Bayes Net Toolbox for Matlab (BNT)

Written by Kevin Murphy, 1997--2002. Last updated: 19 October 2007.

Installation

To download the BNT: **https://code.google.com/p/bnt/**

To install, just unzip FullBNT.zip, start Matlab, and then add BNT to your path. You can go to File->Set_Path, and then click on "Add Folder with Subfolders" and select the bnt folder. Or you can cd to the directory containing BNT and execute the following

```
>> cd C:\kmurphy\FullBNT\FullBNT-1.0.4  % modify as needed
>> addpath(genpathKPM(pwd))
```

The genpathKPM function is like the builtin genpath function, but it does not add directories called 'Old' to the path, thus preventing old versions of functions accidently shadowing new ones. The warnings occur because Matlab 7 added functions with the same names as my functions. The BNT versions will shadow the built-in ones, but this should be harmless. (Note: the functions installC_BNT etc. are not needed anymore: all C code has either been removed or is unnecessary.)

Now test your installation:

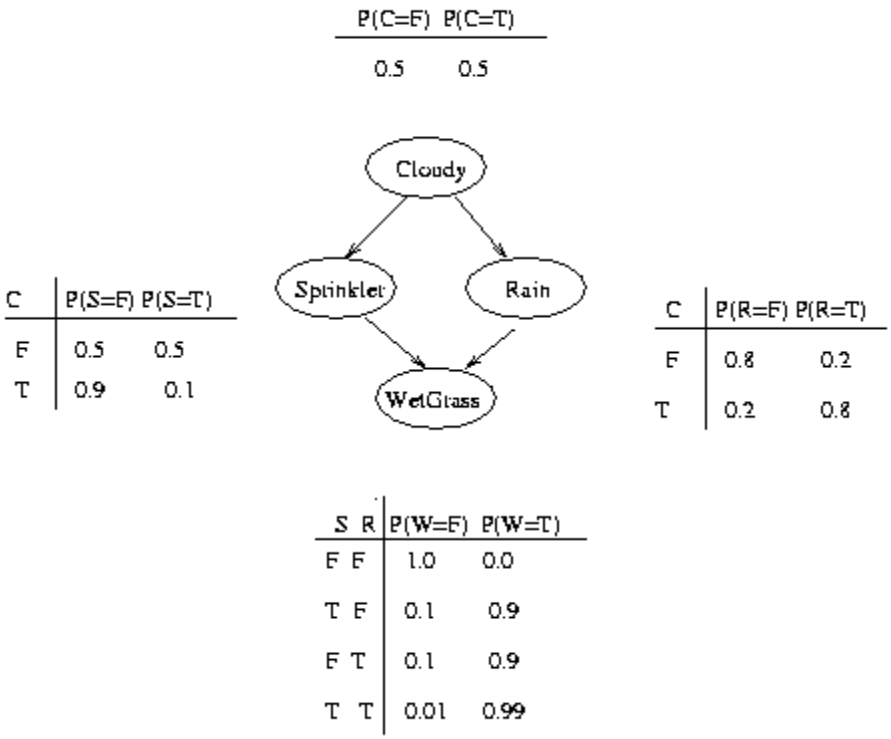
```
>> test_BNT
```

Creating your first Bayes net

To define a Bayes net, you must specify the graph structure and then the parameters. c

Graph structure

Consider the following network.



To specify this directed acyclic graph (dag), we create an adjacency matrix:

```
N = 4;
dag = zeros(N,N);
C = 1; S = 2; R = 3; W = 4;
dag(C,[R S]) = 1;
dag(R,W) = 1;
dag(S,W)=1;
```

We have numbered the nodes as follows: Cloudy = 1, Sprinkler = 2, Rain = 3, WetGrass = 4. **The nodes must always be numbered in topological order, i.e., ancestors before descendants.** .

Creating the Bayes net shell

In addition to specifying the graph structure, we must specify the size and type of each node. If a node is discrete, its size is the number of possible values each node can take on; if a node is continuous, it can be a vector, and its size is the length of this vector. In this case, we will assume all nodes are discrete and binary.

```
discrete_nodes = 1:N;
node_sizes = 2*ones(1,N);
```

If the nodes were not binary, you could type e.g.,

```
node_sizes = [4 2 3 5];
```

meaning that Cloudy has 4 possible values, Sprinkler has 2 possible values, etc. Note that these are cardinal values, not ordinal, i.e., they are not ordered in any way, like 'low', 'medium', 'high'.

We are now ready to make the Bayes net:

```
bnet = mk_bnet(dag, node_sizes, 'discrete', discrete_nodes);
```

By default, all nodes are assumed to be discrete, so we can also just write

```
bnet = mk_bnet(dag, node_sizes);
```

You may also specify which nodes will be observed. If you don't know, or if this not fixed in advance, just use the empty list (the default).

```
onodes = [];
```

```
bnet = mk_bnet(dag, node_sizes, 'discrete', discrete_nodes, 'observed',
onodes);.
```
It is possible to associate names with nodes, as follows:
```
bnet = mk_bnet(dag, node_sizes, 'names', {'cloudy','S','R','W'},
'discrete', 1:4);
```
You can then refer to a node by its name:
```
C = bnet.names('cloudy'); % bnet.names is an associative array
bnet.CPD{C} = tabular_CPD(bnet, C, [0.5 0.5]);
```
This feature uses my own associative array class.

Parameters

A model consists of the graph structure and the parameters. The parameters are represented by CPD objects (CPD = Conditional Probability Distribution), which define the probability distribution of a node given its parents. (We will use the terms "node" and "random variable" interchangeably.) The simplest kind of CPD is a table (multi-dimensional array), which is suitable when all the nodes are discrete-valued. Note that the discrete values are not assumed to be ordered in any way; that is, they represent categorical quantities, like male and female, rather than ordinal quantities, like low, medium and high.

Tabular CPDs, also called CPTs (conditional probability tables), are stored as multidimensional arrays, where the dimensions are arranged in the same order as the nodes, e.g., the CPT for node 4 (WetGrass) is indexed by Sprinkler (2), Rain (3) and then WetGrass (4) itself. Hence the child is always the last dimension. If a node has no parents, its CPT is a column vector representing its prior. Note that in Matlab (unlike C), arrays are indexed from 1, and are layed out in memory such that the first index toggles fastest, e.g., the CPT for node 4 (WetGrass) is as follows

S	R	W	prob.
1	1	1	1.0
2	1	1	0.1
1	2	1	0.1
2	2	1	0.01
1	1	2	0.0
2	1	2	0.9
1	2	2	0.9
2	2	2	0.99

where we have used the convention that false==1, true==2. We can create this CPT in Matlab as follows
```
CPT = zeros(2,2,2);
CPT(1,1,1) = 1.0;
CPT(2,1,1) = 0.1;
...
```
Here is an easier way:
```
CPT = reshape([1 0.1 0.1 0.01 0 0.9 0.9 0.99], [2 2 2]);
```
In fact, we don't need to reshape the array, since the CPD constructor will do that for us. So we can just write
```
bnet.CPD{W} = tabular_CPD(bnet, W, 'CPT', [1 0.1 0.1 0.01 0 0.9 0.9 0.99]);
```
The other nodes are created similarly (using the old syntax for optional parameters)
```
bnet.CPD{C} = tabular_CPD(bnet, C, [0.5 0.5]);
bnet.CPD{R} = tabular_CPD(bnet, R, [0.8 0.2 0.2 0.8]);
bnet.CPD{S} = tabular_CPD(bnet, S, [0.5 0.9 0.5 0.1]);
bnet.CPD{W} = tabular_CPD(bnet, W, [1 0.1 0.1 0.01 0 0.9 0.9 0.99]);
```

Random Parameters

If we do not specify the CPT, random parameters will be created, i.e., each "row" of the CPT will be drawn from the uniform distribution. To ensure repeatable results, use

```
rand('state', seed);
randn('state', seed);
```

To control the degree of randomness (entropy), you can sample each row of the CPT from a Dirichlet(p,p,...) distribution. If p << 1, this encourages "deterministic" CPTs (one entry near 1, the rest near 0). If p = 1, each entry is drawn from U[0,1]. If p >> 1, the entries will all be near 1/k, where k is the arity of this node, i.e., each row will be nearly uniform. You can do this as follows, assuming this node is number i, and ns is the node_sizes.

```
k = ns(i);
ps = parents(dag, i);
psz = prod(ns(ps));
CPT = sample_dirichlet(p*ones(1,k), psz);
bnet.CPD{i} = tabular_CPD(bnet, i, 'CPT', CPT);
```

Inference

Having created the BN, we can now use it for inference. There are many different algorithms for doing inference in Bayes nets, that make different tradeoffs between speed, complexity, generality, and accuracy. BNT therefore offers a variety of different inference "engines".

- For polytrees, , we will use the pearl engine. This can be created as follow:

```
engine = pearl_inf_engine(bnet);
```

- For multy-connected graphs, we will use the junction tree engine, which is the mother of all exact inference algorithms. This can be created as follows.

```
engine = jtree_inf_engine(bnet);
```

Computing marginal distributions

Suppose we want to compute the probability that the sprinker was on given that the grass is wet. The evidence consists of the fact that W=2. All the other nodes are hidden (unobserved). We can specify this as follows.

```
evidence = cell(1,N);
evidence{W} = 2;
```

We use a 1D cell array instead of a vector to cope with the fact that nodes can be vectors of different lengths. In addition, the value [] can be used to denote 'no evidence', instead of having to specify the observation pattern as a separate argument.
We are now ready to add the evidence to the engine.

```
[engine, loglik] = enter_evidence(engine, evidence);
```

In the case of the jtree engine, enter_evidence implements a two-pass message-passing scheme. The first return argument contains the modified engine, which incorporates the evidence. The second return argument contains the log-likelihood of the evidence. (Not all engines are capable of computing the log-likelihood.)
Finally, we can compute p=P(S=2|W=2) as follows.

```
marg = marginal_nodes(engine, S);
marg.T
ans =
      0.57024
      0.42976
p = marg.T(2);
```

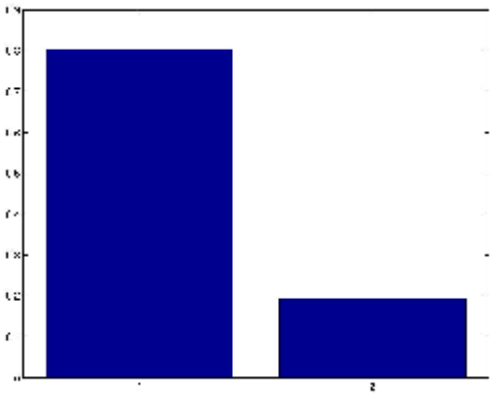
We see that p = 0.4298.
Now let us add the evidence that it was raining, and see what difference it makes.

```
evidence{R} = 2;
[engine, loglik] = enter_evidence(engine, evidence);
marg = marginal_nodes(engine, S);
p = marg.T(2);
```

We find that p = P(S=2|W=2,R=2) = 0.1945, which is lower than before, because the rain can ``explain away'' the fact that the grass is wet.
You can plot a marginal distribution over a discrete variable as a barchart using the built 'bar' function:

```
bar(marg.T)
```

This is what it looks like



Observed nodes

What happens if we ask for the marginal on an observed node, e.g. P(W|W=2)? An observed discrete node effectively only has 1 value (the observed one) --- all other values would result in 0 probability. For efficiency, BNT treats observed (discrete) nodes as if they were set to 1, as we see below:

```
evidence = cell(1,N);
evidence{W} = 2;
engine = enter_evidence(engine, evidence);
m = marginal_nodes(engine, W);
m.T
ans =
     1
```

This can get a little confusing, since we assigned W=2. So we can ask BNT to add the evidence back in by passing in an optional argument:

```
m = marginal_nodes(engine, W, 1);
m.T
ans =
     0
     1
```

This shows that P(W=1|W=2) = 0 and P(W=2|W=2) = 1.

Computing joint distributions

We can compute the joint probability on a set of nodes as in the following example.

```
evidence = cell(1,N);
[engine, ll] = enter_evidence(engine, evidence);
m = marginal_nodes(engine, [S R W]);
```

m is a structure. The 'T' field is a multi-dimensional array (in this case, 3-dimensional) that contains the joint probability distribution on the specified nodes.

```
>> m.T
ans(:,:,1) =
```

```
    0.2900    0.0410
    0.0210    0.0009
ans(:,:,2) =
         0    0.3690
    0.1890    0.0891
```

We see that P(S=1,R=1,W=2) = 0, since it is impossible for the grass to be wet if both the rain and sprinkler are off.

Let us now add some evidence to R.

```
evidence{R} = 2;
[engine, ll] = enter_evidence(engine, evidence);
m = marginal_nodes(engine, [S R W])
m =
    domain: [2 3 4]
         T: [2x1x2 double]
>> m.T
m.T
ans(:,:,1) =
    0.0820
    0.0018
ans(:,:,2) =
    0.7380
    0.1782
```

The joint T(i,j,k) = P(S=i,R=j,W=k|evidence) should have T(i,1,k) = 0 for all i,k, since R=1 is incompatible with the evidence that R=2. Instead of creating large tables with many 0s, BNT sets the effective size of observed (discrete) nodes to 1, as explained above. This is why m.T has size 2x1x2. To get a 2x2x2 table, type

```
m = marginal_nodes(engine, [S R W], 1)
m =
    domain: [2 3 4]
         T: [2x2x2 double]
>> m.T
m.T
ans(:,:,1) =
             0        0.082
             0       0.0018
ans(:,:,2) =
             0        0.738
             0       0.1782
```

Note: It is not always possible to compute the joint on arbitrary sets of nodes: it depends on which inference engine you use..

Soft/virtual evidence

Sometimes a node is not observed, but we have some distribution over its possible values; this is often called "soft" or "virtual" evidence. One can use this as follows

```
[engine, loglik] = enter_evidence(engine, evidence, 'soft', soft_evidence);
```

where soft_evidence{i} is either [] (if node i has no soft evidence) or is a vector representing the probability distribution over i's possible values. For example, if we don't know i's exact value, but we know its likelihood ratio is 60/40, we can write evidence{i} = [] and soft_evidence{i} = [0.6 0.4].

Currently only jtree_inf_engine supports this option. It assumes that all hidden nodes, and all nodes for which we have soft evidence, are discrete.