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

Hakaru is an experimental **probabilistic programming language**, which aims to simplify the implementation of statistical distributions.

- Niche application means that Hakaru is a small language with limited features.
- Running a hakaru program generates a stream of random numbers distributed according the statistical distribution modeled.
- Models can be compiled to C and Haskell for use in larger applications.

`hk-maple` is a provided inference algorithm that uses Maple to perform algebraic transformations on hakaru programs. In its default mode (Simplify), 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!

Motivation

Objectives

- Increase language accessibility: add new language features such as primivite mathematical functions (e.g. log, choose), new distributions, error handling, etc.
- Test language validity: use relationships between distributions to test the validity of program transformations like `hk-maple`.

Endeavours

- Standard library development
- Syntax-highlighting-for-hakaru package for Sublime Text (Figures 2 and 3)
- Test case writing
- Filling in missing language features

Key Concepts

- ⋆ *Hakaru program* $\iff$ *implementation of statistical distribution* $\iff$ *probabalistic model* $\iff$ *model.*
- ⋆ *Values pulled from a distribution are called measures. Measures* $\iff$ *samples.*
- ⋆ '$\sim$' vs '$<\sim$':
  - $X \sim Normal(0, 1)$: random variable, X, is distributed according to a normal distribution (statistics literature).
  - `x <~ normal(0, 1)`: pull a random sample from a normal distribution and bind it `x` (Hakaru code).
- ⋆ *PDF: Probability Density Function (continuous distributions).*
- ⋆ *PMF: Probability Mass Function (discrete distributions).*
- ⋆ *UDR chart: Univariate Distribution Relationship chart.*

Standard Library Development

Accessibility means the standard library implements commonly used statistical distributions. We have followed the UDR to implement distributions will the following guiding principles:

- Whenever possible, implement distributions as transformations on pre-existing models.
- In the case of multiple possible implementations, take the shortest possible path from a primitive distribution on the UDR.

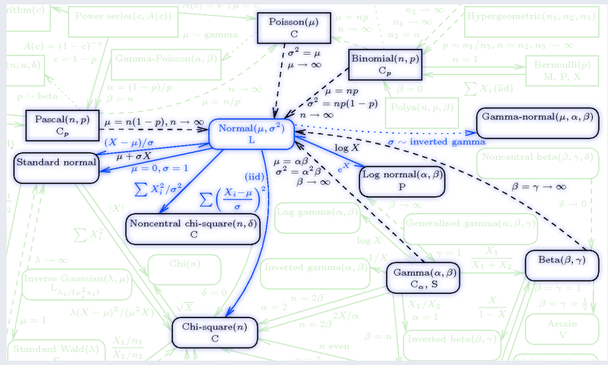


Figure 1: A snapshot of the UDR [1] shows how the normal distribution can be transformed into a multitude of other distributions.

Hakaru does not allow us to transform the model directly. Rather, we must apply transformations to samples pulled from the model using the bind $<\sim$ operator. Therefore, we are interested in implementing transformations of the form:

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

We can extend 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 normal random variables (see Figure 2).

```
def chiSq(means array(real), stdevs array(prob) ):
    q <~ plate _ of size(means): normal(means[_],stdevs[_])
    return summate i from 0 to size(q):
        ((q[i]-means[i])/stdevs[i])^2
```

Figure 2: Our implementation of the chi-square distribution.

Hakaru also lends itself well to Bayesian transformations, which take the following form. The gamma-poisson distribution can be described by such a transformation (see Figure 3).

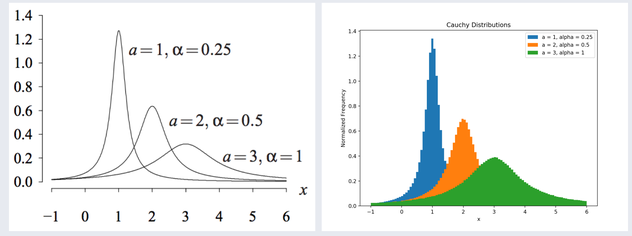
$$X \sim A(p) \Rightarrow Y \sim B(q, X) = C(p, q)$$

```
def gammaPoisson(shape prob, scale prob) measure(nat):
    mu <~ gamma(shape, scale)
    X <~ poisson(mu)
    return X
```

Figure 3: Our implementation of the gamma-poisson distribution.

In the case of unreachable distributions, we have resorted to defining models in terms of their PDF/PMF.

By plotting a histogram of a sample polulation drawn from a distribution's model and comparing its shape to that of its PDF, we can have some confidence that the implementation is correct without proof.



(a) (b)

Figure 4: A few plots of the Cauchy distribution are shown [1], along with a few histograms of data that have been sampled from a Hakaru program.

Testing Relationships Between Distributions

The validity of Hakaru can be tested by checking if it recognizes known relationships between distributions. More specifically:

> *Hypothesis: 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.*

Given that test cases are written in Hakaru, we are limited to testing relationships of the form discussed. Our test cases focus on relationships involving a small set of distributions. Results are summarized below.

Tests that pass help prove that Hakaru is valid. Tests that fail either indicate a language bug, or are the consequence of an intended design choice. In all cases, these results give the language developers useful information for future work on the language.

Conclusions & Future Work

This project was aimed at increasing language accessibility and testing the validity of Hakaru. A lot of ground was covered in 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. Here are just a few examples of where major development can still be done.

- Languages features: Hakaru can be expanded further with new features like import statements and error/exception handling.
- Standard library development: there are still many types of distributions left to be implemented, such as multivariate distributions.
- Testing: There are a plethora of relationships between distributions that have yet to be investigated.

References