Tutorial for Libra 0.4.0

Daniel Lowd <lowd@cs.uoregon.edu>

July 6, 2011

Introduction

This document describes how to use the Libra toolkit for various learning and inference problems. All necessary files are included in the `doc/tutorial/` directory of the standard distribution.

Quick Installation

Libra was designed to run from the command line under Linux, Mac OS X, or Windows (with the help of Cygwin). The following assumes that you have the OCaml programming language installed, available from most package managers, such as MacPorts, and online from: `http://caml.inria.fr/`.

First, unpack the source distribution:

```
tar -xzvf libra-tk-0.4.0.tar.gz
cd libra-tk-0.4.0
```

This creates the `libra-tk-0.4.0/` directory and makes it your working directory. For the remainder of this document, all paths will be relative to this directory.

Next, build the executables:

```
cd src
make clean; make
cd ..
```

All programs should now be present in the directory `bin/`. For convenience, you may wish to add this directory to your path when working with Libra. Here are the commands to do so under the `bash` shell:

```
cd bin
export PATH=$PATH:‘pwd‘
cd ..
```

Note that `pwd` is surrounded by backticks (‘), not apostrophes (’).

Finally, to continue with this tutorial, change to the tutorial directory:

```
cd doc/tutorial
```

Task Description

In this tutorial, you will train a model from data in two different ways, evaluate model accuracy on held-out data, and answer queries exactly and approximately.

As our dataset, we will use the Microsoft Anonymous Web Data[1], which records the areas (Vroots) of microsoft.com that each user visited during one week in February 1998. A small subset of this data is present in the tutorial directory, already converted into Libra's data format: each line is one example, represented as a list of comma-separated values (one discrete value per variable).

Step 1: Train a Chow-Liu Tree

To train a Chow-Liu tree, use the **cl** program:

```
cl -i msweb.data -s msweb.schema -o msweb-cl.xmod -prior 1
```

The most important options are `-i`, for indicating the training data, and `-o`, for indicating the output file. Libra supports two formats for Bayesian networks, BIF and XMOD. XMOD is preferred, since it supports both tree and table CPDs. Programs in Libra will automatically guess the type of Bayesian network you want by looking at the suffix of the filename – `.xmod` for XMOD and `.bif` for BIF.

The `-s` option allows you to specify a variable schema, the number of values for each variable. If unspecified, Libra will guess from the training data. `-prior` specifies the prior counts to use when estimating the parameters of the network, for smoothing.

The `-log` flag allows you to redirect log output to another file instead of standard output. You can use the flag `-v` to enable more verbose output, or `-debug` to turn on detailed debugging output. These three flags are available in every program, although for some programs they have minimal effect. Try them with **cl** to see what happens.

To see a list of all options, run **cl** without any arguments. This also works with any other program in the toolkit.

Step 2: Circuit Compilation

We can convert our newly-trained BN into an equivalent arithmetic circuit using the **acve** program:

```
acve -m msweb-cl.xmod -o msweb-cl.ac
```

acve is a modified implementation of the AC variable elimination algorithm (ACVE) proposed by (Chavira and Darwiche, 2007)[2]. In addition to compiling simple models (such as our tree-structured BN), ACVE can handle many complex models with internal structure by representing CPDs as algebraic decision diagrams (ADDs).

If you run **acve** with no arguments, you will notice a number of options related to thresholds and pruning. These are experimental features for approximately compiling BNs or MNs when the exact compilation would be intractable. The use of these options is not currently recommended.

[1] Available from the UCI repository at: `http://kdd.ics.uci.edu/databases/msweb/msweb.html`

[2] The key difference is that our ADDs handle multivalued variables using k-way splits rather than introducing additional variables. We've found that this sometimes leads to more compact circuits, and sometimes less compact circuits.

Step 3: File Information

To obtain basic information about most file types supported by Libra, use the `fstats` command:

```
fstats -i msweb.data
fstats -i msweb-cl.xmod
fstats -i msweb-cl.ac
```

The output of `fstats` will depend on the file type as well as its contents. Below is the result of running `fstats -i msweb.data`:

```
Filename: msweb.data
Filetype: Data
Points: 1000
Variables: 294
Schema: 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2

Fraction missing: 0.000000
```

The schema here is inferred from the variable ranges observed in the data file. Note that the range is listed as "1" for many of the variables. This is because these variables only appear with the value "0" in this short data file.

Step 4: Circuit Learning

To learn models with higher treewidth, use the **aclearnstruct** program:

```
aclearnstruct -i msweb.data -s msweb.schema -o msweb-ac.ac -ob msweb-ac.xmod
```

As in **cl**, `-i` designates the data and `-s` the (optional) schema. **aclearnstruct** has two outputs: `-o` designates the output AC, and `-ob` designates the output BN. The BN and AC are guaranteed to represent the same distribution, but the AC is more convenient for inference.

aclearnstruct is sensitive to a number of parameters that control how it trades off the accuracy of the model with the size of the circuit. In experiments on a small set of domains, the following parameter settings worked fairly well:

```
-pe 100 -ps 0 -maxe 100000 -shrink
```

`-pe` and `-ps` specify per-edge and per-split penalties, respectively, in the score function. When the per-edge cost is higher, **aclearnstruct** will more strongly prefer structures with few edges. The per-split penalty is for early stopping, in order to avoid overfitting. With the **-psthresh** option,

aclearnstruct will output models for different values of **-ps** as learning progresses, without needing to rerun training.

The last two options specify the maximum number of edges (100,000, in this case) and tell **aclearnstruct** to keep running until this edge limit is met, reducing the per-edge cost as necessary. This way, we can start with a conservative edge cost (such as 100), select only the most promising simple structures, and make compromises later as our edge budget allows.

A final option is `-quick`, which relaxes the default, greedy heuristic to be only approximately greedy. In practice, it is often an order of magnitude faster with only slightly worse accuracy.

The `-parents` option allows you to control which variables are allowed to be parents of which other variables. See the manual for more details.

`aclearnstruct` can also be used to learn BNs that cannot be compactly represented as circuits, by using the `-noac` flag. When this flag is used, no AC is generated and the per-edge penalty is zero.

Step 5: Model Scoring

We can compute the average log likelihood per example using the **mscore** program:

```
mscore -m msweb-cl.xmod -i msweb.test
mscore -m msweb-cl.ac -i msweb.test
```

`-m` specifies the MN, BN, or AC to score, and `-i` specifies the test data. The filetype is inferred from the extension: `.mn` for MNs; `.xmod` and `.bif` for BNs; and `.ac` for ACs. Since we generated this AC from the BN, both scores should be identical, apart from minor rounding errors (which do show up in this case). To see the likelihood of every test case individually, enable verbose output with `-v`.

If we also score `msweb-ac.xmod`, we see that the extra expressiveness allowed by **aclearnstruct** has indeed translated to higher log likelihood on the test data (-9.911 vs. -10.004).

Step 6: AC Exact Inference

Now that we have an AC, we wish to use it to answer queries using **acquery**. The simplest query is computing all single-variable marginals:

```
acquery -c msweb-cl.ac -marg
```

`-marg` specifies that we want to compute the marginals. The output list of 294 marginal distributions is a bit overwhelming. For more convenient reading, you can redirect just the marginals to a file using the `-mo` flag. (Once `-mo` has been specified, `-marg` is implied and may be omitted.)

We can also use **acquery** to compute the probabilities of configurations of multiple variables, such as the probability that variables two through five all equal 0. To do this, we need to create a query file that defines every configuration we're interested in. The format is identical to Libra data files, except that we use "*" in place of an actual value for variables whose values are unspecified. `msweb.q` contains three example queries.

```
acquery -c msweb-cl.ac -q msweb.q
```

acquery outputs the log probability of each query as well as the average over all queries and its standard deviation. With the `-v` flag, **acquery** will print out query times in a second column.

An evidence file can be specified as well using `-ev`, using the same format as the query file:

```
acquery -c msweb-cl.ac -q msweb.q -ev msweb.ev -v
```

Evidence can also be used when computing marginals. If you specify both `-q` and `-marg`, then the probability of each query will be computed as the product of the marginals computed. This is typically less accurate, since it ignores correlations among the query variables.

To obtain the most likely variable state (possibly conditioned on evidence), use the `-mpe` flag:

```
acquery -c msweb-cl.ac -ev msweb.ev -mpe
```

This computes the most probable explanation (MPE) state for each evidence configuration. If the MPE state is not unique, **acquery** will select one arbitrarily. If you specify a query with `-q`, then **acquery** will print out the fraction of query variables that differ from the predicted MPE state:

```
acquery -c msweb-cl.ac -q msweb.q -ev msweb.ev -mpe
```

Step 7: Approximate Inference

Sometimes, we have a BN or MN (perhaps learned with some other toolkit) for which there is no efficient AC. We can still obtain answers from this network by running mean field, belief propagation, or Gibbs sampling.

For example, to run the same queries as above using mean field inference:[3]

```
mf -m msweb-cl.xmod
mf -m msweb-cl.xmod -q msweb.q
mf -m msweb-cl.xmod -q msweb.q -ev msweb.ev -v
```

Note that there is no `-marg` option, since mean field always approximates the distribution as a product of marginals. To trade off accuracy and speed in `mf`, use the `-thresh` and `-maxiter` arguments, which specify the convergence threshold and maximum number of iterations to run, respectively.

Belief propagation (**bp**) supports the same command line parameters as **mf**, e.g.:

```
bp -m msweb-cl.xmod -q msweb.q -ev msweb.ev -v
bp -m msweb-cl.xmod -q msweb.q -ev msweb.ev -sameev -v
```

The `-sameev` option in the second command runs BP only once with the first evidence configuration in `msweb.ev`, and then reuses the marginals for answering all queries in `msweb.q`. (This is also supported by `mf`.)

Gibbs sampling is done by **gibbs**, and follows similar conventions. Gibbs sampling is an anytime algorithm, which means you can specify how long it should run. The parameters controlling this are `-burnin` (the number of burn-in iterations), `-sampling` (the number of sampling iterations), and `-chains` (the number of simultaneous chains to run). For convenience, you can also set

[3] `mf` is also the name of the executable for the Metafont program distributed with TEX. Depending on your system setup, you may need to use the path when referring to **mf**, e.g. `../../bin/mf`.

these parameters using the `-speed` option. Valid arguments are `fast`, `medium`, `slow`, `slower`, and `slowest`, corresponding to a total of 1k/10k/100k/1M/10M sampling iterations, respectively.

Like **acquery**, **gibbs** can answer queries using either the full joint distribution (the default) or just the marginals (with `-marg`). The latter option is helpful for estimating the probabilities of rare events that may never show up in the samples.

You can also print the raw samples to a file using `-so`. To obtain repeatable experiments, use `-seed` to specify a random seed.

Examples:

```
gibbs -m msweb.xmod -mo msweb-gibbs.marg
gibbs -m msweb.xmod -q msweb.q -ev msweb.ev -v
gibbs -m msweb.xmod -q msweb.q -ev msweb.ev -speed medium -v
gibbs -m msweb.xmod -q msweb.q -ev msweb.ev -marg -sameev
```

Approximate MPE (most probable explanation) inference is done by **maxprod**, an implementation of the max-product algorithm. Max-product is identical to belief propagation, except that it replaces summation with maximization to find the (approximate) most likely variable configuration for the non-evidence variables. Like **bp**, it is exact on tree-structured networks. Without a query file, **maxprod** prints the MPE states given the evidence. Given a query file, **maxprod** prints the fraction of non-evidence variables that are different from the estimated MPE state.

Examples:

```
maxprod -m msweb-cl.xmod -ev msweb.ev -mo msweb-cl.mpe
maxprod -m msweb.xmod -q msweb.q -ev msweb.ev -v
```

Step 8: Markov Networks

As of version 0.3.0, Libra now supports inference and scoring of Markov networks. One way to create a Markov network is to convert a Bayesian network using `mconvert`:

```
mconvert -m msweb.xmod -o msweb.mn
```

The resulting MN can be used for approximate inference with `bp`, `gibbs`, or `mf`:

```
bp -m msweb.mn
mf -m msweb.mn -ev msweb.ev
gibbs -m msweb.mn -q msweb.q -ev msweb.ev -v
```

You can also compute the pseudo-log-likelihood (PLL) of test data using `mscore`:

```
mscore -m msweb.mn -i msweb.test
```

In general, log-likelihood is intractable to compute in a Markov network. PLL is the sum of the conditional log-probabilities of each variable given its Markov blanket. This is much faster to compute, but not directly comparable to likelihood. To force `mscore` to use PLL when scoring a BN, use the `-pll` flag:

```
mscore -m msweb.xmod -i msweb.test -pll
```

If you do not use `-pll`, `mscore` will print the unnormalized log-likelihood, which differs from the true log-likelihood by a constant called the log partition function.

`mconvert` can also be used to convert between XMOD and BIF BN formats, e.g.:

```
mconvert -m msweb-cl.xmod -o msweb-cl.bif
```

If an XMOD file is the source, it must not contain trees, since these are not supported by BIF. (`mconvert` can also be used to simplify ACs and MNs by conditioning them on evidence, as described in the manual.)

Dependency Networks

A dependency network (DN) specifies a conditional probability distribution for each variable given its parents. However, unlike a Bayesian network, the graph of parent-child relationships may contain cycles. With Libra, you can learn a dependency network with tree-structured conditional probability distributions using `dnlearn`:

```
dnlearn -i msweb.data -s msweb.schema -o msweb-dn.xmod -prior 1
```

The resulting dependency network will be saved as a XMOD file. The XMOD file itself does not distinguish between BNs and DNs.

DNs can be scored using `mscore`:

```
mscore -m msweb-dn.xmod -i msweb.test -depnet
```

The score of a DN is a pseudo-log-likelihood, which is not directly comparable to log-likelihood. Note that using the flag `-pll` and omitting the flag `-depnet` will cause `mscore` to interpret the DN as a BN and generate incorrect results.

Two approximate inference algorithms are available for DNs: Gibbs sampling and mean field. To use them with a DN instead of a BN, just add the `-depnet` flag to the command line:

```
gibbs -m msweb-dn.xmod -mo msweb-dn-gibbs.marg -depnet
mf -m msweb-dn.xmod -ev msweb.ev -q msweb.q -depnet
```

Miscellaneous

This tutorial covers most of Libra, but not all. The `bnsample` program can be used for generating iid samples from a Bayesian network; its use is fairly straightforward. The `acopt` program is for adjusting the weights of an arithmetic circuit, either to maximize the log likelihood of training data (weight learning), or to minimize the reverse Kullback-Leibler divergence to a Bayesian network (variational inference). Details are beyond the scope of the present tutorial.

Acknowledgments

Thanks to Andrey Kolobov and Marc Sumner for beta testing this tutorial.