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

Often, we wish to create a model for some kind of real world phenomenon so that we can extract useful information and learn more about it. For example, in a machine learning application, the goal is often to predict or infer information about the model based on the past outcomes of some kind of experiment. An experiment could result in many different outcomes, each one with a different likelihood (probability).

The mathematical functions that we use to describe these probabilties 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 variables is described by a probability mass function (PMF). As an example, let us consider one of the most ubiquitous and naturally occuring distributions, the normal distribution. We can say that a random variable, X, is normally distributed with the following notation.

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

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.

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

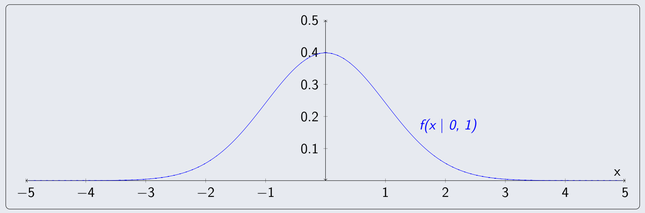

Figure 1: PDF of the normal distribution, with $\mu = 0$ and $\sigma = 1$

Hakaru is an experimental **probabilistic programming language**, which is designed so that probabilistic models (implementations of statistical distributions) can be easily expressed programatically. Running a hakaru program generates a stream of random numbers (known as samples) that are distributed according to the distribution defined in the program code. The language is quite small given its specialized domain. However, Hakaru programs can be compiled to C and Haskell, making for an easy way to export probabilistic models defined in Hakaru into imperative and functional programming domains. These can then be used within larger applications (e.g. simulating scenarios with inherent uncertainty, generating training sets for machine learning algorithms).

The purpose of Hakaru is to simplify the process of implementing efficient probabilistic models. The small size of the language solves the problem of simplifying model implementation. In order to simplify the process of making these implementations efficient, Hakaru includes some inference algorithms that transform Hakaru programs into other forms. Most pertinent to our project is 'hk-maple' (formerly known as 'simplify'). In short, 'hk-maple' 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 which may require billions of samplings, such as machine learning, this has the potential to save a significant amount of processing time!

Key Concepts

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

- *A Hakaru program is a probabalistic model.*
- *A Hakaru program is an implementation of a statistical distribution.*
- *Values that are pulled from a distribution are called samples, or measures.*
- *In mathematics, we write $X \sim Normal(0, 1)$ to denote that the random variable is normally distributed, while in Hakaru code we write* `x <~ normal(0, 1)` *, which pulls a sample from the distribution and binds it to the variable* `x`*.*
- *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

The purpose of our project is two-fold. Our first objective is to increase the accessibility to Hakaru for future users. In its current state, the Hakaru language only offers a small set of primitive distributions. Our main effort in this regard has been the development of a standard library in which we have implemented over 60 statistical distributions which cover a wide range of potential applications. Another effort to increase accessibility has been the development of a syntax highlighting package for Sublime Text; the only one currently implemented for a GUI text editor. This package has been invaluable in increasing code readability/writability, and optimizing our workflow.

Our second objective is to test the validity of the Hakaru language. Our focus is on writing test cases based on well-known relationships between the various distributions implemented in the standard library. In general, our test cases transform two related distributions so that the resulting distributions are (hypothetically) equivalent to each other. The number of possible test cases we could write in this vein is beyond the scope of this project. To this end, we have focused on comprehensively testing a small set distributions, with the intention of laying down the groundwork for future testing in this area.

It is also worth noting that probabilistic reasoning is a foundational technology in machine learning. Expansion of the Hakaru langauge presents an opportunity to learn more about this domain and potentially develop novel applications.

Standard Library Development

Whenever possible, we have implemented distributions as transformations on pre-existing models, starting with the primitive distributions that are native to Hakaru (normal, categorical, etc.). We have used the UDR to guide the development of the standard library, as it reveals many useful relationships bewteen relevant distributions (see Figure 2). In the case of multiple possible implementations for a distribution, we have opted for the implementation which adopts the shortest path from a primitive distribution on the UDR.

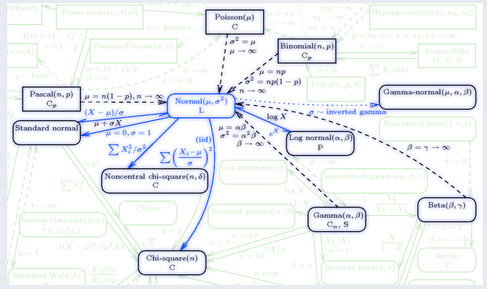

Figure 2: A snapshot of the UDR shows how the normal distribution can be transformed into a multitude of other distributions.

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.

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

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.

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

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.

Discrete Distributions from PMF

When the PMF implies a finite number of categories, the use of the primitive `categorical` distribution combined with the array constructor language feature lends itself very well to an implementation of the distribution. However, discrete distributions have PMFs implying an infinite number of categories. This cant be implemented using `categorical`, which requires a finite size array. In this case, we use the `weight` function primitive to Hakaru. The `weight` function assigns a weight to each sample. Shown below is an example using the `weight` function.

```
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

Conclusions & Future Work

References

References