Automatic code rewriting in probabilistic programming
Internship supervised by Hongseok Yang at the University of Oxford

Diane Gallois-Wong

September 11, 2015
Introduction

Probabilistic programming languages: short, intuitive code to describe probabilistic models, built-in inference algorithms

LDA model:
- straightforward, naive implementation: easy to write but bad performance
- collapsed implementation: complex, better performance

Automatic transformation of naive code into collapsed version
Formalism with a lambda calculus, program analysis techniques

Anglican: a probabilistic programming language (Oxford)
Bayes’ Theorem

\[ p(H|X) = \frac{p(X|H) \cdot p(H)}{p(X)} \]

*H*: hypothesis, *X*: observation (events)

prior probability, posterior probability, likelihood function
Probabilistic programming language

Integrated in the functional language Clojure (dialect of Lisp)

(let [more-heads (sample (flip (/ 1 2)))
      coin (flip (if more-heads (/ 2 3) (/ 1 3)))]
  (observe coin true)
  (observe coin true)
  (observe coin true)
  (predict more-heads))
Latent Dirichlet Allocation (LDA)

**Topic Model**

Input: collection of documents which are collections of words
Aim: classify the documents using topics

**LDA**: generative topic model

\[ \theta_d \sim \text{Dirichlet}(\alpha) \]
prob. vector over topics for each document \( d \)

\[ \varphi_k \sim \text{Dirichlet}(\beta) \]
prob. vector over words for each topic \( k \)

\[ z_{d,n} \sim \text{Discrete} (\theta_d) \]
for each position \( n \) in document \( d \),
choose a topic \( z_{d,n} \) according to \( \theta_d \)

\[ w_{d,n} \sim \text{Discrete} (\varphi_{z_{d,n}}) \]
then a word \( w_{d,n} \) according to \( \varphi_{z_{d,n}} \)
Highly expressive latent variables

Naive implementation of LDA:

Inputs: a corpus of documents $w$

where $w_{d,n}$ is the word at position $n$ in document $d$
and hyperparameters $\alpha$ and $\beta$

for each $d$: $\theta_d = \text{sample Dirichlet}(\alpha)$
for each $k$: $\varphi_k = \text{sample Dirichlet}(\beta)$
for each $d$ and each $n$:

$z_{d,n} = \text{sample Discrete}(\theta_d)$
observe $\text{Discrete}(\varphi_{z_{d,n}}) w_{d,n}$
predict $z$
Conjugate Prior

\[
\begin{align*}
\theta & \sim \text{Dirichlet}(\alpha) \\
\mathbf{x} \mid \theta & \sim \text{Discrete}(\theta) \\
\theta \mid \mathbf{x} & \sim \text{Dirichlet}(f(\alpha, \mathbf{x}))
\end{align*}
\]

Dirichlet is a \textit{conjugate prior} with a Discrete likelihood.

\text{Dirichlet}(\alpha) \text{ interpretation: class } i \text{ observed } \alpha_i - 1 \text{ times.}

\[ f(\alpha, x) = (\alpha \text{ where component with index } x \text{ incremented}) \]
Conjugate Prior and Dirichlet Process

\[ \theta \sim \text{Dirichlet}(\alpha) \]
\[ x \mid \theta \sim \text{Discrete}(\theta) \]
\[ \theta \mid x \sim \text{Dirichlet}(f(\alpha, x)) \]

\[ \theta \sim \text{Dirichlet}(\alpha) \]
\[ x_0 \sim \text{Discrete}(\theta) \]
\[ x_1 \sim \text{Discrete}(\theta) \]
\[ x_2 \sim \text{Discrete}(\theta) \]
\[ x_0 \sim \text{Discrete}(\alpha) ; \quad \alpha^0 = f(\alpha, x_0) \]
\[ x_1 \sim \text{Discrete}(\alpha^0) ; \quad \alpha^1 = f(\alpha^0, x_1) \]
\[ x_2 \sim \text{Discrete}(\alpha^1) ; \quad \alpha^2 = f(\alpha^1, x_2) \]
\[ (\theta \sim \text{Dirichlet}(\alpha^2)) \]

\[ \theta \text{ is marginalised.} \]
Necessary conditions imposed to make things easier.

Main condition to marginalise $\theta$:
$\theta$ should only appear as argument to a Discrete distribution.

Automatically checked through multiple steps of program analysis, including a *type and effect system*. 
State and modular functions

\[ x_1 \sim \text{Discrete}(\theta) \quad \longrightarrow \quad x_1 \sim \text{Discrete}(\alpha^0) ; \quad \alpha^1 = f(\alpha^0, x_1) \]

\[ x_1 = \text{sample Discrete}(\theta) \quad \longrightarrow \quad [x_1; S] = S\text{-sample } S\text{-Discrete}(\theta) S \]
An example of code

```scheme
(let [
  maketheta (fn [_] (sample (dirichlet alpha)))
  thetas (map maketheta range-D)
  makephi (fn [_] (sample (dirichlet beta*)))
  phis (map makephi range-K)

  f (fn [phis theta] (fn [w]
    (let [z (sample (discrete theta))]
      (observe (discrete (nth phis z)) w)
      z)))

  z-corpus (map (fn [d] (map (f phis (nth thetas d)) (nth corpus d))) range-D)
  (predict z-corpus))
```
Vocabulary words: 0, 1, 2.

Input is a corpus generated according to two topics: [0.5 ; 0 ; 0.5] , [0 ; 0.5 ; 0.5]

Plot estimations of a topic.
Conclusion

- Formal definition of the automatic transformation and working implementation (small subset of Anglican)

- Potential improvements: extend the subset, weaken necessary conditions, improve memory management of the state, extend to other conjugate priors

- Will be used as part of a much more general automatic code transformation developed by my supervisor to be eventually added to Anglican

- High level transformation: affects the model itself