HaMMLET(1)

NAME

HaMMLET - Fast Bayesian Inference for Hidden Markov Models using Dynamic Haar Wavelet Compression.

DESCRIPTION

HaMMLET is a fast Forward-Backward Gibbs (FBG) sampler for Bayesian HMM. It also implements alternative sampling schemes (currently supported: Mixture Model sampling). Given numerical input data and prior parameters, it outputs a full distribution of latent HMM states for each position, integrating over the entire parameter space. In modern applications, such as the detection of copy-number variants (CNV) using whole-genome sequencing data, the input sizes are on the order of millions to billions of data points. To avoid prohibitively long running times and slow convergence, HaMMLET uses the Haar wavelet transform to dynamically compress the data into blocks of sufficient statistics, based on the lowest noise estimate in each iteration of the Gibbs sampler.

When using HaMMLET, please cite the following paper:

> Wiedenhoeft, J., Brugel, E., & Schliep, A. (2016). "Fast Bayesian Inference of Copy Number Variants using Hidden Markov Models with Wavelet Compression". PLOS Computational Biology, 12(5), e1004871. http://doi.org/10.1371/journal.pcbi.1004871. This paper was selected for oral presentation at RECOMB 2016.

USAGE EXAMPLE

hammlet -f data.csv -o hmm/result_ .csv -O blocks compression -s C 3 2 -i M 100 0 F 200 5 -a -t 10 1

This reads data from **data.csv**, and writes state marginals to **hmm/result_**marginals**.csv** (notice the space, -o takes two arguments!). It also outputs the sizes of **blocks** for each iteration to hmm/result_**blocks**.csv, and **compression** ratios to hmm/result_**compression**.csv. There are 9 hidden states in total, resulting from all possible **combinations** of **3** emission distributions to generating **2**-dimensional data. Sampling of state sequences is done by first running **mixture sampling** for **100** iterations, **none** of which is recorded, followed by **FBG** for **200** iterations, recording every **fifth** of them. This is done using **automatic** emission priors with standard parameters. Prior **transition** weights are **10** for self-transitions, and **1** for transition between different states.

OPTIONS

-h | -help Display a friendly help message.
-v | -verbose Print information to *STDOUT* during run-time.
-g | -arguments Print arguments. For each flag, print an asterisk if it was set by the user, as well as the parameters being used. If the flag was not set, these are the default parameters.

INPUT AND OUTPUT

HaMMLET reads the observed emission values as a stream of whitespace-separated numeric values from *STDIN*, unless **-f** is specified. Linebreaks are treated as whitespace, and no formatting such as CSV is interpreted. For multivariate models, the dimensions of each position are filled in increasing order, after which dimensions for the next position are filled. If the number of input values is not a multiple of the number of dimensions (see **-s** option), an exception is thrown.

The output consists of a CSV file representing a run-length encoded version of the state marginals. The first column represents the length of a segment (number of input positions), and subsequent columns represent the recorded counts for each state, in increasing order of state number.

-f *FILE* | -input-file *FILE* Read input data from *FILE* instead of *STDIN*.

-o *PREFIX* | -output-prefix *PREFIX* The prefix for the output file paths. *PREFIX* is extended by a short description and an appropriate file extension to yield an output file name, such as *PREFIX*marginals.csv for the file containing the marginal state distribution; for additional files, see the **-O** option for details. Existing files are silently overwritten. [Default: **hammlet**]

-O *TYPE ...* | -output-data *TYPE ...* Specify a list of data types to be output in addition to the marginals. This only applies to recorded iterations as specified using **-i**. It may contain any of the following:

- **S | sequences** Output each state sequence individually, one per line, separated by whitespace, using run-length encoding of the form *LENGTH*:*STATE*.
- **P | parameters** Output the emission parameters for each state in increasing order of state number, separated by tabs.
- **B | blocks** Output the block structure (sizes of all blocks) created by dynamic compression, separated by whitespace.
- **C | compression** Output the compression ratio for each iteration.
- **G | segments** Output the number of segments in each iteration, as well as the number of values used to store the compressed marginals (for diagntic purposes).

-w | -overwrite Overwrite existing output files. If **-w** is not provided and an output file already exists, an exception is thrown.

MODEL SPECIFICATIONS

-s *PARAM* | -states *PARAM* Definition of hidden states. *PARAM* may take the following forms [Default: **3**]:

NRSTATES The typical, simple case of an HMM: Given a single unsigned integer *NRSTATES*, the data is assumed to be one-dimensional and generated from this many hidden states.

***MAPPING NRPARAM* [*NRDIM*]** State definitions in the form of a *MAPPING*. The data has *NRDIM* dimensions, and there are *NRPARAM* different emission distributions. If *NRDIM* is not provided, it defaults to **1**. A state is defined as a vector S of size *NRDIM*, where S[i] denotes that the i-th dimension of an emission sampled from this state was generated from the S[i]-th emission distribution. For instance, if a 3-dimensional data point is assigned to a state with mapping S=[1 1 2], its first two dimensions are assumed to be generated by emission distribution 1, and its third dimension by emission distribution 2. The following *MAPPING* schemes are supported:

C | combination States express all possible combinations of emission distributions, resulting in *NRPARAMS^NRDIM* different states. For instance, **-s C 2 3** generates 2^3=8 state mappings: (1 1 1), (1 1 2), (1 2 1), (1 2 2), (2 1 1), (2 1 2), (2 2 1), and (2 2 2). Note that **-s C 3 1** is equivalent to **-s 3**.

NOTE: No other mapping schemes are currently supported.

-e *DIST PARAM* | -emissions *DIST PARAM* Set the emissions to be variates of a given *DIST*ribution, and let their parameters be sampled from priors using the given hyper*PARAM*eters. The behavior of this option depends on the number of tokens: Let K be the number of hyperparameters per prior, and D the number of emission distributions (see **-s** option). If *PARAM* consists of K tokens, all priors are assumed to have those same hyperparameters. If there are N*K tokens, each prior gets its specific set of hyperparameters. In all other cases, an exception is thrown. Arguments may take the following forms [Default: **normal**. This means that **-a** has to be provided if **-p** is not.]:

normal [*PARAMs*] For Normal emissions, *PARAM* is a collection of 4-tuples *ALPHA BETA MU NU*, representing parameters to the

Normal-Inverse Gamma distribution, sorted by state. If **-a** is set, *PARAM* is *VAR P* instead, where *P* is the probability to sample emission variances less or equal than *VAR*; if these parameters are not provided, they default to **0.2 0.9**.

NOTE: No other emission type than Normal is currently supported.

If neither **-a** nor *PARAM* is provided, an exception is thrown.

-a | -auto-priors Use automatic hyperparameters for emission priors, based on the wavelet transform of the data. This changes the meaning of parameters passed to **-p**.

-t *VALUES* | -transitions *VALUES* Parameters for transition probabilities. These are the parameters alpha for a Dirichlet distribution. *VALUES* can take the following forms:

ALPHA A single number means that all alpha-parameters are set to the same value.

SELF TRANS All alphas corresponding to self-transitions are set to *SELF*, the others to *TRANS*.

-S | -no-self-transitions Do not use self-transition probabilities within blocks (this has no effect for mixture sampling).

-I *ALPHA* | -initial *ALPHA* Sets the alpha parameter of the Dirichlet distribution used as a prior for the initial state distribution.

SAMPLING SCHEME

-R | -random-seed An unsigned integer value to be used to seed the random number generator. If **-R** is not set, a seed is generated from the current epoch time. A seed should be set manually using **-R** whenever reproducibility is required.

-i *SCHEME ...* | -iterations *SCHEME ...* A list of sampling *SCHEME*s, each of which consists of three tokens, *TYPE ITER THIN*:

1. The *TYPE* of sampling method to be used is one of the following:

   F *Forward-Backward Gibbs sampling* uses a dynamic programming trellis to quickly sample state sequences unaffected by autocorrelation due to adjacent blocks. FBG is considered the state-of-the-art for Gibbs sampling in HMM. Running times depends quadratically on the number of states.

   M *Mixture sampling* treats compression as a way to impose equality relations on otherwise exchangeable data points. It completely ignores transition probabilities passed to the model, and instead assumes transitions to be implied in the block structure alone. This is much faster than the other methods, as it depends linearly on the number of states, but is not truly an HMM. High-variance components are prone to oversegmentation, and spurious differences in sampled values can lead to segments which come from the same true state being assigned to different states. However, if the variance is expected to be similar over all states, this variant can yield reasonably good results very fast.

2. The number of sampling *ITER*ations.

3. The type of *THIN*ning to be used to record sampled state sequences (0=record none, 1=record all, 2=record every second sample, etc.).

If the total number of tokens provided to **-i** is not a multiple of 3, an exception is thrown. [Default: **M 100 0 F 250 10**, i.e. an unrecorded burn-in of 100 mixture iterations, followed by 250 FBG iterations with thinning factor 10, resulting in 25 recorded state sequences.]

COMPRESSION

-b *MIN MAX* | -block-limits *MIN MAX* The minimum and maximum block size allowed during compression. 0 means no limit. [Default: **0 0**]

-m *FLOAT* | -weight-multiplier *FLOAT* Multiply weights by this factor, to avoid overcompression. [Default: **1.0**]

CAVEATS

While HaMMLET is designed to minimize memory consumption (univariate models of 100 million data points can be handled on a standard laptop), one

should still be aware that the size of the marginal state records and the trellis cannot be predicted before running the inference. As a consequence, data that only allows for low compression ratios may still incur huge memory overhead, as it negates the central approach that makes FBG feasible on such scales. If memory consumption gets out of hand, you might want to try increasing the number of burn-in steps; if the sampler has not fully converged, individual iterations might have very low compression, even though the data itself would allow for better ratios. Likewise, decreasing the number of states might be an option, since superfluous state parameters will be sampled solely from the prior and yield arbitrarily low noise variances. If this does not work, using Mixture model sampling might be an option, but results should be interpreted with care, see **-i** option.

Though the model should work for any emission distribution in the exponential family (Normal, Poisson, Exponential, Laplace, Gamma, Chi-Squared etc.), only Normal emissions are implemented at the moment.

Multivariate models are supported in the sense that multiple data dimensions may share their generating parameters. True multivariate models such as Normals with non-diagonal covariance matrix are not yet supported.

Plotting the results is done using external Python libraries (NumPy, Matplotlib). As these are not optimized for large-scale applications, this can take a long time, often longer than the inference itself.

HaMMLET does not support the convention of combining single-letter options, such as replacing **-x -y -z** by **-xyz**.

HISTORY

The first version of HaMMLET was developed by Eric Brugel and John Wiedenhoeft, and published in 2016 in PLOS CompBio and RECOMB. It used a wavelet tree data structure for dynamic compression. The current version is designed for minimal memory footprint in large-scale applications. Changes include: a breakpoint array data structure for optimal wavelet compression, an in-place algorithm for its construction, run-length-encoded output, and a queue-based implementation to record run-length-encoded state sequences. It is currently developed and maintained by John Wiedenhoeft (ORCID: 0000-0002-6935-1517) at https://github.com/wiedenhoeft/HaMMLET.

REPORTING BUGS

GitHub issue tracking system: https://github.com/wiedenhoeft/HaMMLET/issues

SEE ALSO

Current hosting site: https://wiedenhoeft.github.io/HaMMLET/

Current repository: https://github.com/wiedenhoeft/HaMMLET

Stable link: https://schlieplab.org/Software/HaMMLET/

Documentation in different formats (pdf, html, txt, man) can be found in the doc/ subfolder of HaMMLET's installation directory.