Programming Assignment 4 – Basic Probability, Computing and Statistics 2015

Fall 2015, Master of Logic, University of Amsterdam

Submission deadline: Monday, September 28st, 9 a.m.

Note: if the assignment is unclear to you or if you get stuck, do not hesitate to contact Philip.

1 Logprobs and sampling

When implementing probabilistic computations, one often faces several practical problems. One of those problems is that in realistic applications, the probabilities involved tend to get really small. Since a computer has limited memory, it can only represent floating point numbers up to certain degree of precision (i.e. up to a specific number of decimal digits). If a probability becomes so small that it cannot be captured by a floating point number within the precision range of the computer, that probability will simply be turned into 0 (this problem is known as underflow). This is undesirable of course. If the zeroed probability is part of a bigger multiplication (e.g. one based on the chain rule), the whole computation will yield 0 as a result.

To prevent underflow, one often uses the logarithm of probabilities (a.k.a. logprobs). This also has an additional advantage: multiplications in real space are additions in log space. Thus, the chain rule can be rewritten as a sum. Since addition on a computer is much faster than multiplication, this results in a speed-up of the computation. Incidentally, mathematicians often use logprobs because additions are easier to handle in proofs than multiplications.

Let us recall what a logarithm (with base b) is.

$$log_b(x) = y \text{ such that } b^y = x \tag{1}$$

Another problem of probabilistic computations is sampling from a distribution. Many programming languages come with build-in functions that allow you to sample from a **uniform** distribution. However, in most interesting cases, you would like to sample from distributions that are non-uniform. This can be achieved using inverse transform sampling. The idea of inverse

transform sampling is simple. To sample values of a random variable X, do the following:

1. Sample a threshold value t uniformly in [0,1] (crucially, this is where the randomness of the sample comes from).

2. Evaluate the cdf of your distribution at each possible value of $supp(X)$ in an arbitrary but a priori fixed order (this order has to be the same for each sample).

3. As soon as the cdf is greater or equal to t return the value on which you just evaluated the cdf.

Care has to be taken with this process. The build-in function of many programming languages only sample in [0,1). In that case, the cdf needs to strictly greater than t before you return a value (try to justify why this is so).

2 The task

Your task is to implement utility functions that do computations with log-probs. For this purpose, we supply you with unit tests. Your implementation is correct if it passes all tests. For details on how to the functions should work, see their docstring. **PHILIP: be more specific here.**

Furthermore you will implement the binomial distribution as a class. Given its parameters, your class should be able to return the probability and logprob of any bit-sequence of size n. Furthermore, it should implement a method `sample()` that returns a random bit-sequence of size n.