Introduction to probabilistic programming (with PyMC3)

A. Richards

02.08.2017

Probabilistic programming

A probabilistic programming language makes it easy to:

1. write out complex probability models
2. And subsequently solve these models automatically.

Generally this is accomplished by:

1. Random variables are handled as a primitive
2. Inference is handled behind the scenes
3. Memory and processor management is abstracted away

The pros and the cons

Why you might want to use probabilistic programming

1. Customization - We can create models that have built-in hypothesis tests
2. Propagation of uncertainty - There is a degree of belief associated prediction and estimation
3. Intuition - The models are essentially 'white-box' which provides insight into our data

Why you might NOT want use out probabilistic programming

1. Deep dive - Many of the online examples will assume a fairly deep understanding of statistics
2. Overhead - Computational overhead might make it difficult to be production ready
3. Sometimes simple is enough - The ability to customize models in almost a plug-n-play manner has to come with some cost.

Bayesian Inference

Degree of belief

You are a skilled programmer, but bugs still slip into your code. After a particularly difficult implementation of an algorithm, you decide to test your code on a trivial example. It passes. You test the code on a harder problem. It passes once again. And it passes the next, *even more difficult*, test too! You are starting to believe that there may be no bugs in this code...

Bayesian methods for hackers

This is a nice intro to Bayesian thinking done on kdnuggets (using PyMC3)

Some definaitons

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{1}$$

- prior - $P(\theta)$ - one's beliefs about a quantity before presented with evidence
- posterior - $P(\theta|x)$ - probability of the parameters given the evidence
- likelihood - $P(x|\theta)$ - probability of the evidence given the parameters
- normalizing constant - $P(x)$

- $P(\theta)$: This big, complex code likely has a bug in it.
- $P(\theta|X)$: The code passed all X tests; there still might be a bug, but it is less likely now.

PyMC3

- Developed by John Salvatier, Thomas Wiecki, and Christopher Fonnesbeck (Salvatier et al., 2016)
- Comes with loads of good examples
- API is is not backwards compartible with models specified in PyMC2

```
import pymc3 as pm

n,h,alpha,beta,niter = 100,61,2,2,1000

with pm.Model() as model: # context management
    # define priors and likelihood
    p = pm.Beta('p', alpha=alpha, beta=beta)
    y = pm.Binomial('y', n=n, p=p, observed=h)

    # inference
    start = pm.find_MAP() # initial state for MCMC
    step = pm.Metropolis() # Have a choice of samplers
    trace = pm.sample(niter, step, start)
```

Markov chain Monte Carlo (MCMC)

MCMC

- It is an family of algorithms for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult.
- The sequence can then be used to approximate the distribution
- It allows for inference on complex models

A particularly useful class of MCMC, known as Hamliltonian Monte Carlo, requires gradient information which is often not readily available so PyMC3 uses Theano to get around this problem. Another class of MCMC that has recently made this whole field a lot more interesting is the No-U-turn sampler (NUTS) because there are self-tuning strategies (Hoffman and Gelman, 2014).

One of the really nice things about probabilistic programming is that you do not have to know how inference is performed, but it can be useful.

- MCMC for Dummies
- More on Hamliltonian MCMC (Not for dummies)

PyMC3 is an improvement over PyMC2

- Intuitive model specification syntax e.g.

$$x \sim N(0, 1) \text{ becomes } x = \mathrm{Normal}(0, 1)$$

- Powerful sampling algorithms such as the No U-Turn Sampler
- Variational inference: ADVI for fast approximate posterior estimation as well as mini-batch ADVI for large data sets.
- Relies on Theano which provides:
  - Numpy broadcasting and advanced indexing
  - Linear algebra operators
  - Computation optimization and dynamic C compilation
  - Simple extensibility
- Transparent support for missing value imputation

Getting started

We will be using Probabilistic Programming to perform automatic Bayesian inference on user-defined probabilistic models

- PyMC3 repo
- Getting started guide
- Bayesian methods for hackers

woot

-

coin flip example

•

woot

Recommenders

-

woot

-

Recommenders

The example

We will be walking through the PyMC3 PMF example

-

woot

-

Where to go from here

Examples, examples, examples...

- PyMC3 repo
- Getting started guide
- Bayesian methods for hackers
- Blog by Thomas Wiecki
- Doing Bayesian Data Analysis by John Kruschke
- Resource by Mark Dregan

There is also PyStan (Stan paper)

References I

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55.