15780: GRADUATE AI (SPRING 2017)

Homework 3: Probabilistic Modeling and Game Theory

Release: March 29, 2017,
Due: April 12, 2017

1 Maximum Likelihood Estimation [30 points]

1.1 MLE for Uniform Distribution with Two Parameters [10 points]

Given a collection of observed (independent) data points $X = \{x^{(1)}, ...x^{(m)}\}$ from a uniform distribution $U(\alpha, \beta)$, where

$$p(x) = \begin{cases} \frac{1}{\beta - \alpha}, & \text{if } \alpha \leq x \leq \beta \\ 0, & \text{otherwise,} \end{cases}$$

derive the maximum likelihood estimators of α and β, which maximize the probability of observing X.

1.2 MLE for Uniform Distribution with One Parameter [10 points]

Given a collection of observed (independent) data points $X = \{x^{(1)}, ...x^{(m)}\}$ from a uniform distribution $U(-2\alpha, \alpha)$, derive the maximum likelihood estimators of α, which maximizes the probability of observing X.

1.3 MLE for Uniform Distribution with Prior [10 points]

Given a collection of observed (independent) data points $X = \{x^{(1)}, ...x^{(m)}\}$ from a uniform distribution $U(0, e^{\alpha})$ where α follows a prior distribution

$$p(\alpha) \propto e^{-\alpha^2},$$

derive the estimator of α that maximizes the posteriori probability $p(\alpha|X)$. **(Hint: use** $p(\alpha|X) \propto p(X|\alpha)p(\alpha)$**).**

2 Equilibria [30 points]

2.1 Iterated Elimination of Strictly Dominated Strategies [15 points]

One method of simplifying the search for Nash equilibria is through the iterated elimination of strictly dominated strategies. We say that a player's pure strategy s_i' is strictly dominated by another pure s_i if $\forall s_{-i} \in S_{-i},\ u_i(s_i', s_{-i}) < u_i(s_i, s_{-i})$. In other words, if there exist two pure strategies, s_1 and s_2, for player i such that no matter the (possibly mixed) strategies of the other players it is always in player i's interest to play s_1 over s_2, then s_1 dominates s_2.

The iterated elimination of strictly dominated strategies proceeds by eliminating one strictly dominated strategy per round. It then iteratively continues until there are no more dominated strategies to eliminate. For example, iterated elimination of strictly dominated strategies on the following game proceeds as follows.

	North	East	South	West
Top	2,3	1,-1	4,0	3,-3
Middle	7,2	-2,0	5,2	6,7
Bottom	8,2	0,1	6,-1	4,0

- Column eliminates East, as playing North is strictly better.
- Row eliminates Top, as playing either Middle or Bottom is strictly better now that Column has eliminated East.

- Column eliminates South, as playing West is strictly better now.
- No more strategies can be eliminated; this leaves Row: [Middle, Bottom] and Column: [North, West] as the surviving strategies.

Prove that if iterated elimination of dominated strategies eliminates all but one of the strategies of each player, then there is a unique Nash equilibrium in the game. You may find Nash's Theorem useful here.

2.2 Correlated Equilibria [15 points]

Let $p_1, \ldots, p_n$ be probability distributions representing n correlated equilibria in the same 2-player game. Prove that any convex combination of $p_1, \ldots, p_n$ is also a correlated equilibrium.

3 Metropolis-Hastings Implementation [10 points]

This portion is not graded on autolab.

We will implement the Metropolis-Hastings algorithms to generate samples from the unnormalized distribution

$$\tilde{p}(x) = e^{-x^2} + 1.3e^{-(x-2)^2}$$

with a Gaussian sampling function with the centered at the previous point with some standard deviation σ.

1. (5 points) Run your implementation for every combination of $x_0 \in \{-2, 2\}$ and $\sigma \in \{0.1, 1.0, 10\}$. Include plots (1. samples over time and 2. a histogram) for all of these in your writeup.
2. (3 points) How and why does σ impact the samples? For all three values, you should state what happens and why it happens.
3. (2 points) What is the impact of x_0 on the samples?

3.1 Getting started

Download the handout source code and edit the `mh.py` source file. This file contains a `main` method that will run your implementation and generate sampling plots and histograms. In this file, finish the Metropolis-Hastings implementation in the `mh` function from the description on page 22 of Slides 16 (Probabilistic Modeling III: MCMC).

4 Gibbs Sampling [30 points]

For this problem you will implement the Gibbs sampling algorithm to sample values for the variables in a Bayesian network given a set of evidence variables. In particular, you will implement the following function, in the included file `gibbs.py`:

```
def gibbs(vars, cpts, evidence, n_steps=1000):
```

This function takes as input:

- a dictionary mapping each variable in the Bayes net to the set of values it can take on,
- a list of conditional probability tables (CPTs) for each variable, which are instances of the `CPT` class (more details on this shortly),
- a dictionary mapping each variable we observe (evidence variable) to the observed value, and
- (optionally) the number of steps we want to run Gibbs sampling for—at each step, we sample a new value for a single variable that is not in the evidence set.

The function should return a dictionary mapping every variable to its sampled (including the evidence variables).

The basis for the Bayes net representation we will use is the `CPT` class included in the `gibbs.py` file. You won't have to edit this class at all, or even necessarily understand all the code in this class, but you will need to understand how to use the class. This class describes a conditional probability table for a particular variable in the Bayes net. Each `CPT` object contains 1) a Python dictionary that maps each variable in the CPT to a list of its possible values, and 2) a dictionary mapping each possible instantiation of the variables to the conditional probability of the variable of interest given the remaining variables (i.e., given all of its parents). Let's look at a simple example: if we initialize `CPT` by the following:

```
c = CPT({"x1":[0,1], "x2":["TRUE","FALSE"], "x3":["RED","BLUE","GREEN"]})
```

this will create a factor of three variables, "x1", "x2", and "x3," where the first variable can take on values 0 or 1, the second can be "TRUE" or "FALSE", and the third can take on values "RED", "GREEN", or "BLUE" (note that in this problem, unlike the lecture notes, variables can take on more than two values, but this really introduces no added complexity). Note that there is no requirement that the list of possible values for each factor be numbers: indeed, for the real-world Bayes net example, these will typically be lists of strings, but this should not change your code at all. If you want to access this list of variables at any point, use the class member `c.variables`.

To get the probability of a variable v given all its parents and children, you can "call" the CPT object by passing in the variables as keyword arguments. For example, you can write

```
c(v=0, par1=1, par2=1)
```

Note that you must provide arguments for all the variables in the CPT for v; otherwise you will get an error. However, you may pass in additional variables; the function simply ignores them. For example,

```
#same as c(v=0, par1=1, par2=1)
c(v=0, par1=1, par2=1, unrelated_v=1)
```

This is particularly useful, because you can do something like the following

```
all_vars = {"v1":0, "v2":0, "v3":1, "v4":0, "v5":1, "v6":0, "v7":1, "v8":1}
c(**all_vars)
```

Thus, even if only a small subset of the variables are in a particular CPT, the syntax above allows us to easily find the probability of a particular vertex given its parents and children simply by passing in the instantiation of all variables into its CPT.

We will test your code on the ALARM Bayes net[1]. This network has been provided to you in the file `alarm.xmlbif` and code for parsing it has been provided in `gibbs.py`. We will test your code on several test cases where we examine if your code correctly samples a particular variable given a set of parent variables. To see if it correctly samples a variable, we will run your function several times with the same inputs and compare the number of times you sample each value for that variable to the expected conditional distribution of the variable given the evidence variables (as generated by our solution Gibbs sampling code). Three example test cases have been provided in `gibbs.py`. Your code should run efficiently; each test case should run in under two minutes. Note: If you find that you only fail one or two test cases on Autolab, try uploading and running your code again, just in case it failed the tests by chance (but is actually correct).

Hint: You may want to start by thinking about how to **easily** express the probability of a particular variable taking on a value given the value of all other variables in terms of the conditional probabilities of the form in the `CPT` class. This could help you avoid writing complicated code and make your code more efficient.

5 Submitting to Autolab

Create a tar file containing your writeup and the completed `gibbs.py` modules for the programming problems. Make sure that your tar has these files at the root and not in a subdirectory. Use the following commands from a directory with your files to create a `handin.tgz` file for submission.

```
$ ls
gibbs.py  writeup.pdf  [...]
$ tar cvzf handin.tgz writeup.pdf gibbs.py
a writeup.pdf
a gibbs.py
$ ls
handin.tgz  gibbs.py  writeup.pdf  [...]
```

[1] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine, pages 247-256. Springer-Verlag, 1989.