the graph. The joint distribution represented by a Markov network is:

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

where $x_{\{k\}}$ is the state of the k th clique (i.e., the state of the variables that appear in that clique), and Z is a normalization constant called the *partition function*. Markov networks are often conveniently represented as *log-linear models*, with

each clique potential replaced by an exponentiated weighted sum of features of the state:

$$P(X=x) = \frac{1}{Z} \exp \left(\sum_j w_j f_j(x) \right) \quad (2)$$

A feature $f_j(x)$ may be any real-valued function of the state. For discrete data, a feature typically is a conjunction of tests of the form $X_i = x_i$, where X_i is a variable and x_i is a value of that variable. We say that a feature matches an example if it is true of that example.

B. Inference

The main inference task in graphical models is to compute the conditional probability of some variables (the query) given the values of some others (the evidence), by summing out the remaining variables. This problem is #P-complete. Thus, approximate inference techniques are required. One widely used method is Markov chain Monte Carlo (MCMC) [3], and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its *Markov blanket*, the variables it appears with in some potential. These samples can be used to answer probabilistic queries by counting the number of samples that satisfy each query and dividing by the total number of samples. Under modest assumptions, the distribution represented by these samples will eventually converge to the true distribution. However, convergence may require a very large number of samples, and detecting convergence is difficult.

C. Weight Learning

The goal of weight learning is to select feature weights that maximize a given objective function. One of the most popular objective functions is the log likelihood of the training data. In a Markov network, log likelihood is a convex function of the weights, and thus weight learning can be posed as a convex optimization problem. However, this optimization typically requires evaluating the log likelihood and its gradient in each iteration. This is typically intractable to compute exactly due to the partition function. Furthermore, an approximation may work poorly: Kulesza and Pereira [4] have shown that approximate inference can mislead weight learning algorithms.

A more efficient alternative, widely used in areas such as spatial statistics, social network modeling and language processing, is to optimize the pseudo-likelihood [5] instead. Pseudo-likelihood is the product of the conditional probabilities of each variable given its Markov blanket:

$$\log P_w^\bullet(X=x) = \sum_{j=1}^V \sum_{i=1}^N \log P_w(X_{i,j}=x_{i,j} | MB_x(X_{i,j})) \quad (3)$$

where V is the number of variables, N is the number of examples, $x_{i,j}$ is the value of the j th variable of the i th example, $MB_x(X_{i,j})$ is the state of $X_{i,j}$'s Markov

blanket in the data. Pseudo-likelihood and its gradient can be computed efficiently and optimized using any standard convex optimization algorithm, since the pseudo-likelihood of a Markov network is also convex.

D. Structure Learning

Della Pietra et al.'s algorithm [2] is the standard approach to learning the structure of a Markov network. The algorithm starts with a set of atomic features (i.e., just the variables in the domain). It creates candidate features by conjoining each feature to each other feature, including the original atomic features. It calculates the weight for each candidate feature by assuming that all other feature weights remain unchanged, which is done for efficiency reasons. It uses Gibbs sampling for inference when setting the weight. Then, it evaluates each candidate feature f by estimating how much adding f would increase the log-likelihood. It adds the feature that results in the largest gain to the feature set. This procedure terminates when no candidate feature improves the model's score.

Recently, Davis and Domingos [6] proposed an alternative bottom-up approach, called BLM, for learning the structure of a Markov network. BLM starts by treating each complete example as a long feature in the Markov network. The algorithm repeatedly iterates through the feature set. It considers generalizing each feature to match its k nearest previously unmatched examples by dropping variables. If incorporating the newly generalized feature improves the model's score, it is retained in the model. The process terminates when no generalization improves the score.

A recent L1 regularization based approach to structure learning is the method of Ravikumar et al. [1]. It learns the structure by trying to discover the Markov blanket of each variable (i.e., its neighbors in the network). It considers each variable X_i in turn and builds an L1 logistic regression model to predict the value of X_i given the remaining variables. L1 regularization encourages sparsity, so that most of the variables end up with a weight of zero. The Markov blanket of X_i is all variables that have non-zero weight in the logistic regression model. In the limit of infinite data, consistency is guaranteed (i.e., X_i is in X_j 's Markov blanket if and only if X_j is in X_i 's Markov blanket). In practice, this is often not the case and there are two methods to decide which edges to include in the network. One includes an edge if either X_i is in X_j 's Markov blanket or X_j is in X_i 's Markov blanket. The other includes an edge if both X_i is in X_j 's Markov blanket and X_j is in X_i 's Markov blanket. Finally, all features are added to the model and their weights are learned globally using any standard weight learning algorithm.

III. ALGORITHM

We now describe our method for learning Markov network structure from data, DTSL (Decision Tree Structure

Table I
THE DTSL ALGORITHM

```

function DTSL( $D, \mathbf{X}$ )
 $F \leftarrow \emptyset$ 
for  $X_i \in \mathbf{X}$  do
   $T_i \leftarrow \text{LEARNTREE}(X_i, D)$ 
   $F_i \leftarrow \text{GENERATEFEATURES}(T_i)$ 
   $F \leftarrow F \cup F_i$ 
end for
 $M \leftarrow \text{LEARNWEIGHTS}(F, D)$ 
return  $M$ 

```

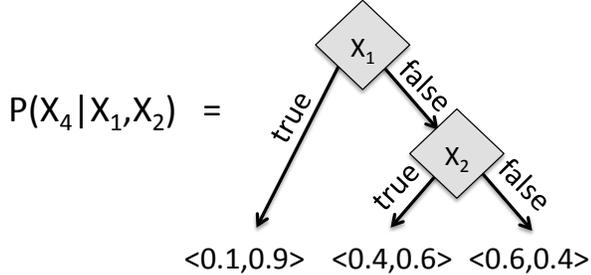


Figure 1. Example of a probabilistic decision tree.

Learning). Table I outlines our basic approach. For each variable X_i , we learn a probabilistic decision tree to represent the conditional probability of X_i given all other variables, $P(X_i | \mathbf{X} - X_i)$. Each tree is converted to a set of conjunctive features capable of representing the same probability distribution as the tree. Finally, all features are taken together in a single model and weights are learned globally using any standard weight learning algorithm.

This is similar in spirit to learning a dependency network [7]: Both dependency networks (with tree distributions) and DTSL learn a probabilistic decision tree for each variable and combine the trees to form a probabilistic model. However, a dependency network may not represent a consistent probability distribution, and inference can only be done by Gibbs sampling. In contrast, the Markov networks learned by DTSL always represent consistent probability distributions and allow inference to be done by any standard technique, such as loopy belief propagation [8], mean field, or MCMC.

We now describe each step of DTSL in more detail.

A. Learning Trees

A probabilistic decision tree represents a probability distribution over a target variable, X_i , given a set of inputs. Each interior node tests the value of an input variable and each of its outgoing edges is labeled with one of the outcomes of that test (e.g., true or false). Each leaf node contains the conditional distribution (e.g., multinomial) of the target variable given the test outcomes specified by its ancestor nodes and edges in the tree. We focus on discrete variables and consider tests of the form $X_j = x_j$, where

Table II
DTSL TREE LEARNING SUBROUTINE

```

function LEARNTREE( $X_i, D$ )
 $best\_split \leftarrow \emptyset$ 
 $best\_score \leftarrow 0$ 
for  $X_j \in \mathbf{X} - X_i$  do
  for  $x_j \in \text{Val}(X_j)$  do
     $S \leftarrow (X_j = x_j)$ 
    if  $\text{SCORE}(S, X_i, D) > best\_split$  then
       $best\_split \leftarrow S$ 
       $best\_score \leftarrow \text{SCORE}(S, X_i, D)$ 
    end if
  end for
end for
if  $best\_score > \log \kappa$  then
  ( $D_t, D_f$ )  $\leftarrow \text{SPLITDATA}(D, best\_split)$ 
   $T_L \leftarrow \text{LEARNTREE}(X_i, D_t)$ 
   $T_R \leftarrow \text{LEARNTREE}(X_i, D_f)$ 
  return new TreeVertex( $best\_split, T_L, T_R$ )
else
  Use  $D$  to estimate  $P(X_i)$ 
  return new TreeLeaf( $P(X_i)$ )
end if

```

X_j is a variable and x_j is value of that variable. Each conditional distribution is represented by a multinomial. Figure 1 contains an example of a probabilistic decision tree.

We can learn a probabilistic decision tree from data in a depth-first manner, one split at a time. We select a split at the root, partition the training data into the sets matching each outgoing branch, and recurse. We select each split to maximize the conditional log-likelihood of the target variable. This is very similar to using information gain as the split criterion. We used multinomials as the leaf distributions with a Dirichlet prior ($\alpha = 1$) for smoothing. In order to help avoid overfitting, we used a structure prior $P(S) \propto \kappa^p$, where p is the number of parameters, as in Chickering et al. [9].

Pseudocode for the tree learning subroutine is in Table II.

B. Generating Features

While decision trees are not commonly thought of as a log-linear model, any decision tree can be converted to a set of conjunctive features. In addition to a direct translation (DEFAULT), we explored four modifications (PRUNE, PRUNE-10, PRUNE-5, and NONZERO) which could yield structures with easier weight learning or better generalization.

The DEFAULT feature generation method is a direct translation of a probabilistic decision tree to an equivalent set of features. For each leaf in the decision tree, we generate a rule for each state of the target variable, containing a condition for each ancestor in the decision tree. For example, to convert the decision tree in Figure 1 to a set of rules, we generate two features for each leaf, one where X_4 is true and one where X_4 is false. The complete list of features is as follows:

- 1) $X_1 = T \wedge X_4 = T$

- 2) $X_1 = T \wedge X_4 = F$
- 3) $X_1 = F \wedge X_2 = T \wedge X_4 = T$
- 4) $X_1 = F \wedge X_2 = T \wedge X_4 = F$
- 5) $X_1 = F \wedge X_2 = F \wedge X_4 = T$
- 6) $X_1 = F \wedge X_2 = F \wedge X_4 = F$

By using the log probability at the leaf as the rule’s weight, we obtain a log linear model representing the same distribution. By applying this transformation to all decision trees, we obtain a set of conjunctive features that comprise the structure of our Markov network. However, their weights may be poorly calibrated (e.g., due to the same feature appearing in several decision trees), so weight learning is still necessary.

The PRUNE method expands the set of features generated by DEFAULT in order to make learning and inference easier. One disadvantage of the DEFAULT procedure is that it generates very long features with many conditions when the source trees are deep. Intuitively, we would like to capture the coarse interactions with short features and the finer interactions with longer features, rather than representing everything with long features. In the PRUNE method, we generate additional features for each path from the root to an interior node, not just paths from the root to a leaf. This is equivalent to applying the default feature generation method to all possible “pruned” versions of a decision tree, that is, where one or more interior nodes are replaced with leaves. This yields two additional rules, in addition to those enumerated above:

- 1) $X_1 = F \wedge X_4 = T$
- 2) $X_1 = F \wedge X_4 = F$

The PRUNE-10 and PRUNE-5 methods extend PRUNE by limiting the tree to a maximum depth of 10 and 5, respectively. This can help avoid overfitting.

Our final feature generation method, NONZERO, is similar to DEFAULT, but removes all false variable constraints in a post-processing step. For example, the decision tree in Figure 1 would be converted to the following set of rules:

- 1) $X_1 = T \wedge X_4 = T$
- 2) $X_1 = T$
- 3) $X_2 = T \wedge X_4 = T$
- 4) $X_2 = T$
- 5) $X_4 = T$

This simplification is designed for sparse binary domains such as text, where a value of false or zero contains much less information than a value of true or one.

C. Asymptotic Complexity

Let n be the number of variables, m be the number of training examples, and l be the number of values per variable. The complexity of selecting the first split is $O(lmn)$, since we must compute statistics for each of the l values of each of the n variables using all of the m examples. At the next level, we now have two splits to select: one

for the left child and one for the right child of the original split. However, since the split partitions the training data into two sets, each of the m examples is only considered once, either for the left split or the right split, leading to a total time of $O(lmn)$ at each level. If each split assigns a fraction of at least $1/k$ examples to each child, then the depth is at most $O(\log_k(m))$, yielding a total complexity of $O(lmn \log_k(m))$ for one tree, and $O(lmn^2 \log_k(m))$ for the entire structure. Depending on the patterns present in the data, the depth of the learned trees could be much less than $\log_k(m)$, leading to faster run times in practice. For large datasets or streaming data, we can apply the Hoeffding tree algorithm instead [10], which uses the Hoeffding bound to select decision tree splits after enough data has been seen to make a confident choice, rather than using all available data.

IV. EMPIRICAL EVALUATION

We evaluate our approach on 13 real-world datasets. The goals of our experiments are two-fold. First, we want to compare the run time and accuracy of DTSL to several other state-of-the-art Markov network structure learners: the algorithm of Della Pietra et al. [2], which we refer to as DP; BLM [6]; and L1 regularized logistic regression [1]. Second, we want to evaluate whether DTSL’s pruning heuristics result in more accurate models.

A. Methods

We used DTSL and each of the baselines to learn structures for all 13 datasets.

DTSL was implemented in OCaml. For both BLM and DP, we used the publicly available code of Davis and Domingos [6]. For Ravikumar et al.’s approach, we used the OWL-QN software package [11] for performing the L1 logistic regression.

The output of each structure learning algorithm is a set of conjunctive features. To learn weights, we optimized the pseudo-likelihood of the data via the limited-memory BFGS algorithm [12] since optimizing the likelihood of the data is prohibitively expensive for the domains we consider.

Like Lee et al. [13], we evaluated our algorithm using test set conditional marginal log-likelihood (CMLL). Calculating the CMLL required dividing the variables into a query set Q and an evidence set E . Then, for each test example we computed $CMLL(X = x) = \sum_{i \in Q} \log P(X_i = x_i | E)$. For each domain, we divided the variables into four disjoint groups. One set served as the query variables while the remaining three sets served as evidence. We repeated this procedure such that each set served as the query variables. We computed the conditional marginal probabilities using the MC-SAT inference algorithm [14]. For all three domains, we set the burn-in to 1,000 samples and then computed the probability using the next 10,000 samples.

Table III
DATA SET CHARACTERISTICS

Data Set	Train Set Size	Tune Set Size	Test Set Size	Num. Vars.	Density
NLTCS	16,181	2,157	3,236	16	0.332
MSNBC	291,326	38,843	58,265	17	0.166
KDDCup 2000	180,092	19,907	34,955	64	0.008
Plants	17,412	2,321	3,482	69	0.180
Audio	15,000	2,000	3,000	100	0.198
Jester	9,000	1,000	4,116	100	0.610
Netflix	15,000	2,000	3,000	100	0.541
MSWeb	29,441	3,270	5,000	294	0.010
Book	8,700	1,159	1,739	500	0.016
EachMovie	4,524	1,002	591	500	0.058
WebKB	2,803	558	838	843	0.063
20 Newsgroups	11,293	3,764	3,764	930	0.049
Reuters-52	6,532	1,028	1,540	941	0.037

We tuned all algorithms using separate validation sets, the same validation sets used by Davis and Domingos. For DTSL, we selected the structure prior κ for each domain that minimized the total log loss of all probabilistic decision trees on the validation set. The values of κ we used were powers of 10, ranging from 0.0001 to 1.0. For each feature generation method, we then tuned the Gaussian weight prior to maximize CMLL on the validation set, with values of 100, 10, 1, and 0.1. For comparisons to other algorithms, we selected the DTSL model with the best overall CMLL score on the validation set.

For L1, on each dataset we tried the following values of the prior λ : 1, 2, 5, 10, 25, 50, 100, 200, 500, and 1000. We also tried both methods of making the Markov blankets consistent, and tuned the weight prior as we did with DTSL. (Tuning the Gaussian weight prior allowed us to get better results than reported by Davis and Domingos [6].)

For BLM and DP, we kept the tuning settings used by Davis and Domingos. Additional tuning of the weight prior might lead to slightly better results.

All of our code is available at <http://ix.cs.uoregon.edu/~lowd/dtssl> under a modified BSD license.

B. Data Sets

For our experiments, we selected a subset of the domains used by Davis and Domingos [6].¹ We excluded the domains that had multivalued variables encoded as multiple binary variables, since this leads to artificial deterministic dependencies. Table III describes the characteristics of each dataset. Datasets are listed in increasing order by number of variables.

From the UCI machine learning repository [15] we used: KDDCup 2000 data, MSNBC anonymous web data, MSWeb anonymous web data and Plants domains. The KDD Cup 2000 clickstream prediction data set [16] consists of web session data taken from an online retailer. Using the subset of

Hulten and Domingos [17], each example initially consisted of 65 Boolean variables, corresponding to whether or not a particular session visited a web page matching a certain category. We dropped one variable that was always set to zero in the training data. The MSNBC anonymous web data contains information about whether a user visited a top-level MSNBC page during a particular session. We created one variable for each page, which is true if the user visited that particular page during a session. The MSWeb anonymous web data contains visit data for 294 areas (Vroots) of the Microsoft web site, collected during one week in February 1998. Again, we created one variable for each page, which is true if the user visited that particular page during a session. The Plants dataset consists of different plant types and locations where they are found. We constructed one binary feature for each location, which is true if the plant is found there.

The National Long Term Care Survey (NLTCS) data consist of binary variables that measure an individual’s ability to perform different daily living activities.²

We used three text domains: 20 Newsgroups, Reuters-52 and WebKB.³ For 20 Newsgroups, we only considered words that appeared in at least 200 documents. For Reuters and WebKB, we only considered words that appeared in at least 50 documents. For all three datasets, we created one binary feature for each word. The text domains contained roughly a 50-50 train-test split, whereas all other domains used around 75% of the data for the training, 10% for tuning, and 15% for testing. Thus we split the test set of these domains to make the proportion of data devoted to each task more closely match the other domains used in the empirical evaluation.

Finally, we considered several collaborative filtering problems: Audio, Book, EachMovie, Jester and Netflix. The Audio dataset consists of information about how often a user listened-to a particular artist.⁴ The data was provided by the company Audioscrobbler before it was acquired by Last.fm. We focused on the 100 most listened-to artists. We used a random subset of the data and reduced the problem to “listened-to” or “did not listen-to.” The Book Crossing (Book) dataset [18] consists of a users rating of how much they liked a book. We considered the 500 most frequently rated book. We reduced the problem to “rated” or “not rated” and considered all people who rated more than of these books. EachMovie⁵ is a collaborative filtering dataset in which users rate movies they have seen. We focused on the 500 most-rated movies, and reduced each variable to “rated” or “not rated”. The Jester dataset [19] consists of users’ real-valued ratings for 100 jokes. For Jester we

²<http://lib.stat.cmu.edu/datasets/>

³<http://web.ist.utl.pt/~acardoso/datasets/>

⁴http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html

⁵Provided by Compaq at <http://research.compaq.com/SRC/eachmovie/>; no longer available for download, as of October 2004.

¹Publicly available at <http://alchemy.cs.washington.edu/papers/davis10a>

selected all users who had rated all 100 jokes, and reduced their preferences to “like” and “dislike” by thresholding the real-valued preference ratings at zero. Finally, we considered a random subset of the Netflix challenge data and focused on the 100 most frequently rated movies and reduced the problem to “rated” or “not-rated.”

C. Results

First, we compared the accuracy of the different DTSL feature generation methods: DEFAULT, PRUNE, PRUNE-5, PRUNE-10, and NONZERO. Results are in Table IV. On 11 out of 13 datasets, PRUNE is more accurate than DEFAULT, sometimes substantially so. PRUNE-10 and PRUNE-5 sometimes improve on the accuracy of PRUNE when PRUNE seems to be overfitting, such as on MSWeb and the text datasets (WebKB, 20 Newsgroups, Reuters-52). NONZERO worked surprisingly well, yielding the most accurate models on eight out of 13 datasets. Its use of fewer and shorter features may be key to avoiding overfitting. Overall, PRUNE or NONZERO is almost always the best choice, yielding the best CMLL on 11 datasets and a close second on the remaining two.

Additional characteristics of the features generated by each method are in Table V. “Average Feature Length” is the average number of conditions per feature. The PRUNE method leads to roughly twice as many features as DEFAULT, which is what one would expect, since half of the nodes in a balanced binary tree are leaves and the other half are interior nodes. NONZERO typically yields the shortest and the fewest rules, as expected.

We then compared DTSL to three standard Markov network structure learners: L1 regularized logistic regression [1], BLM [6], and DP [2]. We also include Atomic, a model that assumes all variables are independent, as a simple baseline. Accuracy and timing results are in Table VI. For the comparison, we selected the DTSL method that performed best on the validation set. In some cases, such as KDDCup 2000, this was not the method that performed best on the test data. Figure 2 contains scatterplots comparing DTSL to each baseline method.

DTSL achieves the best overall performance on five of the 13 datasets. We compare the performance of DTSL to the other algorithms using the Wilcoxon signed-ranks tests, where the test set CMLL of each dataset appears as one sample in the significance test. DTSL outperforms L1 on six of the 13 domains which results in no significant difference according to a Wilcoxon signed-ranks test. Even for the datasets where DTSL performs worse than L1, it offers comparable accuracy, as shown by the scatterplots. DTSL’s accuracy is substantially better than L1’s on the Plants and MSNBC domains. The average feature length for DTSL is 7.61 and 10.42 for Plants and MSNBC, respectively, which supports the hypothesis that inducing longer features can improve the performance of a model.

DTSL achieves a better CMLL score than BLM on 10 of the 13 domains. DTSL significantly outperforms BLM at the 0.006 significance level according to a Wilcoxon signed-ranks test. DTSL beats Della Pietra et al.’s algorithm on 12 of the 13 domains. DTSL significantly outperforms Della Pietra et al. at the 0.0008 significance level according to a Wilcoxon signed-ranks test.

On average, DTSL is 16 times faster than L1 and 870 times faster than BLM. DTSL has a faster run time than L1, BLM and Della Pietra et al.’s algorithm on all 13 domains. DTSL is significantly faster than each of the other algorithms at less than 0.001 significance level according to a Wilcoxon signed-ranks test.

For all algorithms, timing results are for structure selection only, excluding tuning and weight learning, which were not heavily optimized. For BLM and Della Pietra et al., structure learning is the bottleneck, always taking significantly longer than weight learning. Due to L1’s greater speed, weight learning was slower than L1 structure learning on three datasets. Since DTSL is even faster, weight learning was slower than DTSL structure learning on every dataset. Furthermore, because DTSL often selected structures with longer or more features than L1, weight learning was often slower on DTSL’s models than L1’s, in spite of the fact that the same weight learning methods were employed for all models. Since DTSL effectively solves the problem of slow structure learning, efficient weight learning becomes increasingly important. A stochastic optimization algorithm, such as SGD-QN [20], might yield significantly better performance.

Since DTSL is always faster than the baseline algorithms and often more accurate, it is a good choice to consider when performing Markov network structure learning. Whether or not DTSL is most accurate depends on the dataset. Some domains are best captured by a large number of pairwise interactions; in such cases, L1 performs best. Others depend on higher order interactions that are easily discovered by DTSL.

DTSL has two weaknesses. The first is a higher risk of overfitting, since it often generates many very specialized features. For the most part, this can be remedied with careful tuning on a validation set. The second is a limited ability to capture many independent interactions. For instance, to capture pairwise interactions between a variable and k other variables would require a decision tree with 2^k leaves, even though such interactions could be represented exactly by $O(k)$ features. For future work, we would like to learn sets of decision trees or other structures that can capture more independent interactions without making the overly restrictive pairwise assumption of L1.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented DTSL, a new approach to learning Markov networks using decision trees. DTSL is

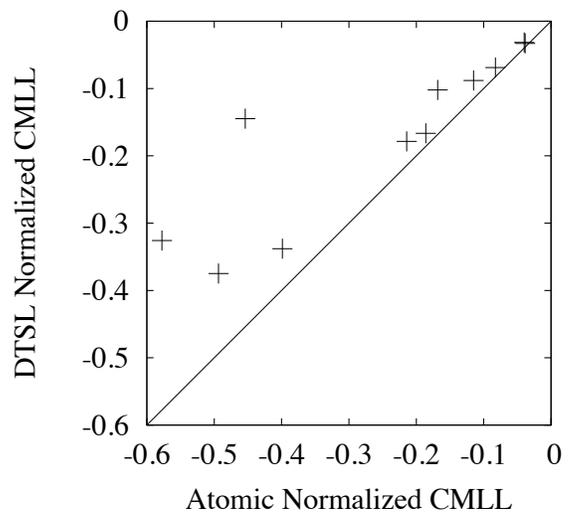
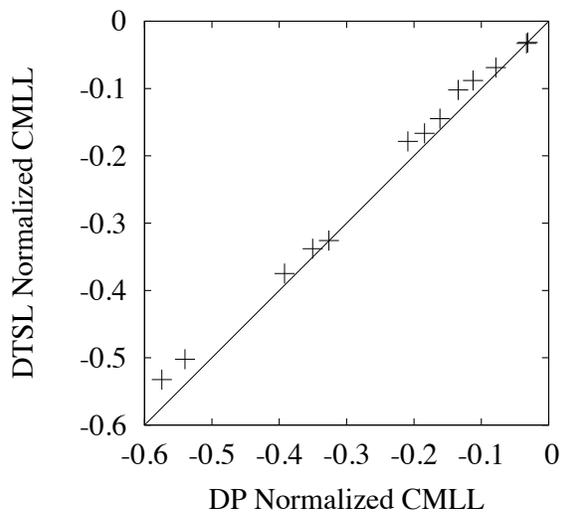
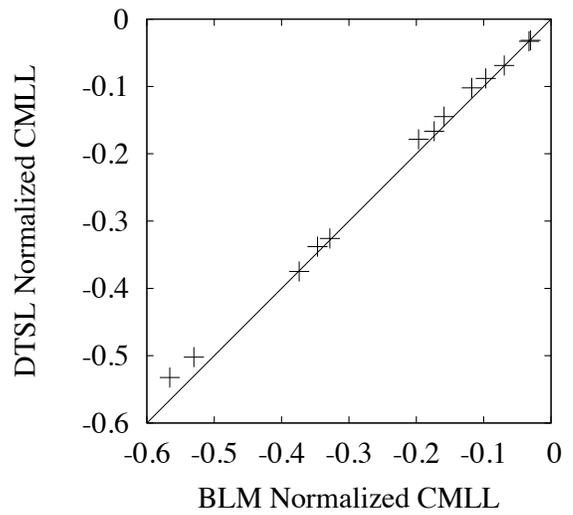
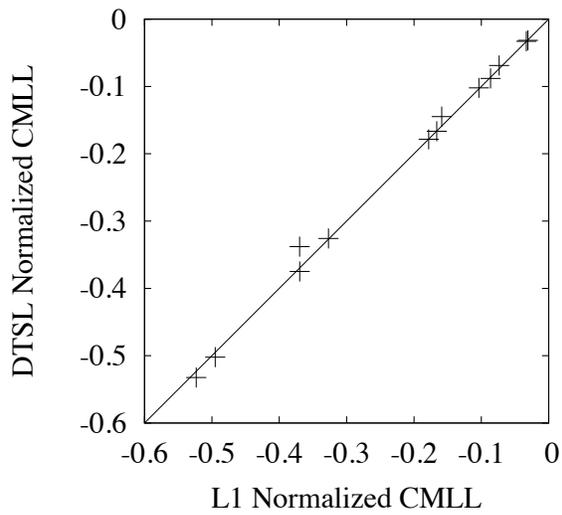


Figure 2. Normalized CMLL of DTSL vs. the normalized CMLL of each baseline method. CMLLs were normalized by dividing by the number of variables. Points above the line are where DTSL outperforms the baseline.

Table IV
TEST SET CMLL FOR DIFFERENT FEATURE GENERATION METHODS.

Data Set	Default	Prune	Prune-10	Prune-5	Nonzero
NLTCS	-5.313	-5.210	-5.213	-5.205	-5.224
MSNBC	-5.724	-5.745	-5.888	-6.111	-5.870
KDDCup 2000	-2.696	-2.155	-2.104	-2.107	-2.085
Plants	-10.814	-9.988	-10.074	-10.636	-11.054
Audio	-38.093	-37.893	-37.900	-38.405	-37.484
Jester	-51.021	-50.818	-50.818	-51.155	-50.212
Netflix	-54.389	-54.177	-54.179	-54.561	-53.234
MSWeb	-29.757	-28.648	-21.891	-16.589	-9.278
Book	-35.484	-34.451	-34.451	-34.718	-35.238
EachMovie	-54.400	-51.043	-51.088	-52.464	-52.197
WebKB	-158.790	-151.195	-151.104	-151.577	-150.529
20 Newsgroups	-195.607	-199.516	-201.060	-169.877	-154.825
Reuters-52	-128.009	-107.613	-106.547	-100.283	-82.929

Table V
FEATURE CHARACTERISTICS FOR DIFFERENT FEATURE GENERATION METHODS.

Data Set	Average Feature Length					Number of Features in the Learned Model				
	DEFAULT	PRUNE	PRUNE-10	PRUNE-5	NONZERO	DEFAULT	PRUNE	PRUNE-10	PRUNE-5	NONZERO
NLTCS	7.20	6.32	6.24	4.17	3.96	1,529	2,958	2,910	899	980
MSNBC	11.39	10.42	8.17	4.17	4.50	12,356	24,530	12,530	1,015	4,159
KDDCup 2000	8.44	7.61	6.11	3.94	2.87	4,403	8,585	6,941	2,831	2,274
Plants	7.58	6.69	6.17	4.15	3.61	6,264	12,289	11,439	3,865	4,303
Audio	6.80	5.93	5.78	4.17	3.09	7,097	13,866	13,510	5,851	4,946
Jester	6.20	5.35	5.34	4.18	3.70	5,834	11,308	11,292	5,832	4,796
Netflix	6.67	5.79	5.79	4.18	3.75	7,897	15,437	15,433	5,897	6,659
MSWeb	20.12	20.03	5.04	3.41	2.49	7,744	14,788	9,392	5,665	3,879
Book	4.22	3.59	3.59	3.22	2.06	6,466	11,720	11,720	10,408	3,454
EachMovie	5.15	4.42	4.40	3.60	2.48	10,343	19,568	19,504	14,704	6,999
WebKB	4.09	3.48	3.47	3.14	2.15	10,004	17,971	17,939	16,247	5,858
20 Newsgroups	5.78	4.99	4.89	3.79	2.69	28,622	55,005	54,241	35,533	21,008
Reuters-52	5.09	4.40	4.29	3.44	2.44	16,390	30,684	30,232	23,042	10,485

Table VI
TEST SET CMLL AND RUNNING TIME FOR DTSL AND BASELINES. DTSL FEATURE GENERATION METHOD SELECTED USING THE VALIDATION SET.

Data Set	CMLL					Run Time (Minutes)			
	DTSL	L1	BLM	DP	Atomic	DTSL	L1	BLM	DP
NLTCS	-5.213	-5.231	-5.253	-5.220	-9.241	< 0.1	0.1	24.5	15.8
MSNBC	-5.745	-6.286	-5.892	-5.957	-6.780	0.4	1.1	203.5	1440.0
KDDCup 2000	-2.107	-2.124	-2.099	-2.112	-2.456	2.0	9.8	62.9	1440.0
Plants	-9.988	-10.962	-10.960	-11.143	-31.321	0.2	3.2	514.6	1440.0
Audio	-37.484	-36.972	-37.385	-39.224	-49.362	0.3	2.7	434.3	1398.5
Jester	-50.212	-49.508	-53.025	-53.999	-63.891	0.2	4.4	350.2	1440.0
Netflix	-53.234	-52.329	-56.598	-57.429	-64.578	0.3	6.1	1367.8	1440.0
MSWeb	-9.278	-9.075	-8.936	-9.187	-11.720	2.2	43.9	64.8	1440.0
Book	-34.451	-36.814	-34.768	-39.254	-41.308	1.9	24.2	47.3	1440.0
EachMovie	-51.043	-51.861	-58.852	-67.175	-84.102	1.1	39.5	41.3	1440.0
WebKB	-150.529	-150.249	-165.736	-176.208	-180.640	1.6	45.0	49.9	1440.0
20 Newsgroups	-154.825	-154.540	-161.269	-171.380	-172.908	11.0	347.7	468.9	1440.0
Reuters-52	-82.929	-81.285	-91.196	-105.664	-108.262	4.5	105.5	170.9	1440.0

similar to the approach of Ravikumar et al. [1], except that we use decision trees in place of L1 logistic regression. This allows us to learn longer features capturing interactions among more variables, which yields substantially better performance in several domains. DTSL is also similar to methods for learning dependency networks with tree conditional probability distributions [7]. However, dependency networks may not represent consistent probability distributions and require that inference be done with Gibbs sampling, while

the Markov networks learned by DTSL have neither of those limitations.

In terms of speed, we found DTSL to be an order of magnitude faster than L1 logistic regression, and 3-4 orders of magnitude faster than the global structure learning approaches of BLM [6] and Della Pietra et al. [2]. In terms of accuracy, DTSL is comparable in accuracy to other approaches, placing first on 5 out of 13 datasets.

Future work includes exploring other methods of learning

Table VII
 FEATURE CHARACTERISTICS FOR DTSL AND BASELINES. DTSL FEATURE GENERATION METHOD SELECTED USING THE VALIDATION SET.

Data Set	Average Feature Length				Number of Features in the Learned Model			
	DTSL	L1	BLM	DP	DTSL	L1	BLM	DP
NLTCS	6.24	2.00	4.63	3.12	2,910	155	458	137
MSNBC	10.42	2.00	4.47	4.20	24,530	167	3,168	845
KDDCup 2000	3.94	2.00	2.46	4.12	2,831	2,144	2,421	584
Plants	7.61	2.00	3.18	2.76	12,289	2,255	1,148	325
Audio	3.09	2.00	2.71	2.03	4,946	5,101	2,755	272
Jester	3.70	2.00	6.27	3.13	4,796	5,121	479	191
Netflix	3.75	2.00	6.29	2.55	6,659	5,148	1,040	186
MSWeb	2.49	2.00	2.81	2.26	3,879	13,663	4,504	397
Book	3.59	2.00	2.07	2.14	11,720	12,036	5,577	120
EachMovie	4.42	2.00	3.61	2.05	19,568	11,155	1,615	158
WebKB	2.15	2.00	11.86	2.11	5,858	7,671	2,152	37
20 Newsgroups	2.69	2.00	10.95	2.10	21,008	29,349	3,376	20
Reuters-52	2.44	2.00	10.52	2.11	10,485	37,413	3,481	27

local structure, such as rule sets, boosted decision trees, and neural networks; determining sufficient conditions for the asymptotic consistency of local learning; further improving speed, perhaps by using frequent itemsets; and incorporating faster methods for weight learning, since structure learning is no longer the bottleneck.

ACKNOWLEDGMENTS

This research was partly funded by ARO grant W911NF-08-1-0242. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO or the United States Government.

REFERENCES

- [1] P. Ravikumar, M. J. Wainwright, and J. Lafferty, "High-dimensional ising model selection using L1-regularized logistic regression," *Annals of Statistics*, 2009.
- [2] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 380–392, 1997.
- [3] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, Eds., *Markov Chain Monte Carlo in Practice*. London, UK: Chapman and Hall, 1996.
- [4] A. Kulesza and F. Pereira, "Structured learning with approximate inference," in *NIPS 20*, 2007, pp. 785–792.
- [5] J. Besag, "Statistical analysis of non-lattice data," *The Statistician*, vol. 24, pp. 179–195, 1975.
- [6] J. Davis and P. Domingos, "Bottom-up learning of Markov network structure," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning*. Haifa, Israel: ACM Press, 2010.
- [7] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *Journal of Machine Learning Research*, vol. 1, pp. 49–75, 2000.
- [8] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proc. UAI'99*, 1999.
- [9] D. Chickering, D. Heckerman, and C. Meek, "A Bayesian approach to learning Bayesian networks with local structure," in *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. Providence, RI: Morgan Kaufmann, 1997, pp. 80–89.
- [10] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston, MA: ACM Press, 2000, pp. 71–80.
- [11] G. Andrew and J. Gao, "Scalable training of l1-regularized log-linear models," in *Proc. ICML'07*, 2007, pp. 33–40.
- [12] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 3, pp. 503–528, 1989.
- [13] S.-I. Lee, V. Ganapathi, and D. Koller, "Efficient structure learning of Markov networks using L1-regularization," in *NIPS 19*, 2007, pp. 817–824.
- [14] H. Poon and P. Domingos, "Sound and efficient inference with probabilistic and deterministic dependencies," in *Proc. AAAI'06*, 2006, pp. 458–463.
- [15] C. Blake and C. J. Merz, "UCI repository of machine learning databases," Dept. ICS, UC Irvine, CA, Tech. Rep., 2000, <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- [16] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng, "KDD-Cup 2000 organizers' report: Peeling the onion," *SIGKDD Explorations*, vol. 2, no. 2, pp. 86–98, 2000.

- [17] G. Hulten and P. Domingos, "Mining complex models from arbitrarily large databases in constant time," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Canada: ACM Press, 2002, pp. 525–531.
- [18] C. Ziegler, S. McNee, J. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proc. WWW'05*, 2005, pp. 22–32.
- [19] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.
- [20] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *Journal of Machine Learning Research*, vol. 10, pp. 1737–1754, 2009.