BayesPy: Variational Bayesian Inference in Python

Jaakko Luttinen

Department of Information and Computer Science
Aalto University, Finland

Abstract
BayesPy is an open-source Python software package for performing variational Bayesian inference. It is based on the variational message passing framework and supports conjugate exponential family models. By removing the tedious task of implementing the variational Bayesian update equations, the user can construct models faster and in a less error-prone way. Simple syntax, flexible model construction and efficient inference make BayesPy suitable for both average and expert Bayesian users.

Keywords: Variational Bayes, variational message passing, Python, probabilistic programming

1. Introduction
Bayesian framework provides theoretically solid and consistent way to construct models and perform inference. In practice, however, the inference is usually analytically intractable and is therefore based on approximation methods such as variational Bayes (VB), expectation propagation (EP) and Markov chain Monte Carlo (MCMC) (Bishop, 2006). Deriving and implementing the formulas for an approximation method is often straightforward but tedious, time consuming and error prone.

BayesPy is a Python package providing tools for constructing Bayesian models and performing variational Bayesian inference easily and efficiently. It is based on variational message passing (VMP) framework which defines a simple and local message passing protocol (Winn and Bishop, 2005). This enables implementation of small general modules that can be used as building blocks for large complex models. BayesPy offers a comprehensive collection of built-in blocks that can be used to build a wide range of models and a simple interface for implementing custom blocks. The package is written for Python 3 and released under the GNU General Public License v3.0 (GPLv3).

Several other projects have similar goals for making Bayesian inference easier and faster to apply. VB inference is available in Bayes Blocks (Raiko et al., 2007), VIBES (Bishop et al., 2002) and Infer.NET (Minka et al., 2012). Bayes Blocks is an open-source C++/Python package but it is limited to scalar Gaussian nodes, a few deterministic functions and fully factorial posterior approximation, thus making it very limited. VIBES is an old software package for Java, released under the revised BSD license, but it is no longer actively developed. VIBES has been replaced by Infer.NET, which supports a wide range of posterior approximations. However, Infer.NET is partly closed source and licensed for non-commercial use only. Instead of VB inference, mainly MCMC is supported by other projects such as

©2014 Jaakko Luttinen.
Luttinen, PyMC (Patil et al., 2010), OpenBUGS (Thomas et al., 2006), Dimple (Hershey et al., 2012) and Stan (Stan Development Team, 2014). Thus, there is a need for an open source and maintained VB software package.

2. Design and Features

BayesPy can be used to construct a wide range of conjugate exponential family models. The documentation provides detailed examples of how to construct, for instance, regression models, principal component analysis models, linear state-space models, mixture models and hidden Markov models. In addition, BayesPy has been used in two publications about parameter expansion and time-varying dynamics for linear state-space models (Luttinen, 2013; Luttinen et al., 2014), which show that it is applicable to large complex models requiring efficient inference.

Using BayesPy for Bayesian inference consists of four main steps: constructing the model, providing data, finding the posterior approximation and examining the results. The user constructs the model from small modular blocks called nodes. Roughly, each node corresponds to a latent variable, a set of observations or a deterministic function. Some of the nodes may be set as observed by providing data. After that, inference engine is initialized and the message passing algorithm is run in order to obtain the posterior approximation. The resulting posterior can be examined, for instance, by using a few built-in plotting functions or printing the parameters of the posterior distributions.

Nodes are the key part of BayesPy as they are used to construct the model. There are two types of nodes in BayesPy: stochastic and deterministic. Stochastic nodes correspond to probability distributions and deterministic nodes correspond to functions. Built-in stochastic nodes include all common exponential family distributions (e.g., Gaussian, gamma, Dirichlet and Poisson distributions), a general mixture distribution and a few complex nodes for dynamic variables (e.g., discrete and Gaussian Markov chains). Built-in deterministic nodes include a gating node and a general sum-product node. In case a model cannot be constructed using the built-in nodes, the documentation provides instructions for implementing new nodes.

BayesPy is designed to be simple enough for average users but flexible and efficient enough for expert users. One goal is to keep the syntax easy and intuitive to read and write by making it similar to the mathematical formulation of the model. Other notable features of BayesPy include easy handling of missing values, parameter expansions (Qi and Jaakkola, 2007) for speeding up inference and monitoring of the variables by plotting the posterior distributions during the message passing iteration. For developers, the unit testing framework helps in finding bugs, making changes and implementing new features in a robust manner.

BayesPy can be installed similarly to other Python packages. It requires Python 3 and a few popular packages: NumPy, SciPy, matplotlib and h5py. The latest release can be installed from Python Package Index (PyPI) and detailed instructions can be found from the comprehensive online documentation (Luttinen, 2014). The latest development version is available at GitHub[1], which is also the platform used for reporting bugs and making pull requests.

1. https://github.com/bayespyp/bayespyp
3. Example

In order to demonstrate BayesPy, this section solves an extremely simple problem but which includes the main steps of using BayesPy. The task is to estimate the unknown mean and precision parameters of a Gaussian distribution given ten observations. Thus, the likelihood function is Gaussian:

\[
p(y_1, \ldots, y_{10}|\mu,\tau) = \prod_{n=1}^{10} N(y_n|\mu,\tau),
\]

where the Gaussian distribution is parameterized by precision instead of variance. The unknown mean \(\mu\) and precision \(\tau\) are given broad Gaussian and gamma priors, respectively:

\[
p(\mu) = \mathcal{N}(\mu|0,10^{-6}), \quad p(\tau) = \mathcal{G}(\tau|10^{-6},10^{-6}).
\]

Figure 1 shows the graphical model of this simple model. The model is constructed in BayesPy by creating nodes:

```python
from bayespy.nodes import GaussianARD, Gamma
mu = GaussianARD(0, 1e-6)
tau = Gamma(1e-6, 1e-6)
y = GaussianARD(mu, tau, plates=(10,))
y.observe([4.5, 3.9, 6.3, 5.6, 4.9, 2.8, 7.4, 6.1, 4.8, 2.1])
```

These two nodes represent the two unknown variables that we are interested in. They are the parameters of the Gaussian distribution of the observed variable:

```python
from bayespy.nodes import GaussianARD, Gamma
mu = GaussianARD(0, 1e-6)
tau = Gamma(1e-6, 1e-6)
y = GaussianARD(mu, tau, plates=(10,))
y.observe([4.5, 3.9, 6.3, 5.6, 4.9, 2.8, 7.4, 6.1, 4.8, 2.1])
```

The number of observations is given as plates which define the number of repetitions. Note that the syntax of the model construction is similar to the mathematical formulation which defines the conditional distributions for each variable. Thus, it is easy to understand the model from the Python code. After the model has been constructed, some nodes are marked as observed and given data:

```python
y.observe([4.5, 3.9, 6.3, 5.6, 4.9, 2.8, 7.4, 6.1, 4.8, 2.1])
```

Next, we want to find the posterior approximation for our latent variables. We create the VB inference engine:

```python
from bayespy.inference import VB
Q = VB(y, mu, tau)
```

The inference engine takes all the stochastic nodes of the model as input. The message passing algorithm can be run by calling `update` method:
The algorithm is run for 100 iterations or until it converges. In this case, the algorithm converges in four iterations. The posterior approximation can be examined, for instance, by plotting the probability density functions of the latent variables:

```python
import bayespy.plot as bpplt
import numpy as np
bpplt.pyplot.figure()
bpplt.pdf(Q[µ], np.linspace(0, 10, num=300))
bpplt.pyplot.figure()
bpplt.pdf(Q[τ], np.linspace(0, 2, num=300))
bpplt.pyplot.show()
```

Figure 2 shows the resulting approximate posterior distributions. The module `bayespy.plot` contains functions for simple plotting tasks, such as plotting the probability density function or the Hinton diagram of a variable. Note that the standard `pyplot` module of Matplotlib is available at the `plot` module of BayesPy.

4. Conclusions

BayesPy provides a simple and efficient way to construct conjugate exponential family models and to find the variational Bayesian posterior approximation in Python. Future plans for BayesPy include implementing more inference engines (e.g., maximum likelihood, expectation propagation and Gibbs sampling), improving the VB engine (e.g., collapsed variational inference [Hensman et al., 2012] and Riemannian conjugate gradient method [Honkela et al., 2010]) and implementing nodes for non-parametric models (e.g., Gaussian processes and Dirichlet processes).

References

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2nd edition, 2006.

Christopher M. Bishop, David Spiegelhalter, and John Winn. VIBES: a variational inference engine for Bayesian networks. In Suzanna Becker, Sebastian Thrun, and Klaus
BayesPy: Variational Bayesian Inference in Python

Obermayer, editors, Advances in Neural Information Processing Systems 15. MIT Press, Cambridge, MA, 2002.

James Hensman, Magnus Rattray, and Neil D. Lawrence. Fast variational inference in the conjugate exponential family. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 23. MIT Press, Cambridge, MA, 2012.

Shawn Hershey, Jeffrey Bernstein, Bill Bradley, Andrew Schweitzer, Noah Stein, Theophane Weber, and Benjamin Vigoda. Accelerating inference: towards a full language, compiler and hardware stack. Computing Research Repository, 2012.

Antti Honkela, Tapani Raiko, Mikael Kuusela, Matti Tornio, and Juha Karhunen. Approximate Riemannian conjugate gradient learning for fixed-form variational Bayes. Journal of Machine Learning Research, 11:3235–3268, 2010.

Jaakko Luttinen. Fast variational Bayesian linear state-space model. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip elezn, editors, Machine Learning and Knowledge Discovery in Databases, volume 8188 of Lecture Notes in Computer Science, pages 305–320. Springer, 2013. doi:10.1007/978-3-642-40988-2_20.

Jaakko Luttinen. BayesPy online documentation. http://bayespy.org, 2014.

Jaakko Luttinen, Tapani Raiko, and Alexander Ilin. Linear state-space model with time-varying dynamics. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, Machine Learning and Knowledge Discovery in Databases, volume ? of Lecture Notes in Computer Science, page ?? Springer, 2014. To appear.

Tom Minka, John Winn, John Guiver, and David A. Knowles. Infer.NET 2.5, 2012. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Anand Patil, David Huard, and Christopher J. Fonnesbeck. PyMC: Bayesian stochastic modelling in Python. Journal of Statistical Software, 35(4):1–81, 2010.

Yuan (Alan) Qi and Tommi S. Jaakkola. Parameter expanded variational Bayesian methods. In Bernhard Schölkopf, John Platt, and Thomas Hoffman, editors, Advances in Neural Information Processing Systems 19, pages 1097–1104. MIT Press, Cambridge, MA, 2007.

Tapani Raiko, Harri Valpola, Markus Harva, and Juha Karhunen. Building blocks for variational Bayesian learning of latent variable models. Journal of Machine Learning Research, 2007.

Stan Development Team. Stan: A c++ library for probability and sampling, version 2.4, 2014. URL http://mc-stan.org/

Andrew Thomas, Bob O’Hara, Uwe Ligges, and Sturtz Sibylle. Making BUGS open. R News, 6(1):12–17, 2006.

John Winn and Christopher M. Bishop. Variational message passing. Journal of Machine Learning Research, 6:661–694, 2005.