Bottom-Up Learning of Markov Network Structure

Jesse Davis JDAVIS@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

Abstract

The structure of a Markov network is typically learned using top-down search. At each step, the search specializes a feature by conjoining it to the variable or feature that most improves the score. This is inefficient, testing many feature variations with no support in the data, and highly prone to local optima. We propose bottom-up search as an alternative, inspired by the analogous approach in the field of rule induction. Our BLM algorithm starts with each complete training example as a long feature, and repeatedly generalizes a feature to match its k nearest examples by dropping variables. An extensive empirical evaluation demonstrates that BLM is both faster and more accurate than the standard top-down approach, and also outperforms other state-of-the-art methods.

1. Introduction

Markov networks are a powerful representation for joint distributions, but learning them from data is extremely difficult. When learning structure, scoring each candidate requires first learning the optimal weights for it. Weight learning cannot be done in closed form, and requires inference as a subroutine. In turn, inference is intractable. As a result, despite its promise, Markov network structure learning has not been widely used to date.

Roughly speaking, the goal of Markov network structure learning is to discover regions of high probability in instance space, form features to represent them, and learn the corresponding weights. The standard approach to learning the structure of a Markov network is the algorithm of Della Pietra et al. (1997), which induces a set of features. It performs a top-down, or general-to-specific, search. At each step, the search specializes a feature by conjoining it to the variable or feature that most improves the score. Several other algorithms exist that perform top-down heuristic search through the space of candidate structures (McCallum, 2003; Kok & Domingos, 2005). General-to-specific search is inefficient as it tests many feature variations with no support in the data, and is highly prone to local optima.

In recent years, a number of alternative approaches have been developed. One approach is to couple parameter learning and feature induction into one step through L1 regularization, which forces most weights to be zero (Lee et al., 2007; Höfling & Tibshirani, 2009; Ravikumar et al., 2010). By providing the optimization procedure with a large initial feature set (i.e., all possible features of interest), model selection occurs by picking the features with non-zero weights after learning. However, due to tractability concerns, these approaches only construct pairwise networks (Höfling & Tibshirani, 2009; Ravikumar et al., 2010). In principle, Lee et al.'s (2007) algorithm can learn arbitrarily long features. However, in practice it has only been evaluated for inducing features of length two (i.e., learning a pairwise network). Another class of algorithms learns only models of small tree width, which ensures that inference remains tractable (Narasimhan & Bilmes, 2004; Chechetka & Guestrin, 2007). However, small tree width is a very restrictive assumption, and the applicability of this approach is limited.

The problem of feature induction for Markov networks is similar to the problem of rule induction for classification. Rule induction constructs a rule set to discriminate between different categories. Each rule consists of a body and a head. The body is a conjunction of antecedents, where each antecedent tests a single variable. A rule covers (or matches) an example if all antecedents of the rule are true of the example. An alternative paradigm, known as bottom-up or specific-to-general learning (Domingos, 1996; Muggleton & Feng,

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010. Copyright 2010 by the author(s)/owner(s).

1990), exists in rule induction, which addresses the shortcomings of top-down search. Specific-to-general induction starts with a rule body that contains many antecedents. It then generalizes the rule by removing antecedents from its body, which expands the number of examples the rule matches. To our knowledge, the only approach for learning Markov (logic) networks to date that uses bottom-up ideas is Mihalkova and Mooney's BUSL algorithm (2007). However, it only uses them in a pre-processing step to reduce the number of candidate features; the main feature induction process is still top-down.

We propose bottom-up search as an alternative method for learning the structure of a Markov network. Our algorithm, called BLM (Bottom-up Learning of Markov Networks), is inspired by the RISE algorithm for rule induction (Domingos, 1996). 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, which indicates that a local optima has been reached. We perform an extensive empirical evaluation on 21 data sets, and find that BLM is both faster and significantly more accurate than the standard top-down approach (Della Pietra et al., 1997), while also outperforming other state-of-the-art methods.

2. Markov Networks

2.1. Representation

A *Markov network* models the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n)$ (Della Pietra et al., 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

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

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 represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state:

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

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 attribute. A feature matches an example if it is true of that example.

2.2. 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 et al., 1996), and in particular Gibbs sampling, which samples each variable in turn given its *Markov blanket.*

2.3. Weight Learning

Ideally, each candidate model would be scored by its training set log-likelihood. Doing so requires computing the maximum likelihood estimate of the weight, which is a computationally challenging task as it requires performing inference over the model. The derivative of the log-likelihood with respect to the jth feature is:

$$\frac{\partial}{\partial w_j}\log P_w(X=x) = n_j(x) - \mathbb{E}_w[n_j(x)] \tag{3}$$

where $n_j(x) = \sum_i f_j(x_i)$, x_i is the ith training example and $\mathbb{E}_w[n_j(x)]$ is computed using the current weight vector. In other words, the jth component of the gradient is simply the difference between the total value of the jth feature in the data and its expectation according to the current model. Weight learning requires iterative optimization where each step performs inference over the current model to compute the expectations. Furthermore, efficient optimization methods also require computing the log-likelihood itself, and thus the partition function Z. Additionally, Kulesza and Pereira (2007) have found that approximate inference can mislead weight learning algorithms.

A more efficient alternative, widely used in areas like spatial statistics and social network modeling, is to optimize the pseudo-likelihood (Besag, 1975):

$$\log P^\bullet_w(X=x) = \sum_{j=1}^{V}\sum_{i=1}^{N}\log P_w(X_{i,j}=x_{i,j}|MB_x(X_{i,j})) \tag{4}$$

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.

2.4. Structure Learning

Della Pietra et al.'s algorithm (1997) 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 in two ways. First, it considers conjoining each feature currently in the model with every other feature in the model. Second, it composes each feature in the model with each atomic feature. 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 estimates the benefit of including each candidate feature f in the model based on the KL-divergence which captures the improvement in the log-likelihood that would result by including the candidate feature. 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.

A recent L1 regularization based approach to structure learning is the method of Ravikumar et al. (2010). It learns the structure by trying to discover the Markov blanket of each attribute (i.e., its neighbors in the network). It considers each attribute X_i in turn and builds an L1 logistic regression model to predict the value of X_i given the remaining attributes. The Markov blanket of X_i is all attributes 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 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 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 X_i is in X_j's Markov blanket and X_j is in X_i's Markov blanket.

3. Rule Induction

The goal of binary classification is to distinguish a set of positive examples from a set of negative examples. Rule induction constructs a rule set to discriminate between the two categories. Each rule consists of a body and a head. The body is a conjunction of antecedents, where each antecedent tests a single variable. For discrete data, each antecedent performs an equality test $X_i = x_i$, where X_i is a variable and x_i is a value of that variable. A rule covers (or matches) an example if all antecedents of the rule are true of the example. Rule induction algorithms typically employ a "divide and conquer" approach to constructing the rule set. A rule is induced on the full training set that covers as many positive examples and as few negative ones as possible. The newly covered positive examples are removed from the training set and the process repeats until all positive examples are covered. Top-down, or general-to-specific, induction is the most common strategy for learning a single rule (Cohen, 1995). Each induced rule begins as just a head (i.e., it matches all examples). The induction algorithm then adds antecedents to its body so that it covers a smaller set of examples. The process stops when no specialization improves the rule's accuracy. Notice the similarity between this search strategy and the feature induction approach presented in Subsection 2.4.

Top-down induction is inefficient, testing many feature variations with no support in the data, and highly prone to local optima. Bottom-up, or specific-to-general, rule learning addresses these shortcomings (Domingos, 1996; Muggleton & Feng, 1990). Specific-to-general induction starts with a rule body that contains many antecedents. It then generalizes the rule by removing antecedents from its body, which expands the number of examples the rule matches.

RISE (Domingos, 1996) is one of the most successful bottom-up rule induction algorithms, and it forms the basis for our approach. RISE works as follows. First, it converts each training example into a rule. It then iterates through the rule set. For each rule, it finds the nearest unmatched example of the same class. It generalizes the rule to match the example by removing any antecedent that disagrees with the example. If including the generalized rule in the rule set improves its accuracy, then the new rule is retained. If the rule set already contains an identical rule, it removes the original rule. The algorithm terminates when no generalization improves the rule set's accuracy.

RISE has parallels with both instance-based learning and agglomerative clustering. If RISE accepts no generalizations, it retains the training set as the final classifier and reduces to the nearest-neighbor algorithm. The relation between bottom-up and top-down rule induction is similar to the relation between agglomerative and divisive clustering, and between forward and backward feature selection for classification and regression.

4. Bottom-Up Feature Induction

We now describe BLM (Bottom-up Learning of Markov Networks), a bottom-up feature induction algorithm for Markov networks inspired by RISE. BLM performs a specific-to-general search by starting with each complete training example as a long feature. It

Algorithm 1 BLM(training set TS, set of integers K)

```
$FS = TS$
$score = S(TS, FS)$
repeat
  for all Features $F \in FS$ do
    $(newScore, FS') =$
        Generalize_Feature($TS$, $FS$, $F$, $K$)
    if $newScore > score$ then
      Replace $FS$ with $FS'$
      $score = newScore$
    end if
  end for
until No generalization improves $score$
return $FS$
```

repeatedly generalizes a feature to match its k nearest examples by dropping variables. In this paper, we only consider discrete variables, but the extension to numeric ones is straightforward.

The four key elements of BLM, introduced in the next subsections, are: (i) how to score candidate structures, (ii) how to construct the initial feature set, (iii) how to generalize a feature, and (iv) how the overall algorithm functions.

4.1. Structure Scoring and Weight Learning

As mentioned in Subsection 2.3, several possibilities exist for scoring candidate structures. BLM can use any score function and weight optimization procedure. In particular, we use a score function of the form

$$S(TS, FS, L, \alpha) = L(TS, FS) - \alpha \sum_{f_i \in FS} |f_i| \quad (5)$$

where TS is the training set, FS is the feature set, $L(TS, FS)$ can be the likelihood or pseudo-likelihood, α is a penalty term to avoid overfitting, and $|f_i|$ is the number of tests in feature f_i.

4.2. Initial Feature Set

The initial feature set consists of one feature for each example. The feature is a conjunction over all attributes in the example (as in RISE). This is a good initial feature set as it contains a set of maximally specific features such that each feature matches at least one training example. All duplicate features are removed from the model.

4.3. Feature Generalization

The key step, outlined in Algorithm 2, is generalizing the features. When generalizing a feature f, BLM retrieves the k nearest examples to f that currently do

Algorithm 2 Generalize_Feature(training examples TS, feature set FS, feature F, K)

```
$bestScore = -\infty$
$FS_{best} = FS$
for all $k \in K$ do
  $F' = F$
  $TS_k = k$ nearest examples to $F$
      that do not match $F$
  for all Tests $t$ in $F'$ do
    if $t$ does not match all examples in $TS_k$ then
      Remove $t$ from $F'$
    end if
  end for
  if $F'$ is identical to another feature in $FS$ then
    $FS' = FS$ without $F$
  else
    $FS' = FS$ with $F$ replaced by $F'$
  end if
  $newScore = S(TS, FS')$
  if $newScore > bestScore$ then
    $bestScore = newScore$
    $FS_{best} = FS'$
  end if
end for
$FS' = FS$ without $F$
$newScore = S(TS, FS')$
if $newScore > bestScore$ then
  $bestScore = newScore$
  $FS_{best} = FS'$
end if
return $(FS_{best}, bestScore)$
```

not match it. One possible way to measure the distance between an example and a feature is generalized Hamming distance, which counts the number of variables that would need to be dropped in order for f to match the example. In rule induction, a more sophisticated measure, called the value difference metric, usually performs much better in practice (Domingos, 1996). The metric is designed for classification and the intuition behind it is that two values are similar if they make similar predictions about the class value. We propose a generalized value difference metric $D(f, e)$:

$$D(f, e) = \sum_{c \in f} GVDM(f, e, c) \quad (6)$$

where f is a feature, e is an example, c ranges over the variables in f and

$$GVDM(f, e, c) = \sum_h \sum_{f_i \in f, f_i \neq c} |P(c = h|f_i) - P(c = h|e_{f_i}))|^Q \quad (7)$$

where h ranges over the values that variable c can take on, f_i is the value of the ith variable in f, e_{f_i} is value of the attribute referenced by f_i in e, and Q is an integer. For a variable c that appears in f, GVDM measures how well the other variables in f predict c. The intuition is that if c appears in a feature then the other variables should be good predictors of c.

In the generalization f', BLM drops any variable that causes a mismatch with one of the k nearest examples. Thus f' matches all k examples in addition to the examples that f previously matched.

4.4. The BLM Algorithm

BLM receives a set of training examples, TS, and a set of integers, K, as input. It includes one feature, called an atomic feature, for each variable, to capture its marginal probability. This allows the other features to focus on capturing interactions among the variables. The atomic features remain unchanged throughout learning. Then it creates one feature for each example. Next, the algorithm repeatedly iterates through all non-atomic features. For each feature f, BLM scores several candidate generalizations. For each $k \in K$, it creates one candidate by generalizing f to match its k nearest unmatched examples. The generalization is constructed by dropping the variables that cause the mismatch. At each step, it also considers the generalization of removing f from the model. To score each potential generalization f', BLM replaces f with f' in the model. If f' is identical to another feature in the model, then f' is removed. BLM retains the best scoring candidate model if it improves the score. The process terminates when no generalization improves the score, which indicates that a local optimum has been reached. Algorithm 1 illustrates this process in pseudo-code (ignoring atomic features for simplicity).

4.5. Time Complexity

The worst case time-complexity of BLM compares favorably with Della Pietra et al.'s algorithm. For Markov network structure learning, the time bottleneck is learning feature weights, so we will analyze the number of calls to the weight learning subroutine. Let n be the number of examples and v be the number of variables in the domain.

Each **for** loop in BLM considers generalizing each feature, of which there are at most n. Each loop calls weight learning $O(ng)$ times, where is g is number of k values tried at each generalization step. In the worst case, a loop removes a single variable from a single feature, and the whole algorithm removes all variables from all features, requiring $O(ngv)$ repetitions of the **repeat** cycle. Thus the maximum number of calls to weight learning is $O(gvn^2)$.

Table 1. Data Set Characteristics

Data Set	Train Set Size	Tune Set Size	Test Set Size	Num. Feats.
20 Newsgroups	11,293	3,764	3,764	930
Abalone	3,134	417	626	31
Adult	36,631	4,884	7,327	125
Audio	15,000	2,000	3,000	100
Book	8,700	1,159	1,739	500
Covertype	30,000	4,000	6,000	84
EachMovie	4,524	1,002	591	500
Facebook	9,206	1,765	1,780	698
Jester	9,000	1,000	4,116	100
KDDCup 2000	180,092	19,907	34,955	64
MSNBC	291,326	38,843	58,265	17
MSWeb	29,441	3,270	5,000	294
Netflix	15,000	2,000	3,000	100
NLTCS	16,181	2,157	3,236	16
Plants	17,412	2,321	3,482	69
Reuters-52	6,532	1,028	1,540	941
School	44,443	5,925	8,888	66
Temperature	13,541	1,805	2,708	216
Traffic	3,311	441	662	128
WebKB	2,803	558	838	843
Wine	4,874	650	975	48

To analyze Della Pietra et al.'s algorithm we also need m, the current number of features in the model. In each iteration, the algorithm specializes each feature by conjoining it to every variable and every other feature in the model. Thus, it makes $O(m^2 + mv)$ calls to weight learning in each iteration. Even though Della Pietra et al.'s algorithm is not performing an exhaustive search, it could still end up adding every possible feature to the model, resulting in $O(2^v)$ iterations of search. Thus, the worst case complexity is $O(m^2 2^v + mv2^v)$. In practice, the algorithm requires significantly fewer than $O(2^v)$ iterations to converge.

5. Empirical Evaluation

In this section, we evaluate our approach on 21 real-world data sets. To our knowledge, this is the most extensive empirical evaluation of Markov network structure learning algorithms to date. We compare BLM to the standard top-down Markov network structure learning algorithm (Della Pietra et al., 1997) and the L1 approach of Ravikumar et al. (2010). Additionally, we compare with BUSL (Mihalkova & Mooney, 2007), a bottom-up algorithm for learning the structure of a Markov (logic) network.

We altered Della Pietra et al.'s algorithm to only evaluate candidate features that match at least one example. This simple extension vastly reduces the number of candidate features and greatly improves the algorithm's efficiency. Both BLM and Della Pietra et al.'s algorithm work with any score function. In this paper, we optimize the pseudo-likelihood of the data via the limited-memory BFGS algorithm (Liu & Nocedal, 1989). Optimizing the likelihood of the data is prohibitively expensive for the domains we consider. We used the OWL-QN package (Andrew & Gao, 2007) to implement Ravikumar et al.'s approach and we used the publicly available implementation of BUSL.[1]

We only allow BLM and Della Pietra et al.'s algorithm to construct positive features, which greatly improves efficiency. This is not a restriction, as any Markov network can be represented using only features of this form (Wexler & Meek, 2008). Both approaches estimate a candidate feature's weight by holding the weights of the other features in the model constant. We first describe the data sets we use and then present and discuss our experimental results.

5.1. Tasks

Table 1 describes the characteristics of each data set.[2] Since none of the algorithms can handle numeric data, we discretized all continuous variables by constructing four equal size bins and the bins widths were set using the training data. We binarized all variables (i.e., created one Boolean variable for each value).

From the UCI machine learning repository (Blake & Merz, 2000) we used: Abalone, Adult, Covertype, MSNBC anonymous Web data, Plants and Wine domains. Temperature and Traffic are sensor network data sets and were used in Checketka and Guestrin (2007). The versions of the EachMovie, KDDCup and MSWeb domains came from Lowd and Domingos (2008). The School domain (Yang et al., 2002) contains information about math examinations results in England and the National Long Term Care Survey (NLTCS) data consist of binary variables that measure an individual's ability to perform different daily living activities. The Facebook data comes from the social networking site. We constructed examples by comparing two individuals according to the attributes available on their profiles.

We used three text domains: 20 Newsgroups, Reuters-52 and WebKB. Finally, we considered several collaborative filtering problems: Audio, Book (Ziegler et al., 2005), Jester (Goldberg et al., 2001) and the Netflix challenge data. We reduced each problem to the equivalent of "rated " or "not-rated."

5.2. Methodology

We performed two sets of experiments. The first experiment compared the accuracy and run time performance of the BLM compared to the other three algorithms. We also included the performance achieved by just using the atomic features which capture the marginal probability of each variable. This baseline provides a "sanity check" to ensure that we are actually learning something in each domain. In the second set of experiments, we performed two lesions studies on BLM. First, we compared against using a fixed k size of 1. Second, we compared against using generalized Hamming distance as the distance measure.

Like Lee et al. (2007), 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 compute $CMLL(X = x) = \sum_{i \in Q} \log P(X_i = x_i|E)$. 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 (Poon & Domingos, 2006). For all three domains, we set the burn-in to 1,000 samples and then computed the probability using the next 10,000 samples.

For all algorithms, we imposed a 24-hour time limit for training. For BLM, we used $K = \{1, 2, 5, 10, 20, 25, 50, 100, 200\}$ and $Q = 1$ for GVDM. We did not tune these parameters. For BLM, we varied how often we relearned the full set of feature weights and considered doing so after every $1, 5, 10$ and 25 accepted feature generalizations. We used the tuning set to pick the best setting for this parameter. For all algorithms we tried a variety of values for penalty α, and used the tuning set to select the best setting for each domain. For the L1 approach, we tried both methods described in Subsection 2.4 to enforce consistency. BUSL ran out of memory when learning on the full training set for each domain. We tried various different subsamples of the data and report the best test set results that we obtained.

5.3. Results

Table 2 presents CMLLs and run times for all 21 domains. CMLL is averaged over all test examples and run time is reported in minutes. BLM significantly

[1] Code for BLM and Della Pietra et al. is available at http://alchemy.cs.washington.edu/papers/davis10a/.

[2] Data sets and their descriptions are available at http://alchemy.cs.washington.edu/papers/davis10a/.

Table 2. Experimental results. Atomic is a model that assumes all variables are independent. CMLL is the average CMLL per test example. Run time is in minutes.

	CMLL					Run Time (Minutes)			
Data Set	BLM	DP	L1	BUSL	Atomic	BLM	DP	L1	BUSL
20 Newsgroups	-147.461	-158.552	-139.515	-155.005	-160.610	468.9	1440.0	347.7	468.0
Abalone	-8.075	-7.780	-74.574	-11.547	-17.657	0.1	2.5	0.1	5.8
Adult	-28.673	-24.626	-48.982	-24.241	-24.903	260.9	22.0	18.7	16.2
Audio	-37.385	-39.224	-37.276	-42.163	-49.362	434.3	1398.5	2.7	1020.2
Book	-34.768	-39.254	-35.517	-38.522	-41.308	47.3	1440.0	25.5	208.4
Covertype	-19.464	-18.959	-24.527	-26.286	-28.393	181.9	1440.0	4.3	11.1
EachMovie	-58.852	-67.175	-52.957	-67.540	-84.102	41.3	1440.0	62.1	235.8
Facebook	-9.749	-12.107	-15.409	-9.216	-12.840	1.5	5.6	57.6	1046.8
Jester	-53.025	-53.999	-53.226	-62.126	-63.891	350.2	1440.0	3.2	11.3
KDDCup 2000	-2.099	-2.112	-2.165	-3.104	-2.456	62.9	1440.0	10.4	3.4
MSNBC	-5.892	-5.957	-6.332	-6.735	-6.780	203.5	1440.0	1.2	4.8
MSWeb	-8.936	-9.187	-9.476	-10.476	-11.720	64.8	1440.0	40.5	262.3
Netflix	-56.598	-57.429	-56.129	-59.945	-64.578	1367.8	1440.0	4.1	228.6
NLTCS	-5.253	-5.220	-5.278	-8.414	-9.241	24.5	15.8	0.1	3.9
Plants	-10.960	-11.143	-10.962	-26.045	-31.321	514.6	1440.0	3.2	22.5
Reuters-52	-94.249	-108.449	-85.950	-105.956	-112.880	170.9	1440.0	148.7	1440.0
School	-16.983	-16.170	-25.330	-21.564	-22.430	213.0	1440.0	4.9	4.8
Temperature	-52.497	-65.847	-140.391	-78.375	-113.026	569.3	1440.0	21.0	2.6
Traffic	-34.475	-29.266	-210.014	-40.251	-72.014	6.6	1440.0	1.9	35.8
WebKB	-166.811	-178.030	-151.615	-174.774	-183.875	49.9	1440.0	55.3	1048.0
Wine	-24.559	-23.765	-50.615	-27.138	-27.261	5.8	117.9	0.1	3.5

outperforms the other three approaches in terms of the learned model's accuracy. It finishes first or second in 20 of 21 domains in terms of accuracy.

BLM outperforms Della Pietra et al.'s algorithm on 14 of the 21 domains. BLM significantly outperforms Della Pietra et al. at the 0.032 significance level according to a Wilcoxon signed-ranks test. BLM has a faster run time than Della Pietra et al. on 19 domains. BLM performance can be explained by its greater ability to avoid local optima and its improved efficiency, which allows it to learn larger models as well as models that contain longer features. On 20 domains, its learned model contains more features than Della Pietra et al.'s model. Additionally, on 20 domains, the average length of a feature included in BLM's learned model is higher than that of Della Pietra et al.'s learned model.[3]

BLM outperforms Ravikumar et al.'s L1 approach on 15 of the 21 domains. BLM outperforms L1 at the 0.043 significance level according to a Wilcoxon signed-ranks test. BLM is better on the majority of the domains because many problems require longer features to capture the regularities present in them. In domains where short features are sufficient or that are highly sparse, L1 should do well. BLM can never induce more features than there are training examples. L1 wins on all three text domains as the pairwise feature induction approach of L1 corresponds to learning a bigram model of text, which is known to work well in practice. The four domains where L1 results in the largest improvement all have a large number of attributes ($\geq$ 500) and L1's learned model contained three to ten times more features than there are examples in the training set. In general, the L1 approach is faster than BLM. However, in four of the six domains in which L1 wins on accuracy, it is either slower or has comparable run time to BLM.

BLM outperforms BUSL at the 0.0006 significance level according to a Wilcoxon signed-ranks test. The run time of BUSL is not comparable to the other algorithms due to the subsampling we needed to perform in order for it run. Only on three domains could it run on more than 200 examples and it was never able to run on more than $1,000$ examples.

In the second set of experiments, BLM outperformed its two variants. It beat using a fixed $k = 1$ on 14 of 21 domains. The difference in performance is significant at the 0.012 significance level according a Wilcoxon signed-ranks test. Furthermore, running with a fixed $k = 1$ is slower on 20 of 21 domains. BLM with GVDM outperforms BLM with Hamming distance 14 of 21 do-

[3]Full results are available in the online appendix http://alchemy.cs.washington.edu/papers/davis10a/.

mains. GVDM outperforms Hamming distance at the 0.077 significance level according a Wilcoxon signed-ranks test. There is no general trend in run time when comparing the distance metrics. These results confirm our decisions to consider multiple k sizes and to use GVDM as the distance metric.[4]

6. Conclusions and Future Work

This paper proposed bottom-up search as an alternative approach to top-down search for learning the structure of a Markov network. Bottom-up learning overcomes some of the inefficiency and myopia of top-down learning by using the data more directly to guide the search. Our BLM algorithm starts with complete examples as the initial features, and gradually generalizes them to cover nearby high-probability regions. Experiments in three domains show that this approach is more accurate than the standard top-down one. Furthermore, BLM is significantly more efficient than the standard top-down approach.

Directions for future work include: theoretical analysis of BLM; measuring how the training-set size affects performance; validation on additional domains; extending the algorithm to relational domains; etc.

Acknowledgments

This research was partly funded by ARO grant W911NF-08-1-0242, AFRL contract FA8750-09-C-0181, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, and ONR grant N00014-08-1-0670. 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, DARPA, NSF, ONR, or the United States Government.

References

Andrew, G. and Gao, J. Scalable training of L1-regularized log-linear models. In *Proc. ICML'07*, pp. 33–40, 2007.

Besag, J. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.

Blake, C. and Merz, C. J. UCI machine learning repository. Technical report, Dept. ICS, UC Irvine, CA, 2000.

Chechetka, A. and Guestrin, C. Efficient principled learning of thin junction trees. In *NIPS 20*, pp. 273–280, 2007.

[4]Full results are available in the online appendix http://alchemy.cs.washington.edu/papers/davis10a/.

Cohen, W. W. Fast effective rule induction. In *Proc. ICML'95*, pp. 115–123, 1995.

Della Pietra, S., Della Pietra, V., and Lafferty, J. Inducing features of random fields. *IEEE Trans. on Pat. Anal. and Mach. Intel.*, 19:380–392, 1997.

Domingos, P. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168, 1996.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (eds.). *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, UK, 1996.

Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retrieval*, 4(2):133–151, 2001.

Höfling, H. and Tibshirani, R. Estimation of sparse binary pairwise Markov networks using pseudo-likelihood. *J. Mach. Learn. Res.*, 10:883–906, 2009.

Kok, S. and Domingos, P. Learning the structure of Markov logic networks. In *Proc. ICML'05*, pp. 441–448, 2005.

Kulesza, A. and Pereira, F. Structured learning with approximate inference. In *NIPS 20*, pp. 785–792, 2007.

Lee, S.-I., Ganapathi, V., and Koller, D. Efficient structure learning of Markov networks using L1-regularization. In *NIPS 19*, pp. 817–824, 2007.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

Lowd, D. and Domingos, P. Learning arithmetic circuits. In *Proc. UAI'08*, pp. 383–392, 2008.

McCallum, A. Efficiently inducing features of conditional random fields. In *Proc. UAI'03*, pp. 403–410, 2003.

Mihalkova, L. and Mooney, R. J. Bottom-up learning of Markov logic network structure. In *Proc. ICML'07*, pp. 625–632, 2007.

Muggleton, S. H. and Feng, C. Efficient induction of logic programs. In *Proc. ALT'90*, pp. 368–381, 1990.

Narasimhan, M. and Bilmes, J. PAC-learning bounded tree-width graphical models. In *Proc. UAI'04*, 2004.

Poon, H. and Domingos, P. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proc. AAAI'06*, pp. 458–463, 2006.

Ravikumar, P., Wainwright, M., and Lafferty, J. High-dimensional ising model selection using L1-regularized logistic regression. *Ann. of Stats.*, 2010.

Wexler, Y. and Meek, C. Inference for multiplicative models. In *Proc. UAI'08*, pp. 595–602, 2008.

Yang, M., Goldstein, H., Browne, W., and Woodhouse, G. Multivariate multilevel analyses of examination results. *J. Roy. Stat. Soc., Ser. A*, 165:137–153, 2002.

Ziegler, C., McNee, S., Konstan, J., and Lausen, G. Improving recommendation lists through topic diversification. In *Proc. WWW'05*, pp. 22–32, 2005.