150PP Final Project Proposal: Abstracting for a latent and observable variable

Michael James

November 18, 2014

1 Introduction

Fun, as a language, has problems but it inspired a higher-order way of thinking about probabilistic models. The paper "A Model-Learner Pattern for Bayesian Reasoning" introduced a way to reuse and abstract probabilistic code–albeit with some obtuse F# metaprogramming.

Many applications do not require the full abstraction power of four parameters, namely a hyperparameter, a prior, an input, and an output. Instead, questions we have sought to answer in this course have been largely of the form: *Given that I saw X, what is the probability Y caused it?* Our solutions in Haskell using the probability monad and `pfilter` work and are only readable if the reader already knows what is going on. Perhaps by improving our `Distribution` kind with two parameters we can make both more readable and more expressive code. It may also be possible that with a more specialized kind, maybe with name and kind:

```
Bayesian ::  * -> * -> *
```

we can write spec implementations that will natively include improvements over previous class attempts. Simply put, with a designated latent and observed variable field, what happens?

2 The Work

I plan to create an API suite and simple implementation for this `Bayesian` kind. I plan to explore various type manipulations and determine what is immediately useful.

3 The Goal

I would like to produce a sensible API for latent and observed variables. Hopefully one that supports better code reuse than the `Distribution` kind.

4 Stretch Goals

If the API comes together quickly, then the next reasonable step is to write something with it. A reasonable something would be the dice-world problems because as a class, we have implementations and understanding of the problem with which to compare to my implementation.