Hakaru Language: Standard Library Implementation and Language Validation Testing

Justin Staples, Mahmoud Khattab, Nevin Mahilal and Aryan Sohrabi, supervised by Dr. Christopher Anand & Dr. Jacques Carette

{staplejw, khattm, mahilank, sohraa3, anandc, carette}@mcmaster.ca

Department of Computing and Software, McMaster University

Introduction

Applications

Real world phenomena often behave according to a probability distribution. By letting us sample from distributions, Hakaru enables us to model real world phenomena that include uncertainty. Hakaru can then perform transformations on these models to turn them into conditional probabilities that let us make inferences based on a prior. This for example is useful in machine learning applications where the goal is often to predict or infer information about the model based on the past outcomes.

Mathematical Background

The mathematical functions that we use to describe these probabilities are called probability distributions and the quantity that denotes the outcome of the experiment is called a random variable. A random variable can take on continuous or discrete values depending on the application. Continuous random variables are formally defined by a probability density function (PDF), which maps the outcomes of the random variable to real numbers that represent a probability. Similarly, a discrete random variable is described by a probability mass function (PMF). As an example, let us consider one of the most ubiquitous and naturally occurring distributions, the normal distribution. We can say that a random variable, X, is normally distributed with the following notation.

$$\boxed{X \sim Normal(\mu, \sigma^2)}$$

Here, we can see that the normal distribution is parametrized by μ, the population mean, and σ, the standard deviation. A plot the PDF is shown below. Remember that the PDF describes the liklihood that a value sampled from this distribution will equal x.

$$\boxed{f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}}$$

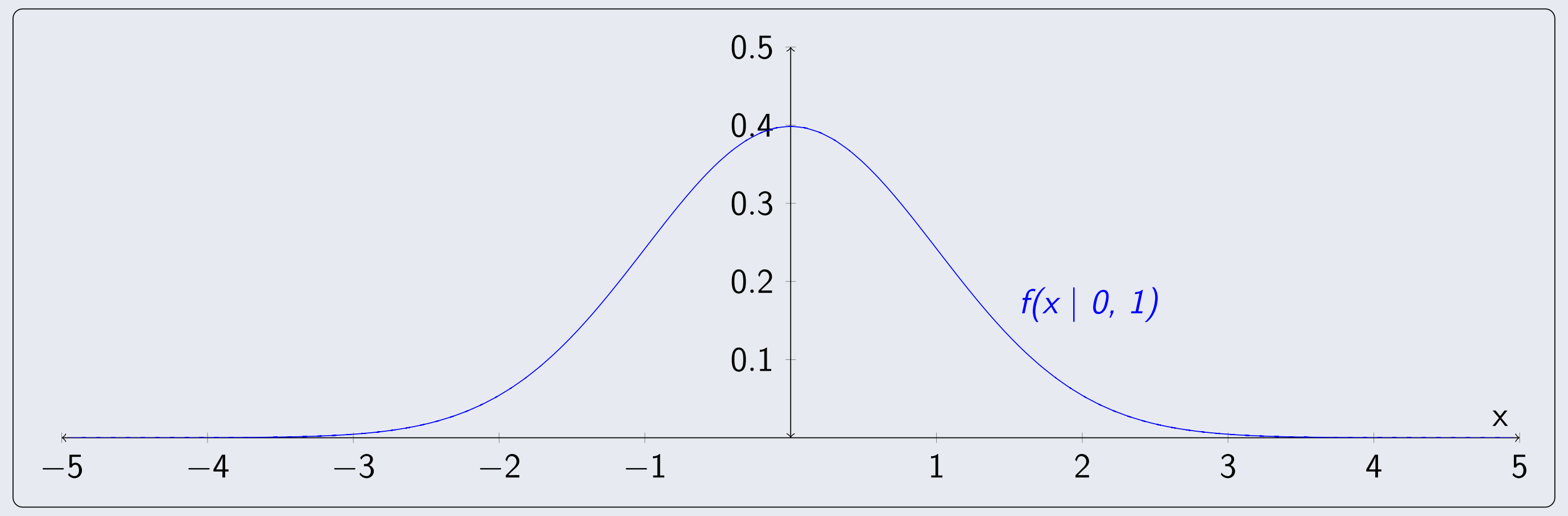

Figure 1: PDF of the normal distribution, with $\mu = 0$ and $\sigma = 1$

Language guide

Hakaru is an experimental **probabilistic programming language**, which aims to simplify the way users implement statistical distributions.

- Niche application ⇒ small language with limited features.
- Running a hakaru program generates a stream of random numbers distributed according the statistical distribution modeled.
- Hakaru code can be compiled to C and Haskell so that models can be exported for use in larger applications.
- Terminology: implementation of statistical distribution == probabilistic model == model.

'hk-maple' is a provided inference algorithm which takes a hakaru program file as an argument and uses Maple to perform algebraic transformations. If the Simplify mode of 'hk-maple' is used (the default mode), it returns an equivalent hakaru program with greater sampling efficiency. In applications requiring billions of samples (e.g. machine learning) this has the potential to save significant processing time!

Key Concepts

This section is dedicated to explaining some key ideas and terminology for our project.

- *Hakaru program == implementation of statistical distribution == probabalistic model == model.*
- *Values pulled from a distribution are called measures. Measures == samples.*

- '∼' vs '<∼':
  - $X \sim Normal(0, 1)$ means that the random variable, X, is distributed according to the standard normal distribution (statistics literature).
  - `x <~ normal(0, 1)` means to pull a random variable from the standard normal distribution and bind it to the variable, `x` (Hakaru code).
- *PDF: Probability Density Function (continuous random variables).*
- *PMF: Probability Mass Function (discrete random variables).*
- *UDR chart: Univariate Distribution Relationship chart (describes relationships and transformations between different distributions).*

Motivation

OBJECTIVES: Increase language accessibility: We have added functions that produce samples from various distributions to the standard library. We created a syntax-highlighting package for Sublime Text to help with the readability of the code. Test language validity: We used Hakaru?s testing procedure to confirm the validity of the distribution of our samples.

Standard Library Development

In order to make Hakaru more accessible, the standard library should help users create probabilistic models using statistical distributions that are commonly used by statisticians. To this this end, we have implemented functions that sample from the distributions described in the UDR.

Sampling Distributions using Transformations

One way to sample from a distribution is to apply algebraic transformations on the output of a sample generating function that samples from another distribution. In other words, if X is distributed according to a PDF/PMF, then f(X) is distributed according to a new distribution with a new PDF/PMF. These transformations are well outlined and documented in the UDR chart, and a portion of the chart is shown in the figure2.

For our development we tried to implement distributions as transformations on pre-existing models whenever possible. Hakaru has several primitive distributions we can build off of. In the case of multiple possible implementations, we took the shortest possible path from a primitive distribution on the UDR.

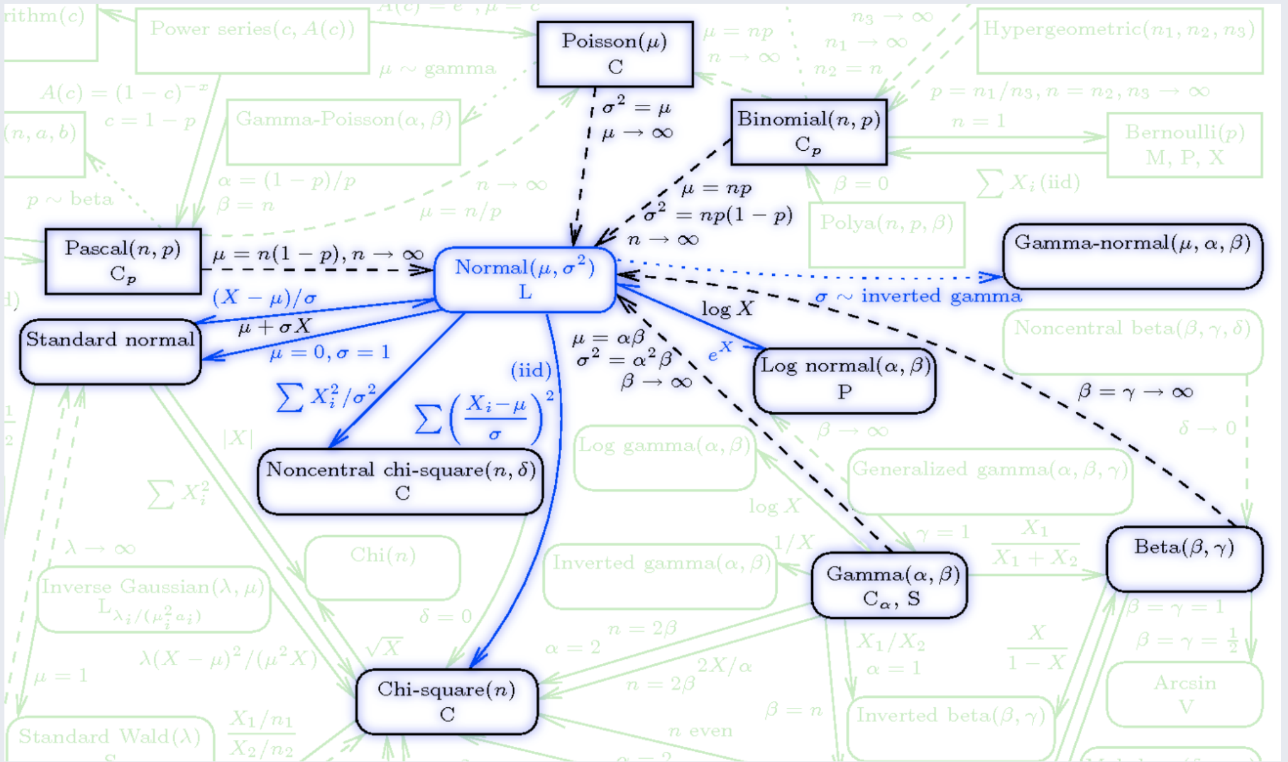

Figure 2: A snapshot of the UDR shows how the normal distribution can be transformed into a multitude of other distributions.

Hakaru functions representing probabilistic models (i.e. functions with a `measure(<type>)` return type) cannot be used as operands in normal mathematical operations. This is to say we cannot transform models directly (i.e. we cant do algebraic transformation on the PDF). The only operator we can use on a model is bind (<∼), to pull a sample from it. However, we are able to transform samples however we like. Therefore, we are interested in implementing transformations of the following form.

$$\boxed{R(p, q) \Rightarrow X \sim A(p) \Rightarrow f(X) \sim B(q)}$$

In the equation above, p is a set of values parameterizing the distribution A, q is a set of values parameterizing the distribution B, *R(p, q)* is a set of relationships between p and q that must be satisfied, and f is a function that applies a transformation to a sample. We can expand this definition to include transformations defined in terms of an aggregation of multiple independent samples. For example, the standard chi-square distribution is defined as the sum of the squares of n standard normal random variables.

Hakaru also lends itself very well to Bayesian transformations where we want to pull a sample from a distribution which is parameterized with a value sampled from another distribution. These transformations take the following form. Here, we are saying that if X is parameterized by p, then the distribution C, which is parameterized by p and q, is equal to another distribution, Y, which is parameterized by q and X.

$$\boxed{X \sim A(p) \Rightarrow C(p, q) = Y \sim B(q, X)}$$

Some distributions on the UDR are unreachable from Hakaru's primitive distributions using transformations like the 2 described above. In these cases, we have to implement the model in terms of the PMF for a discrete distribution or in terms of the PDF for a continuous distributions. We now discuss this process in the following examples.

Sampling Distributions from PMF/PDF

Sometimes we do not know of any transformations (arrows in UDR) coming into a distribution. In these cases there are two ways to generate samples from that distribution. If the number of possible values for the random variable are countable(not infinite), we use the built in categorical function in Hakaru. Otherwise, we use the built in weight function. Shown below is an example using the `weight` and `categorical` functions.

```
burglary <~ categorical([0.0001, 0.9999])
weight([0.95, 0.01][burglary],
return [true,false][burglary])
```

In this example, the value `burglary` is sampled from a `categorical` distribution. Then, the samples are reweighted so that `true` has a weight of 0.95 and `false` has a weight of 0.01. For the distribution samplings that are implemented in the standard library, the first argument is the PMF or PDF of the distribution and the second argument is the random variable. For example the following is the code that implements a model of the discrete Weibull distribution

```
def discreteWeibull(p prob, beta prob):
    x <~ counting()
    weight(if (x < 0): 0 else: (1 - p) ** (x ** beta)
              - (1 - p) ** ((x + 1) ** beta), return x)
```

The first line of code in the function definition, `x <~ counting()`, assigns an integer to x according to a uniform distribution across all integers. The first argument of `weight` is the PMF of the distribution and the second argument is the random variable. In this way, each instance of the random variable is given a weight equal to that instance's probability.

Continuous Distributions from PMF

This is accomplished similar to how discrete distributions with infinite categories implied by their PMF are implemented. The main difference is that the `lebesgue` function is used instead of `counting` to sample values from the real number line.

Testing Relationships Between Distributions

Recall that when developing the standard library, the UDR chart often implied multiple possible implementations for a given distribution. Naturally, we would expect alternative implementations for the same distribution to result in equivalent probabilistic models. This line of thought is the basis for the kinds of test cases we have implemented. More generally, we have the following hypothesis.

Assume we know a proven relationship between 2 statistical distributions, A and B, which allows us to transform A and B into distributions that are equivalent to each other. Further assume this transformation takes on one of the forms discussed in the Standard Library Development section. We hypothesize that by applying the appropriate transformations to implementations of A and B, we can create two Hakaru programs whose 'hk-maple' outputs will be equivalent to each other. Test cases that prove our hypothesis true indicate the validity of the Hakaru language implementation. Test cases that prove our hypothesis false indicate an underlying bug in the language definition which is to be passed back to the language developers.

Conclusions & Future Work

This project was aimed at increasing language accessibility and testing the validity of Hakaru. A lot of ground was covered on both of these aspects in this project, but compared to other popular languages, Hakaru is still under development and has a lot of room for improvement. For increasing the accessibility, work can be done on throwing more descriptive errors. There are many other distributions and tests that were not covered in this project and their implementation can add to the accessibility and robustness of Hakaru.

References

References