CODING DIRICHLET PROCESS MODELS VIA PROBABILISTIC PROGRAMMING

FRANK WOOD AND YURA PEROV

Introduction

Dirichlet process mixtures, reviewed in depth in [Teh, 2010, Orbanz and Teh, 2010] and the subject of excellent tutorial presentations by Teh [2007], are widely used in Bayesian unsupervised clustering and density estimation tasks. In particular the infinite Gaussian mixture model Rasmussen [1999] has been widely used. A canonical example application is neural spike sorting Wood and Black [2008] (this latter applied work also highlights efficient sequential inference).

Stick-breaking constructions [Ishwaran and James, 2001] make coding some Bayesian nonparametric primitives in probabilistic programming systems relatively straightforward. Additionally there is an interesting and deep (in not fully or even well described in the literature) connection between the action of Dirichlet-like stochastic processes and relaxations of the programming languages technique called memoization [Michie, 1968]. The latter, simply put, is the idea of wrapping a function in a hashmap so that it remembers and thus never needs to recompute a return value if called again with the same arguments. Memoization can sometimes give rise to very simple dynamic programming algorithms.

Questions :

(1) Generalize the DPmem code below to default to duplicate the functionality of `mem` if the concentration is 0.

Read the following code very carefully as it generalizes the Dirichlet process in a way that is natural in probabilistic programming – namely the base distribution of the Dirichlet process is a procedure. In probabilistic programming applying a procedure produces a *sample* so it is possible to use any procedure as the base distribution in any Dirichlet process. What is more, the calling interface to the Dirichlet process is the same as `mem`, i.e. it is a function that takes a function and returns a function that calls the inner function when certain conditions are met (like in `mem`, for instance, if there doesn't already exist a return value for the specific provided arguments; or, like in a DP-based model, if a sample is generated from the base distribution rather than simply returning one of the already generated base-distribution samples). The original Church paper [?] introduced this idea and called it stochastic memoization, a powerful realisation that the Dirichlet process and its ilk provide a stochastic generalisation of `mem`. These ideas and their connection to deeper ideas about computability have also been discussed in a short workshop paper [Roy et al., 2008].

```
; sample-stick-index is a procedure that samples an index from
; a potentially infinite dimensional discrete distribution
; lazily constructed by a stick breaking rule
[assume sample-stick-index (lambda (breaking-rule index)
  (if (flip (breaking-rule index))
      index
      (sample-stick-index breaking-rule (+ index 1))))]


; sethuraman-stick-picking-procedure returns a procedure
; that picks a stick each time its called from the set of sticks
; lazily constructed via the closed-over one-parameter stick breaking
; rule
[assume make-sethuraman-stick-picking-procedure (lambda (concentration)
  (begin (define V (mem (lambda (x) (beta 1.0 concentration))))
    (lambda () (sample-stick-index V 1))))]


; DPmem is a procedure that takes two arguments -- the concentration
; to a Dirichlet process and a base sampling procedure
; DPmem returns a procedure
[assume DPmem (lambda (concentration base)
  (begin
    (define get-value-from-cache-or-sample
                            (mem (lambda (args stick-index)
                                          (apply base args))))
    (define get-stick-picking-procedure-from-cache
                             (mem (lambda (args)
               (make-sethuraman-stick-picking-procedure concentration))))
    (lambda varargs
        ; when the returned function is called, the first thing
        ; it does is get  the cached stick breaking
        ; procedure for the passed in arguments
        ; and _calls_ it to get an index
        (begin
           (define index ((get-stick-picking-procedure-from-cache varargs)))
            ; if, for the given set of arguments and
```

```
          ; just sampled index a return value has already
          ; been computed, get it from the cache
          ; and return it, otherwise sample a new value
          (get-value-from-cache-or-sample varargs index)))))]
```

(See this code online.)

(2) Generalize the code above such so that it uses the two parameter stick breaking construction and define `DPmem` in terms of currying this new function. The suggested route is to generalize `make-sethuraman-stick-picking-procedure` so that it uses the more general Pitman-Yor stick breaking in the code on `http://www.robots.ox.ac.uk/~fwood/anglican/examples/dp_mixture_model/index.html`.

(3) Implement a Pitman-Yor process mixture model.

Use the data (one of six World Bank economic indicators) located at `http://www.robots.ox.ac.uk/~fwood/anglican/teaching/mlss2014/py_mem/data.csv`. Please, find file columns description at `http://www.robots.ox.ac.uk/~fwood/anglican/teaching/mlss2014/py_mem/data_description.txt`.

Cluster countries by a Pitnam-Yor process mixture model in a naïve Bayes way. That is, your base distribution should return a pair (an Anglican list) with independently drawn mean and standard deviation of this cluster for the correspondent economic indicator.

The observations (data) can be loaded via `observe-csv` directive (see description on Anglican syntax reference page):

```
[observe-csv
  "http://www.robots.ox.ac.uk/~fwood/anglican/teaching/mlss2014/py_mem/data.csv"
  (apply normal (country-parameters $1)) $4]
```

(See this code online.)

Here \$1 is the country name (from CSV), \$4 is the indicator "GDP per capita, PPP (current international \$)".

Choose any two countries on the same continent by your interest, and compare how close they are by comparing whether they were assigned to the same cluster or not.

Hint: due to the fact that the support of the base distribution is continuous, you can do this comparison by just simply checking whether two countries have the same parameter.

(4) Implement a hierarchical Dirichlet process mixture model as described in [Teh et al., 2004] (see the Wikipedia page for quick intro).

$$G_0 \mid \alpha_0, H \sim \mathrm{DP}(\alpha_0, H)$$
$$G_j \mid G_0, \{\alpha_j\} \sim \mathrm{DP}(\alpha_j, G_0)$$

Consider countries to be divided into groups by continent (the 8th column in the CSV file).

Then use the model and data as follows:

(1) Use the same economic indicator which you used for the previous exercise, and investigate how closer/farther two countries on the same continent became due to the usage of a different model, i.e. a hierarchical one.
(2) Sample and plot GDPs of two groups: "Europe" and "Africa".

References

Hemant Ishwaran and Lancelot F James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453), 2001.

Donald Michie. Memo functions and machine learning. *Nature*, 218(5136):19–22, 1968.

Peter Orbanz and Yee Whye Teh. Bayesian nonparametric models. In *Encyclopedia of Machine Learning*, pages 81–89. Springer, 2010.

Carl Edward Rasmussen. The infinite gaussian mixture model. In *NIPS*, volume 12, pages 554–560, 1999.

DM Roy, VK Mansinghka, ND Goodman, and JB Tenenbaum. A stochastic programming perspective on nonparametric bayes. In *Nonparametric Bayesian Workshop, Int. Conf. on Machine Learning*, volume 22, page 26, 2008.

Yee Whye Teh. Dirichlet processes: Tutorial and practical course. *Machine Learning Summer School*, 2007.

Yee Whye Teh. Dirichlet process. In *Encyclopedia of machine learning*, pages 280–287. Springer, 2010.

Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. In *NIPS*, 2004.

Frank Wood and Michael J Black. A nonparametric bayesian alternative to spike sorting. *Journal of neuroscience methods*, 173(1):1–12, 2008.