ELFI: Engine for Likelihood Free Inference

Jarno Lintusaari[1], Henri Vuollekoski[1], Antti Kangasrääsiö[1], Kusti Skytén[1], Marko Järvenpää[1], Michael Gutmann[2], Aki Vehtari[1*], Jukka Corander[3*], and Samuel Kaski[1*]

[1]Department of Computer Science, Aalto University, Helsinki, Finland
[2]School of Informatics, The University of Edinburgh, Edinburgh, United Kingdom
[3]Department of Biostatistics, University of Oslo, Oslo, Norway
[*]equal contribution
{*jarno.lintusaari,henri.vuollekoski,antti.kangasraasio,kusti.skyten, marko.j.jarvenpaa,aki.vehtari,samuel.kaski*}*@aalto.fi, michael.gutmann@ed.ac.uk, jukka.corander@medisin.uio.no*

August 3, 2017

Abstract

The Engine for Likelihood-Free Inference (ELFI) is a Python software library for performing likelihood-free inference (LFI). ELFI provides a convenient syntax for specifying LFI models commonly composed of priors, simulators, summaries, distances and other custom operations. These can be implemented in a wide variety of languages. Separating the modelling task from the inference makes it possible to use the same model with any of the available inference methods without modifications. A central method implemented in ELFI is Bayesian Optimization for Likelihood-Free Inference (BOLFI), which has recently been shown to accelerate likelihood-free inference up to several orders of magnitude. ELFI also has an inbuilt support for output data storing for reuse and analysis, and supports parallelization of computation from multiple cores up to a cluster environment. ELFI is designed to be extensible and provides interfaces for widening its functionality. This makes addition of new inference methods to ELFI straightforward and automatically compatible with the inbuilt features.

1 Introduction

The Engine for Likelihood-Free Inference (ELFI) is a statistical software package for likelihood-free inference written in Python. The term "likelihood-free inference" (LFI) refers to a family of inference methods that can be used when the likelihood function is not computable or otherwise available, but it is possible to simulate from the model (see e.g. Lintusaari et al., 2017). Other names for likelihood-free inference or closely related approaches include Approximate Bayesian Computation (ABC) (see e.g. Marin et al., 2012; Lintusaari et al., 2017), simulator-based inference, approximative Bayesian inference and indirect inference.

Likelihood-free inference uses outputs of generative models to perform inference. In LFI the generative models are commonly composed of priors and user-specified simulators. To perform inference, the simulated data is usually summarized and a distance is taken between the summaries of the simulated and observed data. In ELFI, the simulators, summaries distances, etc. are called operations and can be implemented in a wide variety of languages.

One of the main features in ELFI is the convenient syntax of combining these operations into a network (Figure 1) that we call an LFI model. Once the LFI model is specified, it can be used with any of the available inference algorithms. ELFI also supports parallelization of the inference from a single computer up to a computational cluster, and storing the generated data for reuse, post-processing and further analysis. ELFI has emerged from the prior research on the subject by the authors (Lintusaari et al., 2016; Gutmann and Corander, 2016; Lintusaari et al., 2017; Kangasrääsiö et al., 2017) and was used by Kangasrääsiö et al. (2017) and Kangasrääsiö and Kaski (2017).

2 Software design principles

ELFI was designed to support likelihood-free inference research both from the practitioners' and methodologists' point of view. We aim for an easy-to-use ecosystem where practitioners will find the state-of-the-art inference methods, and methodologists models and data to aid in method development and assessment. We will next describe some of the key features in ELFI for both of these target groups.

2.1 Features for practitioners

For practitioners ELFI provides a convenient interface for quickly specifying complex generative models with directed acyclic graphs (DAG). The models can then be directly used with any of the available inference methods. We have provided an initial set of methods that can handle different types of scenarios: basic rejection sampling works for simulators that can generate

```python
# Define your simulator and summaries
def simulator(t1, t2, batch_size=1, random_state=None):
    # Implementation comes here...

def summary(data, argument=0)
    # Implementation comes here...

y = # Observed data...

# Specify the model
t1 = elfi.Prior('uniform', -2, 4)
t2 = elfi.Prior('normal', t1, 5)
SIM = elfi.Simulator(simulator, t1, t2, observed=y)
S1 = elfi.Summary(summary, SIM)
S2 = elfi.Summary(summary, SIM, 2)
d = elfi.Distance('euclidean', S1, S2)

# Run the rejection sampler
rej = elfi.Rejection(d, batch_size=10000)
result = rej.sample(1000, threshold=0.1)
```

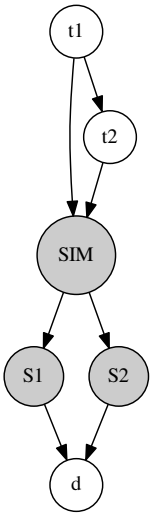

Figure 1: Example of an LFI model and running an ABC rejection sampling in ELFI. The observed data is given to the operation whose output should match it. Two summaries are defined, where the summary S2 has it's argument set to value 2.

large amounts of simulations rapidly, while BOLFI (Gutmann and Corander, 2016) aims to tackle cases with expensive simulators using sophisticated machine learning techniques.

Since likelihood-free inference often requires a moderate amount of experimentation (e.g. for choosing suitable summary statistics) it is important that specifying the generative model is made flexible and that already generated data can be reused. We found the DAG structure to be ideal for these tasks. First, ELFI allows any part of the DAG model (e.g. nodes and their dependencies) to be redefined keeping the rest of the structure intact. Second, ELFI provides automatic storing of the full output of any node (e.g. the simulator). These data will be automatically reused in further iterations of the model, potentially resulting in significant savings in compute time. These features are demonstrated in the ELFI tutorial in the documentation.

Another important factor is the ability to use non-Python operations in the LFI model. For instance, it may not always be practical or even possible to rewrite existing simulators in Python. ELFI provides both tools and examples in the documentation on how to use simulators written in other languages.

Other practical features include the ability to progress in the inference iteratively and to stop early if necessary. The provided visualization functions support assessing the current state of the inference. Finally, the model can be saved to a file and shared with others. ELFI also guarantees that the

results will be identical for the same seeds making the reproduction of the results easy.

2.2 Features for methodologists

For methodologists ELFI provides a convenient platform for testing new algorithms with models from the literature (e.g. Ricker, 1954; Marin et al., 2012; Lintusaari et al., 2017) and comparing their performance against existing algorithms. The framework provides means for parallelization, data storing, pseudo random number handling and other technicalities out of the box. The documentation includes instructions on how to implement new algorithms for ELFI. One of the major benefits is that all existing models will be usable with the new algorithms without modifications.

3 Performance and scalability

Performance is an important factor in computationally heavy inference such as LFI. ELFI uses batches of computations to control execution performance and parallelization. A batch consists of a fixed number of consecutive evaluations of a node in the model before moving to the next (e.g. 100 draws from the prior and then 100 simulations using those parameters). The standard parallelization strategy is to compute multiple batches in parallel. This provides several benefits. First, the computation of a single batch can often be vectorized with, for example, NumPy (van der Walt et al., 2011) for many of the basic operations (e.g. sampling from priors, computing distances), making these operations efficient in Python. Batches are also often relatively constant in their time and memory consumption, allowing flexibility in planning the parallel execution of multiple batches. This helps in avoiding unnecessary message passing, progressing the inference in meaningful steps, and makes it possible to know in advance the size of the returned output data for storing purposes. The algorithms in ELFI update their state after every completed batch.

4 Comparison to other similar software

There exist multiple LFI libraries for parameter inference. Many of them are either restricted to a specific problem domain (Liepe et al., 2014; Cornuet et al., 2014; Louppe et al., 2016), or require existing simulated data (Thornton, 2009; Csilléry et al., 2012; Nunes and Prangle, 2015). General purpose software similar to ELFI are, to our knowledge, ABCtoolbox (Wegmann et al., 2010), EasyABC (Jabot et al., 2013), and ABCpy[1] (unpublished). A

[1] https://github.com/eth-cscs/abcpy

Software	Language	Latest release	Data reuse	Parallelization	Separate modelling	Iterative processing
ABCtoolbox	C++	2009	Partial	cluster (manual)	✗	✗
EasyABC	R	2015	Partial	local	✗	✗
ABCpy	Python	2017	✗	local and cluster	✗	✗
ELFI	Python	2017	✓	local and cluster	✓	✓

Table 1: Comparison of general purpose LFI frameworks

relatively recent categorization of LFI software is provided by Nunes and Prangle (2015).

Among the general-purpose LFI software, ELFI is the only one that supports a standalone generative model and provides considerable flexibility in specifying dependencies in the model and modifications to the model (Table 1). For example, it is possible to embed multiple simulators into a single model without modifying their codes.

Regarding parallelization, EasyABC supports multiple cores on a single computer while the others can also run in cluster environments. By default, ABCpy uses Spark (Zaharia et al., 2010) and ELFI ipyparallel[2] for parallelization but both can be used with alternative backends. ABCtoolbox does not provide a parallel solution out of the box.

ABCtoolbox, EasyABC and ELFI support reusing generated data. ELFI is more flexible in that it allows the output of any node of the LFI model to be stored, and it automatically uses that data to compute the output of its current or future child nodes. There is thus no need to manually transform existing data.

ELFI is the only library that supports iterative advancing of the inference. Also, ELFI is currently the only general-purpose software to implement the BOLFI method (Gutmann and Corander, 2016), which can handle models with expensive-to-evaluate simulators outside the reach of other methods.

5 Source code and dependencies

ELFI has been designed to be open source and modular, and can be extended through interfaces. For instance, it is possible to add new types of operations, data stores or parallel clients. All the dependencies of ELFI are also open source.

ELFI is written in Python and is officially tested under Linux and MacOS but also works in Windows. The code style follows PEP 8 and documentation NumPy format. Code development uses the continuous integration practice with code review and automated tests to ensure the quality and usability of the software. The venue for distributing the source is GitHub[3] that among

[2]https://github.com/ipython/ipyparallel

[3]https://github.com/elfi-dev/elfi

the above features also allows anyone to raise issues regarding the software and make pull requests for new features. Online documentation is hosted in the Read The Docs [4]. ELFI also has a community chat for the users.

Acknowledgements

We would like to acknowledge support for this project from the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN).
We acknowledge the computational resources provided by the Aalto Science-IT project.

References

J. M. Cornuet, P. Pudlo, J. Veyssier, A. Dehne-Garcia, M. Gautier, R. Leblois, J. M. Marin, and A. Estoup. DIYABC v2.0: a software to make approximate Bayesian computation inferences about population history using single nucleotide polymorphism, DNA sequence and microsatellite data. *Bioinformatics*, 30(8):1187–1189, Apr 2014.

Katalin Csilléry, Olivier François, and Michael G. B. Blum. abc: an R package for approximate Bayesian computation (ABC). *Methods in Ecology and Evolution*, 3(3):475–479, 2012. ISSN 2041-210X. doi: 10.1111/j.2041-210X.2011.00179.x.

Michael U. Gutmann and Jukka Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016.

Franck Jabot, Thierry Faure, and Nicolas Dumoulin. EasyABC: performing efficient approximate Bayesian computation sampling schemes using r. *Methods in Ecology and Evolution*, 4(7):684–687, 2013. ISSN 2041-210X. doi: 10.1111/2041-210X.12050.

Antti Kangasrääsiö and Samuel Kaski. Inverse reinforcement learning from summary data. *arXiv preprint arXiv:1703.09700*, 2017.

Antti Kangasrääsiö, Kumaripaba Athukorala, Andrew Howes, Jukka Corander, Samuel Kaski, and Antti Oulasvirta. Inferring cognitive models from data using approximate Bayesian computation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1295–1306, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025576.

[4] http://elfi.readthedocs.io/en/stable/

J. Liepe, P. Kirk, S. Filippi, T. Toni, C. P. Barnes, and M. P. Stumpf. A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation. *Nat Protoc*, 9(2):439–456, Feb 2014.

Jarno Lintusaari, Michael U. Gutmann, Samuel Kaski, and Jukka Corander. On the identifiability of transmission dynamic models for infectious diseases. *Genetics*, 2016. ISSN 0016-6731. doi: 10.1534/genetics.115.180034.

Jarno Lintusaari, Michael U. Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Fundamentals and recent developments in approximate Bayesian computation. *Systematic Biology*, 66(1):e66, 2017. doi: 10.1093/sysbio/syw077.

Gilles Louppe, Kyle Cranmer, and Juan Pavez. carl: a likelihood-free inference toolbox, March 2016. URL `http://dx.doi.org/10.5281/zenodo.47798`.

Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. ISSN 0960-3174. doi: 10.1007/s11222-011-9288-2.

Matthew A. Nunes and Dennis Prangle. abctools: An R Package for Tuning Approximate Bayesian Computation Analyses. *The R Journal*, 7(2):189–205, 2015.

W. E. Ricker. Stock and recruitment. *Journal of the Fisheries Research Board of Canada*, 11(5):559–623, 1954. doi: 10.1139/f54-039.

Kevin R. Thornton. Automating approximate Bayesian computation by local linear regression. *BMC Genetics*, 10(1):35, 2009. ISSN 1471-2156. doi: 10.1186/1471-2156-10-35.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. doi: 10.1109/MCSE.2011.37.

Daniel Wegmann, Christoph Leuenberger, Samuel Neuenschwander, and Laurent Excoffier. Abctoolbox: a versatile toolkit for approximate Bayesian computations. *BMC Bioinformatics*, 11(1):116, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-116.

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.