Learning Arithmetic Circuits

Daniel Lowd and **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{*lowd,pedrod*}*@cs.washington.edu*

Abstract

Graphical models are usually learned without regard to the cost of doing inference with them. As a result, even if a good model is learned, it may perform poorly at prediction, because it requires approximate inference. We propose an alternative: learning models with a score function that directly penalizes the cost of inference. Specifically, we learn arithmetic circuits with a penalty on the number of edges in the circuit (in which the cost of inference is linear). Our algorithm is equivalent to learning a Bayesian network with context-specific independence by greedily splitting conditional distributions, at each step scoring the candidates by compiling the resulting network into an arithmetic circuit, and using its size as the penalty. We show how this can be done efficiently, without compiling a circuit from scratch for each candidate. Experiments on several real-world domains show that our algorithm is able to learn tractable models with very large treewidth, and yields more accurate predictions than a standard context-specific Bayesian network learner, in far less time.

1 EXPERIMENTS

1.1 DATASETS

We evaluated our methods on three widely used real-world datasets. The KDD Cup 2000 clickstream prediction dataset (**?**) consists of web session data taken from an online retailer. Using the subset of **?** (**?**), each example consists of 65 Boolean variables, corresponding to whether or not a particular session visited a web page matching a certain category. Anonymous MSWeb is visit data for 294 areas (Vroots) of the Microsoft web site, collected during one week in February 1998. It can be found in the UCI machine learning repository (**?**). EachMovie[1] is a collaborative filtering dataset in which users rate movies they have seen. We took a 10% sample of the original dataset, focused on the 500 most-rated movies, and reduced each variable to "rated" or "not rated". For KDD Cup and MSWeb, we used the training and test partitions provided with the datasets. For EachMovie, we randomly selected 10% of the data for the test set and used the remainder for training.

Table 1: Summary of experimental datasets.

Domain	Vars.	Train Exs.	Test Exs.	Density
KDD Cup	65	199,999	34,955	0.0079
MSWeb	294	32,711	5,000	0.0102
EachMovie	500	6,117	591	0.0581

Basic statistics for each dataset are shown in Table **??**. Density refers to the fraction of non-zero entries across all examples and all variables.

1.2 LEARNING

For each dataset, we randomly split the training data into tuning and validation sets, corresponding to 90% and 10% of the training data, respectively. All parameters were tuned by training models on the tuning data and selecting the parameter sets that led to the highest log likelihood of the validation set. Finally, models were retrained using the full training set. All experiments were run on CPUs with 4 GB of RAM running at 2.8 GHz.

We used two versions of the algorithm for learning arithmetic circuits from Section 4: AC-Greedy, which guarantees that we pick the best split in each iteration, and AC-Quick, which uses a heuristic to avoid recomputing edge costs but may sometimes choose worse splits. We varied the per-edge cost k_e from 1.0 to 0.01. Not surprisingly,

[1]Provided by Compaq at http://research.compaq.com/SRC/-eachmovie/; no longer available for download, as of October 2004.

our models were most accurate on the validation set with low per-edge costs (0.01 or 0.02). We also tuned the per-parameter cost k_p. For KDD Cup, the best cost was 0.0; for MSWeb and EachMovie, the best costs were 1.0 for greedy ACs and 0.5 for quick ACs.

We used the WinMine Toolkit(**?**) as a baseline. WinMine implements the algorithm for learning Bayesian networks with local structure described in Section 2 (**?**), and has a number of other state-of-the-art features. We tuned WinMine's multiplicative per-parameter penalty κ; the best values were: 1 (no penalty) for KDD Cup, 0.1 for MSWeb, and 0.01 for EachMovie. We looked into using thin junction trees as a second baseline, but they do not scale to datasets of these dimensions.

A summary of the learned models appears in Table **??**. For each dataset, we report the log-likelihood per example on the test data, the number of edges in the arithmetic circuit, the number of leaves across all decision trees, the average and maximum number of parents across all variables, the treewidth (estimated using a min-fill heuristic), the number of edges generated by compiling the Bayesian network using c2d[2], and the training time. On each model for which c2d ran out of memory, we obtained a lower bound by compiling a model with fewer splits, obtained by halting the learning process early. We varied the number of splits until we found the most complex sub-model that could still be compiled, within 10 splits. For WinMine, the chosen sub-models had less than one quarter of the original splits.

The test-set log-likelihoods of the AC learners and WinMine are very similar, with WinMine having a slight edge. This is not surprising, given that WinMine is free to choose expensive splits. Perhaps more remarkable is that this freedom translates to very little improvement in likelihood. The difference in accuracy between quick and greedy ACs is negligible except in the case of EachMovie, where the greedy AC is actually less accurate because it did not converge in the allowed time (72h).

Not surprisingly, WinMine is much faster than the AC learners. It is worth noting that the cost of learning is only incurred once, while the cost of inference is incurred many times. Also, the AC learner directly outputs an arithmetic circuit, while WinMine's Bayesian network would still have to be compiled into one, which can be very time-consuming. Finally, the quick heuristic offers up to an order-of-magnitude speedup with similar accuracy; additional heuristics might offer additional improvements.

[2]Available at http://reasoning.cs.ucla.edu/c2d/. We also tried using the ACE package, but it does not support decision tree CPDs and, for our models, tabular CPDs would be prohibitively large.

[3]AC-Greedy did not finish running in the maximum allowed time of 72h. As a result, it has fewer edges and lower log-likelihood than AC-Quick.

Table 2: Summary of Learned Models

KDD Cup	AC-Greedy	AC-Quick	WinMine
Log-likelih.	-2.16	-2.16	-2.16
Edges	382K	365K	
Leaves	4574	4463	2267
Treewidth	38	38	53
c2d edges	$>$18.2M	3664k	$>$39.5M
Time	50h	3h	3m

MSWeb	AC-Greedy	AC-Quick	WinMine
Log-likelih.	-9.85	-9.85	-9.69
Edges	204K	256K	
Leaves	1353	1870	1710
Treewidth	114	127	118
c2d edges	$>$23.5M	$>$44.6M	$>$63.5M
Time	8h	3h	2m

EachMovie	AC-Greedy	AC-Quick	WinMine
Log-likelih.	-55.7	-54.9	-53.7
Edges	155K	372K	
Leaves	4070	6521	4830
Treewidth	35	54	281
c2d edges	207k	855k	$>$27.3M
Time	$>$72h[3]	22h	3m

1.3 INFERENCE

For each dataset, we used the test data to generate queries with varied numbers of randomly selected query and evidence variables. Each query asked the probability of the configuration of the query variables in the test example conditioned on the configuration of the evidence variables in the same test example.

We estimate inference accuracy as the mean log probability of the test examples's configuration across all test examples. This is an approximation (up to an additive constant) of the Kullback-Leibler divergence between the inferred distribution and the true one, estimated using the test samples. For KDD Cup and MSWeb, we generated queries from 1000 test examples; for EachMovie, we generated queries from all 593 test examples.

For the arithmetic circuits, we used exact inference. For the Bayesian networks learned using WinMine, we used Gibbs sampling. We initialized the sampler to a random state, ran it for a burn-in period, and then collected samples to estimate the probability of the queried marginal or conditional event. All estimates were smoothed by uniformly distributing a count of 1 across all states of the query variables. Since convergence is difficult to diagnose and may take prohibitively long, we ran Gibbs sampling in four scenarios: fast (one chain, 100 burn-in iterations, 1000 sampling iterations); medium (ten chains, 100 burn-in itera-

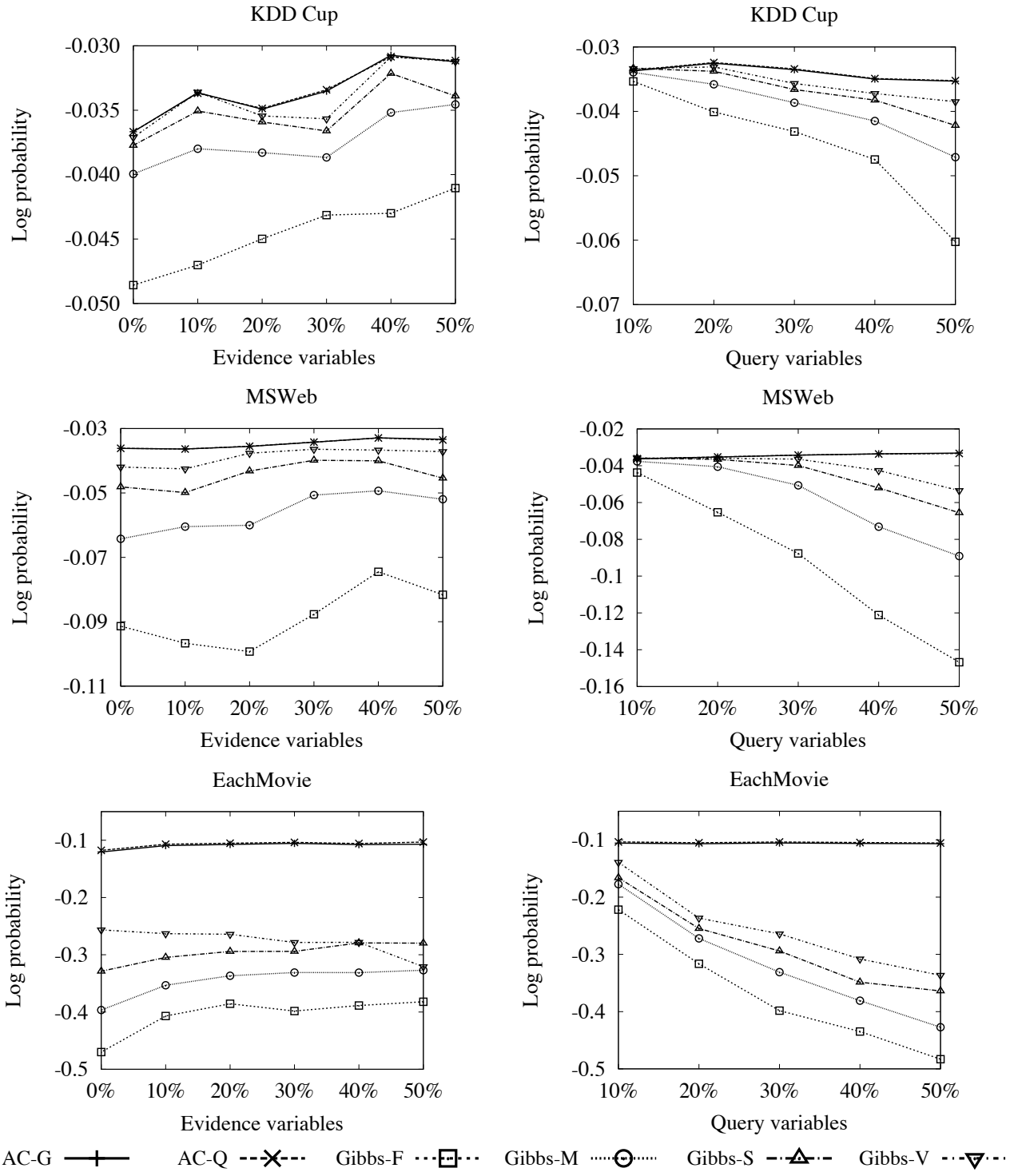

Figure 1: Conditional log probability per query variable, per query. In the legend, AC-G refers to AC-Greedy and AC-Q refers to AC-Quick. Gibbs-F, Gibbs-M, Gibbs-S and Gibbs-V refer to the fast, medium, slow, and very slow Gibbs sampling scenarios, respectively.

Table 3: Average inference time per query.

Algorithm	KDD Cup	MSWeb	EachMovie
AC-Greedy	194ms	91ms	62ms
AC-Quick	198ms	115ms	162ms
Gibbs-Fast	1.46s	1.89s	7.22s
Gibbs-Medium	11.3s	15.6s	42.5s
Gibbs-Slow	106s	154s	452s
Gibbs-VerySlow	1124s	1556s	3912s

tions, 1000 sampling iterations); slow (ten chains, 1000 burn-in iterations, 10,000 sampling iterations); and very slow (ten chains, 10,000 burn-in iterations, 100,000 sampling iterations).

Figure **??** shows the relative accuracy of the different methods on each dataset. Per-variable query log-likelihood is on the y axis. In the graphs on the left, each query included 30% of the variables in the domain, conditioned on 0% to 50% of the domain variables as evidence. In the graphs on the right, the number of query variables varies from 10% to 50%, conditioned on 30% of the variables in the domain as evidence. Inference times (averaged over all queries) are listed in Table **??**. Note that AC inference times are in milliseconds, while Gibbs inference times are in seconds.

The ACs were roughly one order of magnitude faster than the fastest runs of Gibbs sampling, and three orders of magnitude faster than the slowest. Except when the number of query variables is very small, the ACs also easily dominate even the slowest runs of Gibbs sampling on accuracy. Because of the approximate inference, the slightly higher test-set log-likelihood of WinMine's models does not translate into higher accuracy in answering queries. Presumably, given enough time Gibbs sampling will eventually catch up with the ACs in accuracy, but by then it will be many orders of magnitude slower. Further, Gibbs sampling (like other approximate inference methods) requires tuning for best results, and we can never be sure that it has converged. In contrast, the AC inference is reliable, the time it takes is predetermined, and the time is short enough for online or interactive use.