Learning Markov Networks with Arithmetic Circuits

Daniel Lowd, University of Oregon
<lowd@cs.uoregon.edu>

Pedram Rooshenas, University of Oregon
<pedram@cs.uoregon.edu>

Introduction

- Markov networks (MNs) are a powerful and flexible representation, but using them is difficult:
  - Exact inference is typically intractable.
  - Weight learning is also hard, since the likelihood and its gradient are intractable to compute.
  - Previous work has examined restricted classes of MNs where learning and inference are easy.
- We introduce ***the first general-purpose*** learning algorithm for tractable MNs, and compare it to 4 baselines on 12 benchmark datasets.
- Our method uses arithmetic circuits, an inference representation similar to sum-product networks, to learn MNs with arbitrary conjunctive feature and high treewidth.

Markov Networks and Arithmetic Circuits

A ***Markov network (MN)*** represents a probability distribution as a normalized product of factors:

$$P(X) = \frac{1}{z}\prod_c \phi_c(D_c)$$

where $D_c \subset X$ represents the variables in the domain of factor c and Z is the normalization constant.

If all probabilities are positive, the distribution can be represented as an equivalent ***log-linear model***:

$$\log P(X) = \sum_i w_i f_i(D_i) - \log z$$

The ***network polynomial*** for a Markov network is a polynomial with an exponential number of terms, one for each possible state of random variables. Each term is the product of indicator variables (λ_{x_i}) and the parameters ($\theta_j = e^{w_j}$) of the satisfied features

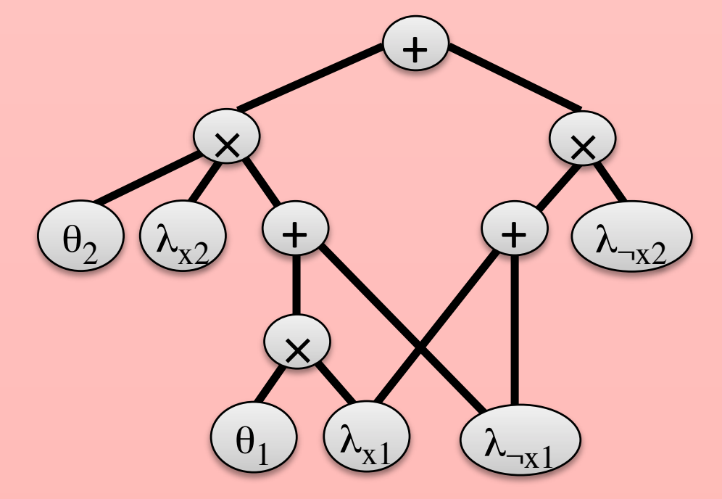
Example: Network polynomial of Markov network over Boolean variables x_1 and x_2 with features $f_1 = x_1 \wedge x_2$ and $f_2 = x_2$:

$$f(\lambda_{x_1}, \lambda_{x_2}, \lambda_{\neg x_1}, \lambda_{\neg x_2}) = \lambda_{x_1}\lambda_{x_2}\theta_1\theta_2$$
$$+ \lambda_{x_1}\lambda_{\neg x_2} + \lambda_{\neg x_1}\lambda_{x_2}\theta_2 + \lambda_{\neg x_1}\lambda_{\neg x_2}$$

For this Markov network, $z = f(1,1,1,1)$ and

$$P(x_1 = 1, x_2 = 0) = f(1,0,0,1)$$

An ***arithmetic circuit (AC)*** is a rooted, directed acyclic graph that can compactly represent network polynomial in some cases.



Evaluating or differentiating the AC with or without evidence can be done in linear time in the size of circuit. Therefore, we can perform efficient inference in any MN if we have compact representation of it as an AC. [Darwiche, 2003]

Algorithm: ACMN

ACMN performs a greedy search through structure space. The initial structure is the set of all single variable features. The search operations are to take an existing feature, f, and ***split*** it on another variable, v, creating two new features: $f \wedge v$ and $f \wedge \neg v$

Splits are scored according to their effect on the log-likelihood of the MN and the size of the corresponding AC:

$$Score(s) = \Delta_{ll}(s) - \gamma\Delta_e(s)$$

Δ_{ll} measures how much the split will increase the log-likelihood.

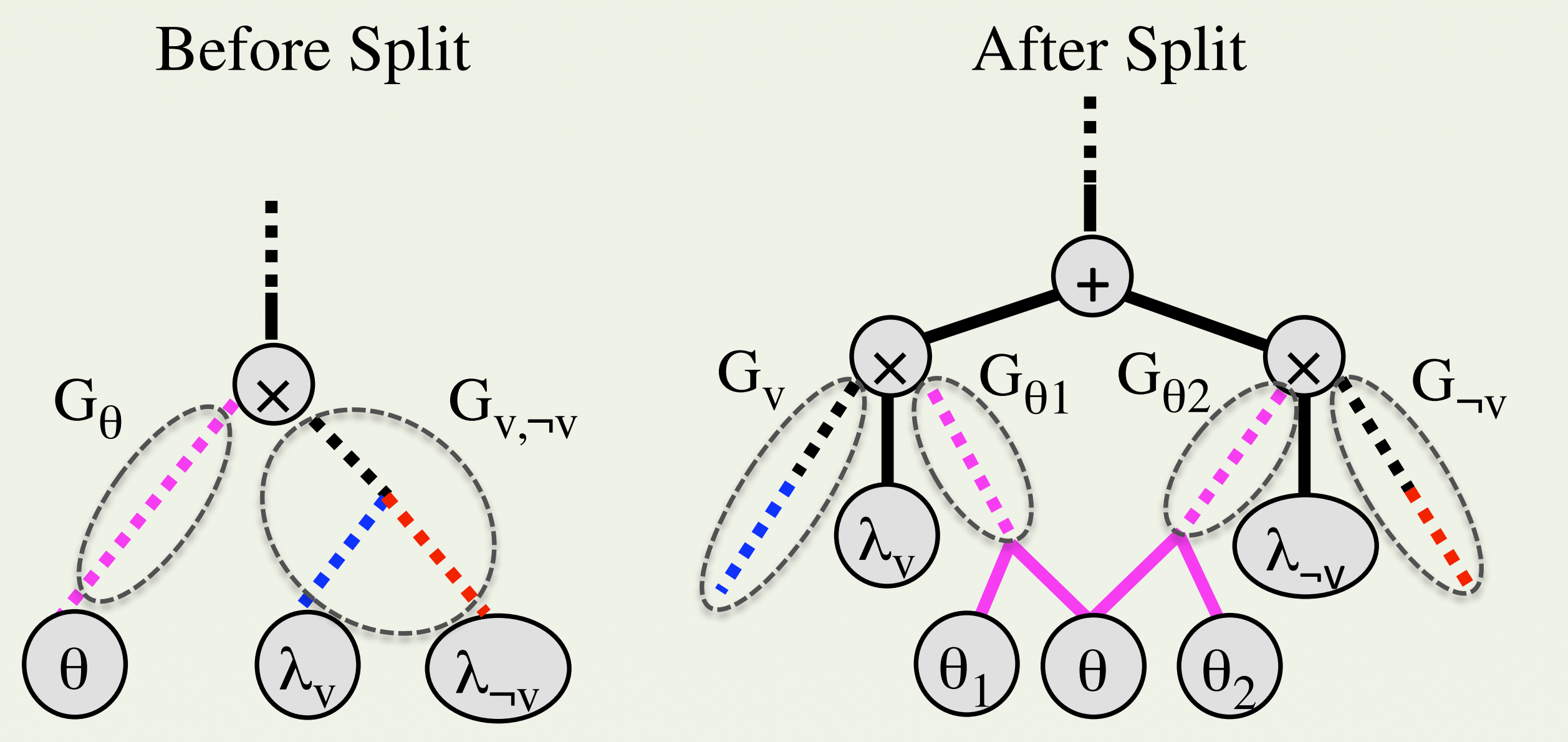
- Measuring the exact effect would require jointly optimizing all model parameters along with the parameter for the two new features.
- To make scoring more efficient, we measure the log-likelihood gain from modifying only the weights of the two new features, keeping all others fixed

ACMN computes the gain by solving a simple two dimensional convex optimization problem, which depends only on the empirical counts of the new features in the data and their expected counts in the model. Used also by Della Piera et. al. (1997) and McCallum (2003).

By conditioning the AC on feature f and differentiating it with respect to indicator variables, ACMN computes the expectations $E[f \wedge x_i]$ and $E[f \wedge \neg x_i]$ for all variables in a single pass over the AC which takes linear time in the size of the circuit.

$\Delta_e(s)$ denotes the number of edges that would be added to the AC if this split were included.

Computing the $\Delta_e(s)$ has similar time complexity to actually performing the split.



ACMN Algorithm

```
Create initial product of marginals circuit
Create initial split list
Until convergence:
    For each split in list
        Apply split to circuit
        Score result
        Undo split
    If no split has positive score:
        Reduce per-edge penalty and continue
    Else:
        Apply highest-scoring split to circuit
        Add new child splits to list
        Remove inconsistent splits from list
```

How to split a circuit

```
D: Parameter nodes to be split
V: Indicators for the splitting variable
M: First mutual ancestors of D and V
For each indicator λ in V,
    Copy all nodes between M and D or V, conditioned on λ.
For each m in M,
    Replace children of m that are ancestors of D or V with a sum over copies of the ancestors times the λ each copy was conditioned on.
```

ACMN and LearnAC (ACBN)

ACMN is similar to the LearnAC [Lowd et al., 2008].

- LearnAC learns Bayesian networks (BNs) where each conditional probability distribution (CPD) is a tree.
- ACMN learns MNs where the distribution is a log-linear model and each feature is conjunction of feature tests.

Every BN with tree CPDs can easily be expressed as a set of conjunctive features, but the converse is not true. Therefore, ***ACMN should be more expressive than LearnAC and able to learn better models***.

Dataset	Vars	ACMN	ACBN	LTM
NLTCS	16	**-6.01**	-6.02	-6.49
MSNBC	17	**-6.04**	**-6.04**	-6.52
KDDCup 2000	64	**-2.15**	-2.16	-2.18
Plants	69	-12.89	**-12.85**	-16.39
Audio	100	**-40.32**	-41.13	-41.90
Jester	100	**-53.35**	-54.43	-55.17
Netflix	100	**-57.26**	-57.75	-58.53
MSWeb	294	**-9.77**	-9.81	-10.21
Book	500	-35.62	-36.02	**-34.22**
WebKB	839	-161.30	-159.85	**-156.84**
20 Newsgroup	889	-159.56	-159.65	**-156.77**
Reuters-52	910	-89.54	**-89.27**	-91.23

We computed average log-likelihood of ACMN, ACBN and LTM which is intractable for L1 and GSSL. ACMN is the most accurate algorithm on 6 of the 12 datasets.
Due to imposed limits on running time, ACMN learns relatively simpler models on high-dimensional datasets.

We approximated the KL divergence between the BN and MN ACs using sampling. We found that the KL divergence between the two is often much larger than their difference in log-likelihood on the test data, suggesting that they do indeed learn significantly different distributions:

$$KL(P \| Q) = E_P[\log(P(x)/Q(x)]$$
$$\approx \frac{1}{m}\sum_{i=1}^{m}\log(P(x^{(i)})/Q(x^{(i)}))$$

Dataset	Original	Generated
NLTCS	0.01	0.08
MSNBC	0.00	0.02
KDDCup 2000	0.01	0.08
Plants	0.04	1.43
Audio	0.81	2.46
Jester	1.08	3.14
Netflix	0.49	2.57
MSWeb	0.04	0.56
Book	0.49	0.73
WebKB	1.45	14.57
20 Newsgroup	0.09	10.13
Reuters-52	0.27	6.98

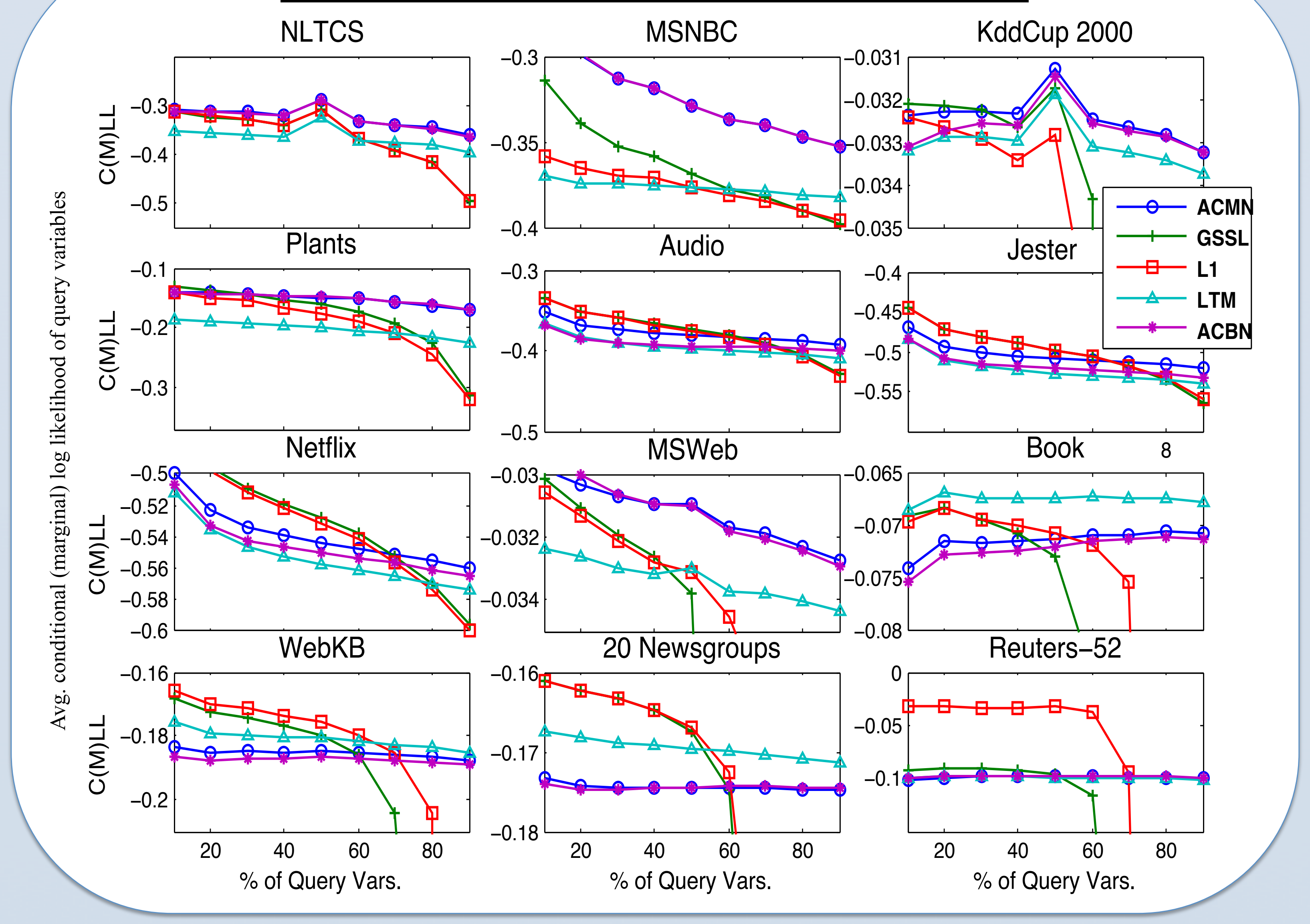
Experiments

We compared ACMN to four state-of-the-art algorithms.
- MN baselines: GSSL [Van Haaren et al., 2012] and L1-regularized logistic regression [Ravikumar et al., 2009] as MN baslines;
- Tractable baselines: learning latent tree models (LTM) [Choi et al., 2011] and LearnAC algorithm (ACBN) [Lowd et al., 2008] which searches through Bayesian network structures rather than MNs.

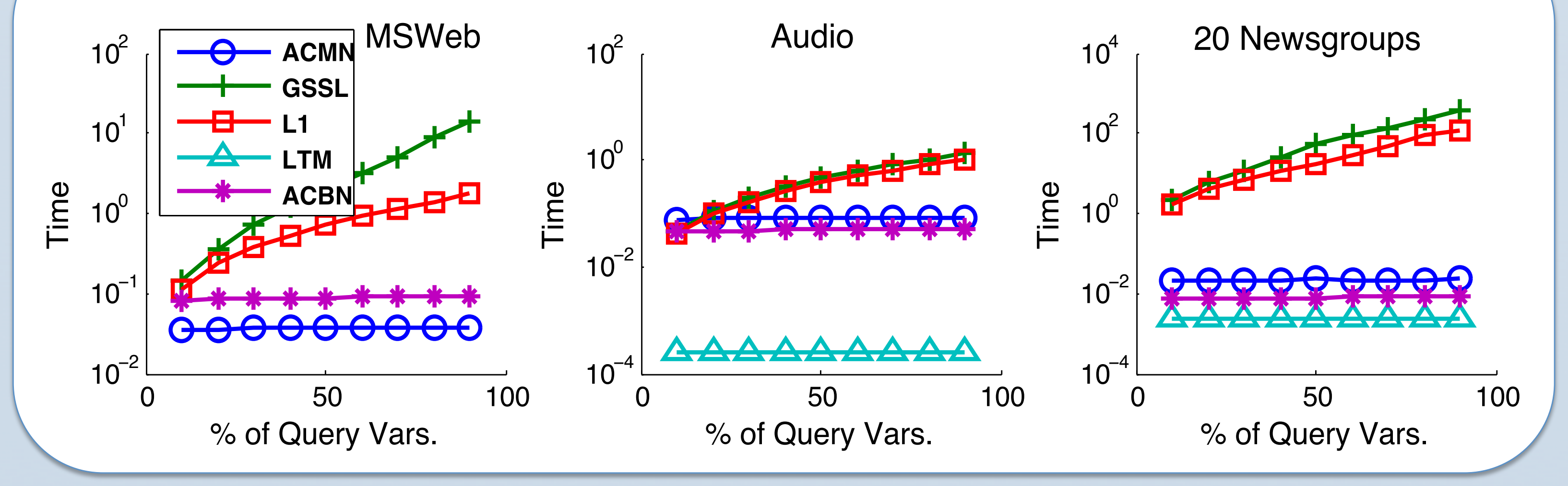
The objective function of GSSL and L1 is pseudo log-likelihood while ACMN, ACBN, and LTM optimize log-likelihood.

We compared the exact conditional log-likelihood (CLL) of the query variables given the evidence ($\log P(X = x \mid E = e)$). For L1 and GSSL, we computed the conditional marginal log-likelihood (CMLL) instead. CMLL is used when joint probabilities are hard to estimate: $\sum_i \log P(X_i = x_i \mid E = e)$

Inference accuracy vs. fraction of evidence variables



Query time vs. fraction of evidence variables



Results and Future Work

ACMN dominates GSSL and L1 on every dataset with < 20% evidence, both in inference speed and accuracy.
ACMN is usually more accurate than LTM, especially on datasets with fewer variables.
ACMN is often more accurate than ACBN, and seems to learn a significantly different distribution even when the accuracy is similar.

A final direction is learning ACs with latent variables. Ideally, this could give ACMN the advantages of LTM on datasets where clustering structure was present, while maintaining the flexibility of the unrestricted conjunctive feature representation.

Source code: http://libra.cs.uoregon.edu/