Markov Network Structure Learning: A Randomized Feature Generation Approach

Jan Van Haaren and **Jesse Davis**
Department of Computer Science, KU Leuven
Celestijnenlaan 200A - box 2402, 3001 Heverlee, Belgium
{jan.vanhaaren, jesse.davis}@cs.kuleuven.be

Abstract

The structure of a Markov network is typically learned in one of two ways. The first approach is to treat this task as a global search problem. However, these algorithms are slow as they require running the expensive operation of weight (i.e., parameter) learning many times. The second approach involves learning a set of local models and then combining them into a global model. However, it can be computationally expensive to learn the local models for datasets that contain a large number of variables and/or examples. This paper pursues a third approach that views Markov network structure learning as a feature generation problem. The algorithm combines a data-driven, specific-to-general search strategy with randomization to quickly generate a large set of candidate features that all have support in the data. It uses weight learning, with L1 regularization, to select a subset of generated features to include in the model. On a large empirical study, we find that our algorithm is equivalently accurate to other state-of-the-art methods while exhibiting a much faster run time.

Introduction

A Markov network is an undirected graphical model for compactly representing a joint probability distribution over a set of random variables. The goal of structure learning is to discover conditional (in)dependencies in the data such that the joint distribution can be represented more compactly. Markov networks are often represented as a log-linear model, which means that structure learning can be posed as a feature induction problem.

Typically, the structure learning problem is addressed through standard search based techniques. Algorithms that follow this strategy use the current feature set to construct a set of candidate features. After evaluating each feature, the highest scoring feature is added to the model. The search can follow a top-down (i.e., general-to-specific) strategy (e.g., McCallum (2003); Della Pietra, Della Pietra, and Lafferty (1997)) or bottom-up (i.e., specific-to-general) strategy (e.g., Davis and Domingos (2010); Mihalkova and Mooney (2007)). Search based approaches tend to be slow due the large number of candidate structures. Furthermore, scoring each candidate structure requires learning the weights of each feature. Weight learning requires iterative optimization, where each iteration requires running inference over the model. Unfortunately, inference is often intractable.

An alternative approach that has gained popularity in recent years involves learning a set of local models and then combining them into a global model. Algorithms that follow this strategy consider each variable in turn and build a model to predict this variable's value given the remaining variables. Each predictive model is then transformed into a set of features, each of which is included in the final, global model. Two successful approaches that use this strategy are Ravikumar et al.'s (2010) algorithm, which employs L1 logistic regression as the local model and DTSL (Lowd and Davis 2010), which uses a probabilistic decision tree learner as the local model. Still, it can be computationally expensive to learn the local models if the dataset contains a large number of variables and/or examples.

This paper presents GSSL, a two-step approach to Markov network structure learning. The first step involves quickly generating a large set of candidate features by combining aspects from randomization and specific-to-general search. GSSL constructs an initial feature set by converting each training example into a feature. It then repeatedly picks a feature at random, generalizes it by dropping an arbitrary number of variables, and adds the generalized feature to the feature set. Note that the same feature can be generated multiple times. The second step selects a subset of features to include in the final model. GSSL prunes all features that were generated fewer times than a pre-defined threshold and then performs weight learning with L1 regularization to enforce sparsity in the final model. This approach is similar to that of Huynh and Mooney's (2008) for discriminative structure learning algorithm for Markov logic networks (which are templates for constructing Markov networks). Their approach generates features in the form of first-order definite clauses and then uses weight learning with L1 regularization to select a subset of the features. The algorithm restricts the structure of the features and in essence it learns the features for a logistic regression model.

GSSL combines some of the benefits of recent approaches to structure learning. In the feature generation phase, the algorithm proceeds in a data-driven, bottom-up fashion to explore the space of candidate features. As a result, GSSL only constructs features that have support in the data. In the feature selection phase, the algorithm performs weight learning only once to select the best features. Here, it follows the philosophy of local model based approaches that try to minimize the computational expense of weight learning. A large scale empirical evaluation on 20 real-world datasets demonstrates the advantages of GSSL. Despite its simplicity, its run times are on average twice as fast as DTSL and 15 times faster than Ravikumar et al.'s L1 approach. Furthermore, it learns more accurate models than its competitors.

Markov Networks

This section reviews the basics about representation, inference and learning for Markov networks.[1]

Representation

A *Markov network* is a model for compactly representing the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n)$ (Della Pietra, Della Pietra, and Lafferty 1997). 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:

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

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique), and Z is a normalization constant. 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:

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

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 for that example.

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) (Gilks, Richardson, and Spiegelhalter 1996), 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). Another popular method for approximate inference is loopy belief propagation (Murphy, Weiss, and Jordan 1999).

[1]Markov networks are also called Markov random fields.

Weight Learning

Weight learning uses data to automatically learn the weight associated with each feature by optimizing a given objective function. Ideally, each candidate model would be scored by its training set log-likelihood. For Markov networks, the log-likelihood is a convex function of the weights and learning can be solved via convex optimization. However, this typically requires an iterative optimization technique where each step of the optimization must calculate both the log-likelihood and its gradient. This is often computationally infeasible as it requires computing the partition function Z (see equation 2). Additionally, Kulesza and Pereira (2008) have found that employing approximate inference can mislead weight learning algorithms.

Optimizing the pseudo-likelihood (Besag 1975) is a more efficient alternative that has been widely used in domains such as spatial statistics, social network modeling and language processing. The pseudo-likelihood is defined as:

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

where V is the number of variables, N is the number of examples, $x_{i,j}$ is the value of the jth variable of the ith example, $MB_x(X_{i,j})$ is the state of $X_{i,j}$'s Markov blanket in the data. This is much more efficient to compute and can also be optimized via convex optimization.

Structure Learning

Most Markov network structure learning approaches pose the task as a feature induction problem.

Search Based Structure Learning. Della Pietra et al.'s algorithm (1997) is the standard approach to Markov network structure learning and it uses a greedy, general-to-specific (i.e., top-down) search. 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 evaluates each candidate feature f by estimating how much including f in the model would improve the model's 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.

BLM (Davis and Domingos 2010) is a more recent algorithm that employs a greedy, specific-to-general (i.e., bottom-up) search. 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.

Search based approaches suffer the drawback that they must perform weight learning to score each candidate feature. This is computationally expensive even when optimizing the pseudo-likelihood.

Local Model Based Structure Learning. More recently, researchers have explored ways to learn a set of local models and combine them into a global model. At a high level, these algorithms try to discover the Markov blanket of each variable X_i by building a model to predict the value of X_i given the remaining variables. Finally, all features are added to the model and their weights are learned globally using any standard weight learning algorithm. Ravikumar et al.'s (2010) algorithm employs L1 logistic regression models as local model. In the limit of infinite data, consistency is guaranteed (i.e., X_i is in $X_j's$ Markov blanket iff 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. A weakness of this algorithm is that it only constructs pairwise features. DTSL (Lowd and Davis 2010) employs this general strategy using a probabilistic decision tree learner as local model. Each tree is converted to a set of conjunctive features. The most straightforward conversion constructs one feature for each root-to-leaf path through the tree (the paper proposes several other conversion methods).

Still, it can be computationally expensive to learn the local models if the dataset contains a large number of variables and/or examples.

Algorithm

We now describe GSSL (<u>G</u>enerate <u>S</u>elect <u>S</u>tucture <u>L</u>earning), an algorithm for Markov network structure learning. GSSL has two main steps: (1) feature generation, and (2) feature selection. In the feature generation step, starting from an initial feature set, GSSL quickly generates a large set of candidate features by combining aspects from randomization and specific-to-general search. In the feature selection step, GSSL attempts to discard irrelevant features through a preprocessing step and then by applying weight learning with a L1 penalty.

The four key elements of GSSL, introduced in the next subsections, are: (i) how to construct the initial feature set, (ii) how to generate new features, (iii) how to perform feature selection and (iv) how the overall algorithm functions.

Initial Feature Set

The algorithm requires an initial feature set. Since the generation process generalizes features, the initial features should be specific so that it is possible to generalize them. The training examples can provide very specific features and GSSL considers two ways of converting them into the initial feature set. The first approach creates one long feature for each unique training example by forming a conjunction over all variables in that example. Because the features are maximally specific, generalization can generate every feature that has support in the data. The second approach works for problems that have only binary variables and it builds "positive" features by forming a conjunction only over those variables that have a value of true. Because many domains are sparse, this conversion has the advantage of being more compact. Both approaches have the advantage that every initial feature has support (i.e., occur) in the data. Consequently, generalizing any of these features yields a feature that is guaranteed to match at least one training example.

The following example dataset, where each row is a training example, illustrates the conversion process.

#	V_0	V_1	V_2	V_3	V_4
1.	true	false	false	true	true
2.	true	false	true	false	true
3.	false	true	true	true	true
4.	true	false	true	false	true

The first approach yields the following initial feature set:

1a $V_0 = 1 \wedge V_1 = 0 \wedge V_2 = 0 \wedge V_3 = 1 \wedge V_4 = 1$

2a $V_0 = 1 \wedge V_1 = 0 \wedge V_2 = 1 \wedge V_3 = 0 \wedge V_4 = 1$

3a $V_0 = 0 \wedge V_1 = 1 \wedge V_2 = 1 \wedge V_3 = 1 \wedge V_4 = 1$

The second approach yields the following initial feature set:

1b $V_0 = 1 \wedge V_3 = 1 \wedge V_4 = 1$

2b $V_0 = 1 \wedge V_2 = 1 \wedge V_4 = 1$

3b $V_1 = 1 \wedge V_2 = 1 \wedge V_3 = 1 \wedge V_4 = 1$

In this example, the initial feature set is smaller than the example dataset since GSSL removes duplicate training examples as a preprocessing step. Note that the second feature matches both the second and the fourth training example.

Feature Generation

The key step involves generating the features. To create a new feature, GSSL uniformly picks a feature, f, at random from the feature set. It generalizes f by dropping n arbitrary variables, where n is drawn from the uniform distribution between 1 and $l - 2$, with l the length of f. The one ensures that the new feature is actually a generalization. The $l - 2$ ensures the length of the generalized feature is at least length two since atomic features (i.e., features of length one) cannot be further generalized. To decide which variables to drop from the feature, GSSL shuffles the order of variables in the ungeneralized feature and drops the first n to create the new feature $f^{'}$. Note that $f^{'}$ is added back into the feature set so that it can be selected for generalization in the future. This does not change which features can be generated, but it biases the generation towards shorter features. This process is repeated for a fixed number of times. Note, that the same generalization can be produced multiple times. As a final step, GSSL adds an atomic feature for each variable, which captures its marginal probability, to its feature set. This is known to help performance in practice.

To illustrate the process, suppose that GSSL picks feature **1a** from the above dataset. Since this feature is of length five, it uniformly draws an arbitrary number between one and three. Let us assume that this number is two. The algorithm then continues with dropping two arbitrary variables, say V_2 and V_3, such that the resulting feature is $V_0 = 1 \wedge$

$V_1 = 0 \wedge V_4 = 1$. The new feature is more general and now matches the second and fourth training examples in addition to the first training example.

Feature Selection

GSSL does feature selection in two steps. In the first step, GSSL tries to identify and discard unnecessary features. One idea would be to perform pruning based on the support of each feature (i.e., how many examples it matches) in the data. However, GSSL does not count the number of training examples that each feature matches as this is a computationally expensive endeavor. As a result, GSSL requires another mechanism to identify features that potentially have high support in the data. GSSL removes any feature that was generated fewer times than a given threshold value. The use of a threshold is based on the assumption that GSSL is likely to generate features with high support in the data more often. In a second step, the algorithm performs L1 weight learning on the remaining features to produce the final model. Placing a L1 penalty on the magnitude of the weight forces many of the weights to be zero, which has the effect of removing them from the model. Thus the weight learning helps select the most relevant features.

Algorithm Overview

The overall control structure of GSSL proceeds in two phases. The first step, outlined in Algorithm 1, generates a large number of features. As input, this subroutine receives a set of training examples, TS, and the desired number of non-unique features to be generated, max. The TS is converted into the initial feature set FS. Then, a feature f is randomly selected, generalized to $f^{'}$, and $f^{'}$ is added to FS. The procedure iterates until it has generated max number of (non-unique) features. Finally, it adds an atomic feature for each variable to FS.

Algorithm 1 FEATURE GENERATION (training set TS, number of non-unique features max)

```
1: Feature set $FS \leftarrow$ Convert $TS$ into features
2: while $|FS| < max$ do
3:     Uniformly draw a feature $f$ from $FS$
4:     Let $l$ be the length of $f$
5:     Uniformly draw a number $n \sim [1, l-2]$
6:     Drop $n$ arbitrary variables from feature $f$
7:     $FS \leftarrow FS \cup \{f\}$
8: end while
9: Add unit clause for each variable to $FS$
```

The second step is outlined in Algorithm 2. As input, this subroutine receives the generated features, FS, and a lower bound, $thres$, on the number of times each feature was proposed during feature generation. First, the algorithm loops through the feature set and discards all features that were generated fewer than $thres$ times. Second, the algorithm learns the weights for each feature through L1 optimization, which reduces the number of features in the model by forcing many weights to be zero.

Algorithm 2 FEATURE SELECTION (feature set FS, lower bound on the desired number of occurrences $thres$)

```
1: for all features $f$ in $FS$ do
2:     if $f$ occurs $\leq thresh$ times then
3:         $FS \leftarrow FS \setminus \{f\}$
4:     end if
5: end for
6: Perform L1 weight learning on $FS$
```

Dataset	Size of train set	Size of tune set	Size of test set	Numb. of var.	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	0.199
Jester	9,000	1,000	4,116	100	0.608
Netflix	15,000	2,000	3,000	100	0.541
Accidents	12,758	1,700	2,551	111	0.291
Retail	22,041	2,938	4,408	135	0.024
Pumsb Star	12,262	1,635	2,452	163	0.270
DNA	1,600	400	1,186	180	0.253
Kosarak	33,375	4,450	6,675	190	0.020
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	0.059
WebKB	2,803	558	838	839	0.064
Reuters-52	6,532	1,028	1,540	889	0.036
20 Newsgroups	11,293	3,764	3,764	910	0.049
BBC	1,670	225	330	1,058	0.078
Ad	2,461	327	491	1,556	0.008

Table 1: Dataset characteristics

Experimental Results

In this section, we evaluate GSSL on 20 real-world datasets. The evaluation consists of two parts. In the first part, we compare GSSL to three state-of-the art Markov network structure learning algorithms in terms of accuracy and run time: DTSL (Lowd and Davis 2010), Ravikumar et al.'s algorithm (2010), referred to as L1, and BLM (Davis and Domingos 2010). In the second part, we investigate how GSSL's parameters influence its performance.

Datasets

Table 1 describes the characteristics of each dataset. Note that each dataset only contains binary variables. The datasets are shown in ascending order by number of variables. We used the 13 datasets from Lowd and Davis (2010). Additionally, we used seven new datasets: Accidents,[2] Ad,[3] BBC,[4] DNA,[5] Kosarak,[2] Pumsb Star[2] and Retail.[2] For Ad and DNA, we used all the provided binary features. For BBC, we created one binary feature for each word in the training set. The remaining four datasets are for frequent itemset

[2]http://fimi.ua.ac.be/data/

[3]http://archive.ics.uci.edu/ml/datasets.html

[4]http://mlg.ucd.ie/datasets/bbc.html

[5]http://www.cs.sfu.ca/ wangk/ucidata/dataset/DNA/

mining. Here, we subsampled the data and divided our subsample into a training, a tuning and a test set. We counted the number of occurrences of each item in the training set. We constructed one binary feature for each item that met a particular threshold (500 for Accidents and Pumsb Star and 50 for Kosarak and Retail) on the training set.

Methodology

We used the training data to learn the structure and weights for all four methods. The code for GSSL is publicly available.[6] To learn the models we used the publicly available code for DTSL and BLM. For L1, we used the OWL-QN software package (Andrew and Gao 2007) for performing L1 logistic regression. For GSSL, we generated half a million, one million, two million and five million features and used pruning thresholds of one, two and five. We tried both methods for converting the training set to the initial feature set. For the baseline algorithms, we used the parameter settings described in Lowd and Davis (2010).

GSSL, DTSL and L1 all produce feature sets and it is necessary to learn the weights for each feature. For weight learning, we used the Libra Toolkit[7] to optimize the train set pseudo-likelihood, which was done for computational tractability. In order to allow for a fair comparison, we performed the exact same weight learning procedure and employed the same set of L1 regularization parameters for all three algorithms. For each dataset, we used Gaussian priors with standard deviations 0.1, 0.5 and 1, combined with L1 norm weights of 1, 5 and 10, resulting in 9 different setups. For each algorithm, we selected the model that maximized the pseudo-log-likelihood on the validation set.

We evaluated the best model using test set conditional marginal log-likelihood (CMLL) (Lee, Ganapathi, and Koller 2007; Lowd and Davis 2010). First, we divided the variables into a query set Q and an evidence set E. Then, we computed $CMLL(X = x) = \sum_{i \in Q} logP(X_i = x_i|E)$ for each example in the test set. We divided the variables into four disjoint groups for each dataset. One set served as query variables while the remaining three sets served as evidence. We repeated this procedure such that each set served as the query variables once. We computed the conditional marginal probabilities using the Gibbs sampler that is part of the Libra Toolkit. We used a burn-in of 100 samples and then computed the probability using the next 1,000 samples.

Results

Table 2 reports the CMLLs, averaged over all test examples, for each of the algorithms on all 20 datasets. We compared GSSL with each of the baselines using a Wilcoxon signed-rank test, which is a non-parametric, paired difference test. The comparison between any two algorithms involves 20 paired samples, where each sample corresponds to the test set CMLL scores on a different dataset. GSSL achieves the best overall CMLL score on 8 of the 20 datasets. According to the Wilcoxon signed-rank test, GSSL is equivalently accurate to L1, where it achieves a better CMLL score on 11 of the 20 datasets. GSSL significantly outperforms DTSL at the 0.0193 significance level according to a Wilcoxon signed-rank test, producing a better CMLL score on 15 of the 20 datasets. GSSL significantly outperforms BLM at the 0.0002 significance level according to a Wilcoxon signed-rank test, beating BLM on 19 of the 20 datasets. Despite its simplicity, GSSL is often (significantly) more accurate than its competitors.

Dataset	GSSL	L1	DTSL	BLM
NLTCS	**-5.175**	-5.232	-5.209	-5.248
MSNBC	-5.947	-6.281	**-5.727**	-5.815
KDDCup 2000	-2.071	-2.108	**-2.046**	-2.077
Plants	**-9.854**	-10.739	-10.709	-10.445
Audio	**-36.803**	-36.878	-37.484	-37.452
Jester	**-49.464**	-49.476	-50.252	-52.762
Netflix	**-52.339**	-52.401	-53.342	-56.521
Accidents	-18.180	**-16.543**	-16.957	-37.558
Retail	-10.547	**-10.534**	-10.578	-10.620
Pumsb Star	-17.245	**-13.905**	-19.508	-133.155
DNA	-81.034	**-69.035**	-69.197	-99.560
Kosarak	-10.137	-10.183	**-10.068**	-10.217
MSWeb	**-8.819**	-8.959	-16.201	-8.848
Book	-34.048	**-34.025**	-34.120	-34.650
EachMovie	**-49.873**	-50.002	-51.448	-58.582
WebKB	-144.206	**-143.290**	-148.192	-164.844
Reuters-52	-79.501	**-78.743**	-81.267	-90.852
20 Newsgroups	-148.565	**-147.007**	-151.723	-160.841
BBC	-242.424	**-239.642**	-250.302	-265.486
Ad	**-14.848**	-15.393	-16.751	-45.638

Table 2: Test set CMLL scores averaged over all test examples. The best score for each dataset is shown in bold.

GSSL exhibits outstanding run time performance, which is reported in Table 3. GSSL is the fastest algorithm on 13 of the 20 datasets, being slower to L1 and/or DTSL only when datasets have very few variables. GSSL is faster than L1 on 16 of the 20 datasets. On average, it exhibits a run time that is 15 times faster than L1. GSSL is faster than DTSL in addition to being significantly more accurate than it. GSSL's run time is lower than DTSL's on 13 of the 20 datasets and is twice as fast on average. Naturally, GSSL is significantly faster than BLM, showing an average speed-up of 4634, as it avoids the computational cost associated with running weight learning to evaluate each candidate feature.

Table 4 shows statistics about the best learned model on each dataset for each algorithm. The models that GSSL and Ravikumar et al.'s algorithm learn have more features. Ravikumar et al.'s algorithm has the lowest average feature length because it is restricted to atomic and pairwise features. On average, GSSL results in shorter features than either DTSL or BLM.

Apart from weight learning, GSSL relies on only two parameters: a desired number of features and a pruning threshold. Table 5 shows the parameters that yielded the best model for GSSL as well as the effect of thresholding on the number of features. The number of features needed to get the best performance depends on the characteristics of the dataset. Generally, if a dataset has more variables or a higher density (i.e., a greater proportion of the variables that are true), it is necessary to generate more features. Intuitively,

[6]http://dtai.cs.kuleuven.be/ml/systems/gssl

[7]http://libra.cs.uoregon.edu

	GSSL			L1			DTSL			BLM
Dataset	FG	WL	Total	FG	WL	Total	FG	WL	Total	Total
NLTCS	5	69	74	5	4	9	2	5	**7**	8,738
MSNBC	5	218	223	72	38	**110**	126	80	206	458,691
KDDCup 2000	7	114	**121**	624	21	645	780	75	855	2,814,585
Plants	8	375	383	194	38	232	34	84	**118**	128,825
Audio	9	186	**195**	166	126	292	49	153	202	184,858
Jester	10	192	202	321	193	514	27	108	**135**	107,156
Netflix	10	333	**343**	385	322	707	47	336	383	188,455
Accidents	9	1,171	1,180	833	116	949	43	242	**285**	166,767
Retail	8	165	**173**	398	100	498	112	126	238	414,684
Pumsb Star	9	891	900	939	219	1,158	43	116	**159**	164,378
DNA	9	38	47	53	25	78	6	14	**20**	20,614
Kosarak	9	259	**268**	1,113	171	1,284	238	214	452	881,393
MSWeb	8	222	**230**	2,634	186	2,820	485	303	788	1,773,963
Book	11	223	**234**	1,731	290	2,021	256	538	794	944,807
EachMovie	10	234	**244**	5,922	310	6,232	191	171	362	701,127
WebKB	11	186	**197**	7,652	718	8,370	212	167	379	775,808
Reuters-52	10	344	**354**	10,331	1,099	11,430	589	748	1,337	2,158,685
20 Newsgroups	11	663	**674**	19,615	2,375	21,990	1,455	690	2,145	5,279,550
BBC	11	136	**147**	6,028	555	6,583	173	87	260	625,467
Ad	9	131	**140**	8,711	208	8,919	740	216	956	2,677,445

Table 3: Feature generation (FG), weight learning (WL) and total run time for each algorithm. All run times are shown in seconds. The best run time for each dataset is shown in bold.

this makes sense. More variables increases the number of possible features such that we need to try more combinations to get the best feature set. A higher density suggests the presence of more regularities in the data such that the models will need more features to capture them. GSSL got the best results (based on tune set pseudo-likelihood) with half a million features three times, one million features four times, two million features seven times and five million features six times. Our experiments have shown that using a pruning threshold value of two is generally better than a threshold value of one. Furthermore, using a threshold value of five seems to be too strict and rules out too many potentially useful features.

On average, GSSL spends more than 95% of its time on weight learning, which is reported in Table 3. The time depends on both the number of generated features and the pruning threshold. Generally, generating smaller feature sets and using a higher pruning threshold yields the lowest run times for weight learning. Averaged acrossed all datasets and pruning thresholds, GSSL spends 124.87 seconds on weight learning when generating half a million features, 206.31 seconds when generating one million features, and 442.45 seconds when generating two million features. We have omitted the run time for five million features as running weight learning using a pruning threshold is often intractable. Averaged acrossed datasets and number of generated features, GSSL spends 414.83 seconds on weight learning for a pruning threshold of one, 235.22 seconds for a threshold of two, and 123.52 seconds for a threshold of five.

The other choice that GSSL has is in terms of its seed set of initial features. Using an initial feature set of only "positive features" (i.e., initial features that are only conjunctions over true variables) is better on 14 of the 20 datasets. Generally, limiting the feature set to positive features allows for much faster weight learning. Full run time and accuracy results for all of GSSL's parameter settings are available in the online appendix.[8]

Dataset	Numb. of generated features	Numb. of unique features	Numb. of features after TH	Percent. pruned by TH	Threshold value
NLTCS	500,000	143,364	12,184	91.50%	2
MSNBC	2,000,000	491,885	52,591	89.31%	1
KDDCup 2000	5,000,000	373,041	102,667	72.48%	2
Plants	1,000,000	690,390	52,709	92.37%	1
Audio	1,000,000	524,688	12,991	97.52%	2
Jester	500,000	369,000	9,679	97.38%	2
Netflix	500,000	382,760	9,393	97.55%	2
Accidents	2,000,000	1,462,685	40,669	97.22%	1
Retail	2,000,000	168,979	19,617	88.39%	2
Pumsb Star	2,000,000	1,565,510	26,419	98.31%	1
DNA	2,000,000	1,383,102	39,088	97.17%	1
Kosarak	1,000,000	288,936	97,023	66.42%	1
MSWeb	1,000,000	149,689	57,576	61.54%	1
Book	2,000,000	855,347	101,451	88.14%	1
EachMovie	5,000,000	2,200,237	113,172	94.86%	2
WebKB	5,000,000	2,738,782	179,473	93.45%	2
Reuters-52	5,000,000	2,514,634	163,971	93.48%	2
20 Newsgroups	5,000,000	3,037,720	159,013	94.77%	2
BBC	5,000,000	2,975,003	215,000	92.77%	2
Ad	2,000,000	578,322	78,797	86.37%	1

Table 5: Statistics of the best model GSSL learned for each dataset. TH stands for thresholding.

Conclusions

While striking in its simplicity, GSSL offers outstanding performance, in terms of both accuracy and run time, as demon-

[8]http://dtai.cs.kuleuven.be/ml/systems/gssl

	GSSL			L1			DTSL			BLM	
Dataset	Numb. of generated features	Numb. of features after WL	Average feature length	Numb. of generated features	Numb. of features after WL	Average feature length	Numb. of generated features	Numb. of features after WL	Average feature length	Numb. of features after WL	Average feature length
NLTCS	12,184	8,881	3.83	134	134	1.88	2,958	2,185	6.15	385	4.17
MSNBC	52,591	52,355	4.18	153	152	1.88	24,530	21,435	10.33	4,213	4.69
KDDCup 2000	102,666	21,023	3.34	2,080	1,433	1.95	8,585	6,039	7.64	4,877	3.25
Plants	52,709	52,499	3.24	2,404	2,286	1.96	12,289	6,243	6.50	2,469	5.95
Audio	12,991	11,317	2.20	5,049	4,878	1.97	4,946	4,805	3.08	1,938	2.21
Jester	9,679	9,461	1.99	5,008	4,966	1.97	4,796	4,751	3.70	992	8.27
Netflix	9,393	9,335	1.99	4,985	4,952	1.97	6,659	6,604	3.74	1,140	5.82
Accidents	40,669	39,236	3.28	5,840	5,786	1.98	10,194	5,390	6.85	1,329	8.33
Retail	19,617	8,813	2.51	9,113	3,383	1.96	4,439	3,944	5.26	2,823	2.12
Pumsb Star	26,419	24,337	3.16	6,475	6,392	1.97	4,666	4,434	5.24	5,789	28.20
DNA	39,088	25,936	2.99	4,302	4,167	1.96	2,246	2,221	3.11	1,413	10.74
Kosarak	97,023	24,193	2.85	7,771	4,725	1.95	8,724	6,402	5.37	3,860	2.95
MSWeb	57,576	24,234	2.90	33,828	11,548	1.97	14,788	11,911	18.17	5,756	3.01
Book	101,451	15,930	1.97	120,833	10,647	1.95	11,720	6,589	3.57	6,077	1.97
EachMovie	113,171	58,792	2.54	70,568	15,900	1.96	19,568	9,399	4.44	2,561	4.10
WebKB	179,472	43,787	2.01	216,123	35,901	1.97	17,939	10,633	3.43	3,029	8.61
Reuters-52	163,970	93,233	2.12	172,730	91,373	1.99	30,684	19,660	4.33	5,907	11.33
20 Newsgroups	159,012	88,005	2.01	193,177	120,881	1.99	21,008	18,915	2.69	4,256	9.69
BBC	215,000	43,566	1.97	270,623	42,297	1.97	4,417	4,131	1.91	2,299	9.56
Ad	78,797	33,222	2.31	212,663	23,786	1.93	13,708	11,335	3.34	2,370	3.87

Table 4: Statistics of each algorithm's best model for each dataset. WL stands for weight learning.

strated by a large empirical evaluation on 20 real-world datasets. It is 15 times faster than Ravikumar et al.'s (2010) L1 approach while being equivalently accurate. It is also faster and significantly more accurate than both DTSL and BLM. GSSL gains its efficiency by avoiding the computational expense associated with building local models. It accomplishes this by using the training data to guide the feature generation. Furthermore, the data-driven generation guarantees that each generated feature occurs in the data. By using a pruning strategy and L1 weight learning, GSSL selects which features to include in the final model. In the future, it may be possible to improve the run time further by exploring more sophisticated pruning strategies. Reducing the number of features considered during weight learning would greatly improve its efficiency.

Acknowledgments. We thank Daniel Lowd and Wannes Meert for their helpful comments. This research is partially funded by the Research Fund K.U.Leuven and the EU FP7 Marie Curie Career Integration Grant (#294068).

References

Andrew, G., and Gao, J. 2007. Scalable Training of L1-Regularized Log-Linear Models. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, 33–40. ACM Press.

Besag, J. 1975. Statistical Analysis of Non-Lattice Data. *The Statistician* 24:179–195.

Davis, J., and Domingos, P. 2010. Bottom-Up Learning of Markov Network Structure. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*. ACM Press.

Della Pietra, S.; Della Pietra, V.; and Lafferty, J. 1997. Inducing Features of Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19:380–392.

Gilks, W. R.; Richardson, S.; and Spiegelhalter, D. J. 1996. *Markov Chain Monte Carlo in Practice*. Chapman and Hall.

Huynh, T., and Mooney, R. 2008. Discriminative Structure and Parameter Learning for Markov Logic Networks. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, 416–423. ACM Press.

Kulesza, A., and Pereira, F. 2008. Structured Learning with Approximate Inference. In *Advances in Neural Information Processing Systems 20*. 785–792.

Lee, S.-I.; Ganapathi, V.; and Koller, D. 2007. Efficient Structure Learning of Markov Networks using L_1-Regularization. In *Advances in Neural Information Processing Systems 19*. 817–824.

Lowd, D., and Davis, J. 2010. Learning Markov Network Structure with Decision Trees. In *Proceedings of the Tenth IEEE International Conference on Data Mining*.

McCallum, A. 2003. Efficiently Inducing Features of Conditional Random Fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, 403–410.

Mihalkova, L., and Mooney, R. J. 2007. Bottom-Up Learning of Markov Logic Network Structure. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, 625–632.

Murphy, K. P.; Weiss, Y.; and Jordan, M. I. 1999. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

Ravikumar, P.; Wainwright, M. J.; and Lafferty, J. 2010. High-Dimensional Ising Model Selection using L1-Regularized Logistic Regression. *Annals of Statistics* 38(3):1287–1319.