The Journal of Open Source Software

HiddenMarkovModels.jl: generic, fast and reliable latent variable modeling

Guillaume Dalle [1]

1 EPFL (IdePHICS and INDY labs), Switzerland

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Submitted: 12 September 2023
Published: unpublished

Summary

Hidden Markov Models (or HMMs) are a very popular statistical framework, with numerous applications ranging from speech recognition to bioinformatics. They model a sequence of *observations* $Y_1, \dots, Y_T$ by assuming the existence of a hidden sequence of *states* $X_1, \dots, X_T$. The distribution of a state X_t can only depend on the previous state X_{t-1}, and the distribution of an observation Y_t can only depend on the current state X_t. This is a very versatile and practical set of assumptions: see Rabiner (1989) for an introduction and Cappé et al. (2005) for a book-length treatment.

Given a sequence of observations and a parametric family of HMMs $\mathbb{P}_\theta$, there are several problems one can face. In generic graphical models, these problems are often intractable, but HMMs have a tree-like structure that yields exact solution procedures with polynomial complexity.

Problem	Algorithm
Observation sequence likelihood $\mathbb{P}_\theta(Y_{1:T})$	Forward
State marginals $\mathbb{P}_\theta(X_t \vert Y_{1:T})$	Forward-backward
Best state sequence $\operatorname{argmax}_{X_{1:T}} \mathbb{P}_\theta(X_{1:T} \vert Y_{1:T})$	Viterbi
Maximum likelihood parameter $\operatorname{argmax}_\theta \mathbb{P}_\theta(Y_{1:T})$	Baum-Welch

The package `HiddenMarkovModels.jl` leverages the Julia language (Bezanson et al., 2017) to implement those algorithms in a *generic*, *fast* and *reliable* way.

Statement of need

The initial motivation for HiddenMarkovModels.jl was an application of HMMs to reliability analysis for the French railway company SNCF (Dalle, 2022). In this industrial use case, the observations were marked temporal point processes (sequences of timed events with structured metadata) generated by condition monitoring systems.

Unfortunately, the major implementations of HMMs we surveyed (in Julia and Python) all expect the observations to be generated by a *predefined set of distributions*. In Julia, the reference package `HMMBase.jl` (Mouchet, 2023) requires compliance with the `Distributions.jl` (Besançon et al., 2021) interface, which precludes anything not scalar- or array-valued. In Python, `hmmlearn` (hmmlearn, 2023) only offers Gaussians, mixtures of Gaussians and a few discrete distributions (categorical, multinomial, Poisson). Meanwhile, `pomegranate` (Schreiber, 2018) has a wider set of available distributions, but it doesn't allow for easy extension by the user.

Focusing on Julia specifically, other downsides of `HMMBase.jl` include the lack of support for multiple observation sequences, and the mandatory use of 64-bit floating point numbers. Two other packages provide functionalities that `HMMBase.jl` lacks: `HMMGradients.jl` (Antonello, 2021) contains a differentiable loglikelihood function, while `MarkovModels.jl` (Ondel et al., 2021) focuses on GPU acceleration. Unfortunately, all three have mutually incompatible APIs.

Package design

`HiddenMarkovModels.jl` was designed to overcome the limitations mentioned above, with the following guiding principles in mind.

It is *generic*. Observations can be arbitrary objects, and the associated distributions only need to implement two methods: a loglikelihood `logdensityof(dist, x)` and a sampler `rand(rng, x)`. The extendable `AbstractHMM` interface allows incorporating features such as priors or structured transition matrices. Number types are not restricted, and automatic differentiation of the sequence loglikelihood (Qin et al., 2000) is supported both in forward and reverse mode.

It is *fast*. Julia's blend of multiple dispatch and just-in-time compilation delivers satisfactory speed even when working with arbitrary observations. Inference routines rely on BLAS calls for linear algebra, and exploit multithreading to process sequences in parallel.

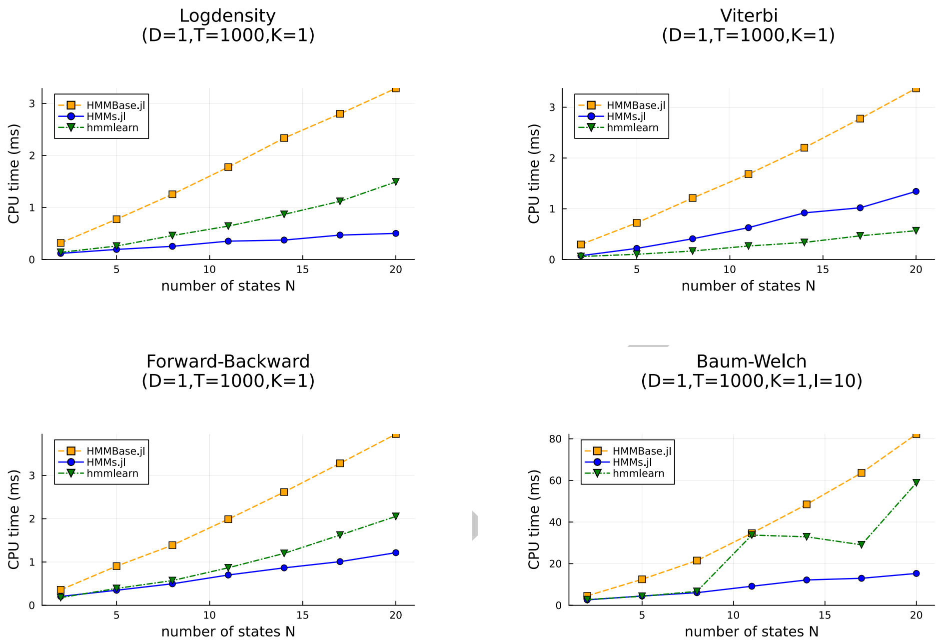
It is *reliable*. The package is thoroughly tested and documented, with an extensive API reference and accessible tutorials. Special care was given to code quality, type stability, and compatibility checks with various downstream packages.

As a consequence, it is also *limited in scope*. It centers around CPU efficiency, and remains untested on GPU. Its primary target is small- to medium-sized HMMs (a few tens of states), mostly because memory requirements scale quadratically for the chosen storage mode. Finally, it does not perform probability computations in the logarithmic domain, but instead uses the scaling trick from Rabiner (1989) with a clever variation borrowed from `HMMBase.jl`. Thus, its numerical stability might be lower than that of `hmmlearn` or `pomegranate` in challenging instances. However, thanks to unrestricted number types, users are free to bring in third-party packages like `LogarithmicNumbers.jl` (Rowley, 2023) for additional precision.

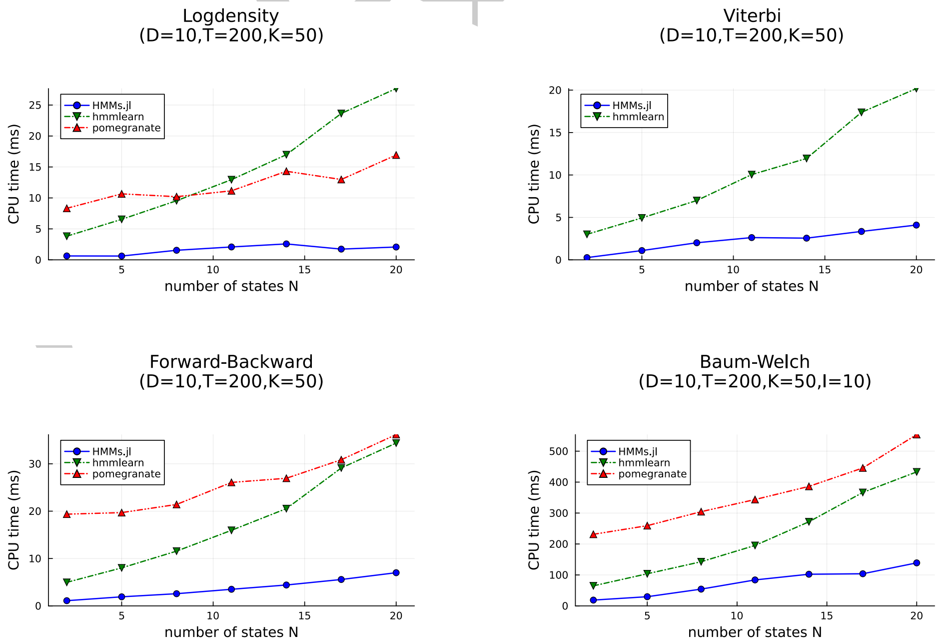
Benchmarks

We compare `HiddenMarkovModels.jl` (abbreviated to `HMMs.jl`), `HMMBase.jl`, `hmmlearn` and `pomegranate` on a test case with multivariate Gaussian observations. The relevant parameters are the number of states `N`, the sequence duration `T`, the observation dimension `D`, the number of sequences `K` and the number of Baum-Welch iterations `I`. This benchmarking code is run automatically by GitHub Actions upon each package release, but here we ran it on a local Linux system with more cores. The numbers of Julia, OpenBLAS (for NumPy) and MKL (for PyTorch) threads were all set to 6, although it is hard to compare all libraries fairly when it comes to parallel computing. See the package documentation for more details on the benchmarks.

In a low-dimensional scenario, `HiddenMarkovModels.jl` runs substantially faster than its predecessor `HMMBase.jl`, even though their algorithms are mathematically identical. We note that performance is less convincing for the Viterbi algorithm, in which linear operations are replaced by max-plus operations (less easy to optimize). `pomegranate` is excluded because it does not support multiple sequences.



In a high-dimensional scenario, `HiddenMarkovModels.jl` scales favorably compared to both Python alternatives, even though the latter make use of highly optimized C++ backends. This can partly be explained by the absence of logarithmic computations. `HMMBase.jl` is excluded because it does not support multiple sequences.



Conclusion

`HiddenMarkovModels.jl` fills a longstanding gap in the Julia package ecosystem, and might even prove interesting for Python users. Future research directions include the implementation of Input-Output HMMs (Bengio & Frasconi, 1994) as well as other estimation methods (gradient-based or spectral).

Acknowledgements

A special thank you goes to Maxime Mouchet and Jacob Schreiber, the developers of `HMMBase.jl` and `pomegranate` respectively, for their help and advice. In particular, Maxime agreed to designate `HiddenMarkovModels.jl` as the official successor to `HMMBase.jl`, which I am very grateful for.

References

Antonello, N. (2021). *HMMGradients.jl: Enables computing the gradient of the parameters of Hidden Markov Models (HMMs)*. Zenodo. https://doi.org/10.5281/zenodo.4454565

Bengio, Y., & Frasconi, P. (1994). An Input Output HMM Architecture. *Advances in Neural Information Processing Systems*, *7*. https://proceedings.neurips.cc/paper/1994/hash/8065d07da4a77621450aa84fee5656d9-Abstract.html

Besançon, M., Papamarkou, T., Anthoff, D., Arslan, A., Byrne, S., Lin, D., & Pearson, J. (2021). Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem. *Journal of Statistical Software*, *98*, 1–30. https://doi.org/10.18637/jss.v098.i16

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Cappé, O., Moulines, E., & Rydén, T. (2005). *Inference in Hidden Markov Models*. Springer New York. https://doi.org/10.1007/0-387-28982-8

Dalle, G. (2022). *Machine learning and combinatorial optimization algorithms, with applications to railway planning* [PhD thesis, École des Ponts ParisTech]. https://www.theses.fr/2022ENPC0047

hmmlearn. (2023). *Hmmlearn: Hidden Markov Models in Python, with scikit-learn like API*. hmmlearn. https://github.com/hmmlearn/hmmlearn

Mouchet, M. (2023). *HMMBase.jl: Hidden Markov Models for Julia*. https://github.com/maxmouchet/HMMBase.jl

Ondel, L., Lam-Yee-Mui, L.-M., Kocour, M., Filippo, C., & Lukás Burget, C. (2021). *GPU-Accelerated Forward-Backward Algorithm with Application to Lattic-Free MMI*. https://hal.science/hal-03434552

Qin, F., Auerbach, A., & Sachs, F. (2000). A Direct Optimization Approach to Hidden Markov Modeling for Single Channel Kinetics. *Biophysical Journal*, *79*(4), 1915–1927. https://doi.org/10.1016/S0006-3495(00)76441-1

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. https://doi.org/cswph2

Rowley, C. (2023). *LogarithmicNumbers.jl: A logarithmic number system for Julia*. https://github.com/cjdoris/LogarithmicNumbers.jl

Schreiber, J. (2018). Pomegranate: Fast and Flexible Probabilistic Modeling in Python. *Journal of Machine Learning Research*, *18*(164), 1–6. http://jmlr.org/papers/v18/17-636.html