Notes on Iterations of Distribution Representation (July 28th, 2017)

Dhruv Rajan

July 28, 2017

Contents

1 First Iteration

- Code at `https://github.com/dhruvrajan/probabilistic-haskell/blob/8eb317d225976c7883ff98b6ac625c87a0074036/probability/src/Distribution.hs`
- Used `MultiParameterTypeClasses`, `FlexibleInstances`, and `ExistentialQuantification` GHC extensions
- **Generic Probabiltiy Distribution**
  - Single type parameter (distribution type)
  - `sumProbabilities` method to sum the probabilities over its domain.
- **Discrete Distribution**
  - Typeclass with two type parameters: a distribution type, and a domain type
    - Example: `instance Discrete Bernoulli Bool` (A Bernoulli distribution defined over `[True, False]`)
    - `pmf` (Probability Mass Function) defined over the discrete domain

- **Continuous Distribution**
  - Typeclass with two type parameters: a distribution type, and a domain type
    - Example: `instance Continuous Normal Double` vs. `instance Continuous Normal Float`
  - `pdf` (Probability Density Function) defined over the continuous domain
  - `cdf` (Cumulative Distribution Function) defined over the continuous domain
- **Distribution Types**
  - Each distribution is represented as its own datatype. The constructor takes the distributions parameters in the constructor
    - Example: (`data Normal = Normal Double Double`)
- **Issue**
  - In this representation, the domain of a probability distribution is kept separate from its dataype. For example, A Bernoulli distribution is only paired with its specific domain (could be over Bool, over Int, over String, etc.) in a typeclass instance.
  - This produces interesting, but incorrect behavior. A Bernoulli distribution is not sufficiently defined by just its parameter, without a domain-and thus should not be allowable. The next iteration attemts to associate distributions with their domains via a type parameter to their respective datatypes.

2 Second Iteration

- Code at `https://github.com/dhruvrajan/probabilistic-haskell/blob/ac8197ded0bada83be2ecbc54ff509bd4847bbc4/probability/src/Distribution.hs`
- Now only using the `Existential Quantification` GHC Extension
- **Generic Probability Distribution**
  - Does not have a typeclass. Probability distributions must either be Discrete, or Continuous. The original generic typeclass was not really of any use, so this is a cleaner distinction.

- **Discrete Distribution**
  - Typeclass with a single type parameter (distribution type)
  - `pmf` (Probability Mass Function) defined over some discrete domain
    - Note the function's type signature:
      - `pmf \:: Eq b => a b -> b -> Double`
      - It requires that `a` has some type parameter `b`. Thus, every discrete distribution datatype must specify its domain type as a type parameter.
      - Example: `data Bernoulli a = Bernoulli Double a a` where the constructor's first parameter is the single parameter to the Bernoulli distribution, and the second two are the outcomes *success* and *fail*
- **Continuous Distribution**
  - Typeclass with a single type parameter (distribution type)
  - `pmf` and `cdf` functions defined over some continuous domain
  - Note the functions' type signatures:
    - `pdf \:: (RealFloat b, Fractional b) => a b -> b -> Double`
    - `cdf \:: (RealFloat b, Fractional b) => a b -> b -> Double`
    - So `b` the domain type of any continuous distribution `a`, must be an instance of `RealFloat` and `Fractional`
  - Example: `data Normal a = Normal a a`
    - For a normal distribution, the type parameter specifies the types of the Normal parameters μ and σ; these could be `Double`, or `Float`, etc.
- **Problems Solved**
  - This second iteration allows for a clean representation of multiple types of distributions. Each domain is associated with its domain in its datatype. This will help in cases like implementing sampling methods for each distribution. Additionally, it makes it easy to tell whether an `Element` is discrete or continuous, depending on its distribution.