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

1.1 Logprobs

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.

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

We can rewrite derive addition of logprobs as follows: let p and q be

probabilities such that $p \geq q$ and define $a = log(p)$ and $b = log(q)$. Then

(2)
$$log(p+q) = log\left(e^{log(p)} + e^{log(q)}\right) = log\left(e^{a} + e^{b}\right) = log\left(e^{b}(e^{a-b}+1)\right)$$
$$= b + log\left(e^{a-b}+1\right) = log(q) + log(\left(e^{log(p)-log(q)}+1\right) \ .$$

You can derive yourselves what this would look like for subtraction.

1.2 Inverse transform sampling

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 $supp(X)$.
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 and a class that implements the binomial distribution. For this purpose, we supply you with unit tests. Your implementation is correct if it passes all tests. For details on what each function should do, see their docstring. Notice also that some functions have comments outside their docstrings. These are programming instructions that YOU HAVE to follow. If you do not adhere to these instructions, points will be deducted.

Let us explain a bit more about the binomial distribution class. Given its parameters, your class should be able to return the probability of any sequence of size n. Furthermore, it should implement a method `sample()` that returns a sequences of size n. For further functionality, check the docstrings.

Notice that both the logarithms utility module and the binomial class may be extremely helpful in your later programming career. You should keep them after this course has finished. Since they have been unit-tested, you can also be fairly sure that they work correctly.

The log-probs function can be used over and over again. The binomial itself may not be of direct use but it gives you a good intuition for how you should implement probability distributions.

The unit test can be downloaded here. The skeletons of the modules that you need to implement are here (`logarithms.py` and `distributions.py`). In order for the unit tests to work correctly, it is important that you maintain the file hierarchy as you find it on Github. To this end, it may actually be easier to just clone the entire code folder by typing on one line

```
git clone
https://github.com/BasicProbability/PythonCode_Fall2015.git
```

and then work in there. This can be done by importing the cloned folder into your eclipse workspace. Obviously, cloning will only work if you have git installed.

3 Grading

This time you will be very restricted in that we only allow you to use the imports and methods that we have pre-defined for you. You may not define your own functions, methods or classes! Your code is correct if it passes all unit tests.

0.5 points for each passed ValueError test (3 points in total).

1 point if test_log_add passes.

1 point if test_log_difference passes.

1 point if test_compute_probability passes.

1 point if test_sample_with_k_successes passes.

1 point if test_sample passes.

1 point if test_sample_list passes.

1 point if all the programming instructions (in-line comments) have been followed. Give 0 points here if one or more of them have been violated.

-1 point for each additional import, class, method or function introduced by the student.