Rules and associations : hidden Markov models and neural networks in the psychology of learning

Visser, I.

Publication date
2002

B Manual of Markovfit 1.6

A program for fitting and selecting hidden Markovmodels for categorical time series data

Ingmar Visser
Unpublished manuscript
Copies and source codes available from ingmar@dds.nl

Abstract

`Markovfit 1.6` implements the EM-algorithm for maximum likelihood estimation of hidden Markov model parameters. It can also be used to estimate or compute parameters for standard Markov models with data from multiple subjects. It also computes confidence intervals for (hidden) Markov model parameters using either likelihood profiling or bootstrapping. In addition separate routines are available for generating data according to a specified (hidden) Markov model and for explorative search for models. The hidden Markov model is suitable for modeling sequences of categorical responses or single categorical timeseries. Both the number of categories and the length of the sequences are unlimited in principle.
Keywords: hidden Markov model; Markov model; EM algorithm; maximum likelihood estimation; confidence intervals; likelihood profiles; bootstrapping; data generation; error computation; fitmeasures; categorical data; categorical timeseries.

B.1 The hidden Markov model

Hidden Markov models (HMM) are mainly used in the area of speech recognition (c.f. Schmidbauer et al., 1993; Chien and Wang, 1997; Rabiner, 1989), but have also found their way into psychological modeling, for example in coding of recall data (Durbin et al., 2000). Other applications are in physiology (Becker et al., 1994) and in computational genetics (Krogh, 1998).

HMMs are stochastic finite state machines but they come in many different guises and have many different designations: latent Markov models, (analytically tractable) Boltzmann machines, (analytically tractable) belief networks and (analytically tractable) state space models to name but a few.

An HMM consists of a Markov transition process, the hidden part of the model, and a observation process. The hidden process is a, not necessarily stationary, Markov process over a finite number of states. The observation process is a function that determines the observation given the hidden (or latent) state of the process.

Formally an HMM consists of the following elements (notation from Rabiner, 1989):

1. a set of states S_i, $i = 1, \ldots, N$
2. a set V of observation symbols V_k, $k = 1, \ldots, M$
3. a matrix A of transition probabilities a_{ij} for moving from state S_i to state S_j
4. a matrix B of observation probabilities $b_j(k)$ of observing symbol V_k while being in state S_j
5. a vector π of initial state probabilities π_i corresponding to the probability of starting in state S_i at $t = 1$

The equations describing the dynamics of the model are as follows:

$$\begin{aligned} S_{t+1} &= A\,S_t + \zeta_{t+1} \\ O_{t+1} &= B\,S_t + \xi_{t+1}, \end{aligned}$$

where S_t is the hidden (Markov) process and O_t is the observed process; ζ_{t+1} and ξ_{t+1} are zero mean martingale increment processes (c.f. Elliott et al., 1995, p. 20 for further details).

An HMM becomes a normal Markov model when the response process is known and fixed. Some authors add the requirement that the observation process is a deterministic function. In terms of the above described model, this means that the observation matrix B is known and fixed and in fact is the identity matrix. Often, in ordinary Markov modeling the initial state distribution is also known and fixed.

B.1.1 Markov models versus hidden Markov models

In the literature on Markov models the terminology is somewhat different from that in the hidden Markov literature. In particular, the observation function (or observation matrix) in hidden Markov models is referred to as the response function in Markov models. The state space for which the (hidden) Markov process is defined is sometimes referred to as the knowledge state space in literature on Markov models. The response function maps the (knowledge) state space onto a response state space, i.e. the possible responses. In a large part of the literature on Markov models the response state space consists of two states: a correct and an incorrect response. This limitation to two possible responses, however, is not a principled limitation; in general, any finite number of categorical responses is allowed. In the remainder of this manual the transition function is referred to as the `transition matrix` and the response function is referred to as the `observation matrix`.

B.2 Running `Markovfit 1.6`

In order to fit (hidden) Markov models on a dataset, two things are needed: a datafile and a command or syntax file. `Markovfit` does not have a graphical user interface. All configurations and data specifications have to be fed to the program in syntax file that from here on will be called the `command file`. In the following sections the requirements for both the data file and the command file are described.

B.2.1 Data file format

The `data file` should contain the data and, importantly, nothing else. Data that can be fitted with `Markovfit` are single categorical timeseries or multiple sequences of categorical responses. For both these types of data it is required that the possible response categories are numbered from 1 through M where M is the number of possible response categories. Note in particular that 0 and 1 response categories for incorrect/correct responses are not recognized by the program so these have to be recoded to 1 and 2 respectively. All response categories have to occur in the data. In particular, it is not possible to have response categories 1, 2 and 4 but not 3. In this case response category 4 should be recoded to response category 3.

In the case of a single timeseries the responses have to be written in the data file in order of recording them, separated by white space. White space can be either spaces, tabs, newlines or returns. The specific type of white space does not matter. In the case of multiple sequences of responses, for example sequences generated by different subjects, the responses should be listed in order separated by white space. For the program it is not necessary to separate the different sequences of responses by returns or other types of white space. For example, the `data file` of a single timeseries of 1000 responses with 4 categories may look exactly the same as the `data file` of 10 sequences of 100 responses with the same number of response categories. However, for purposes of readability it is of course advisable to have a clear separation between different sequences of responses.

Note that, although all response categories have to occur in the data, it is not necessary that, when fitting multiples sequences, each sequence should contain each response category.

B.2.2 Command file format

When running the program it requests the name of the `command file`. After typing this name and hitting return, the `command file` is read, checked and executed. The `command file` has four main sections: the type of analysis the program has to perform, a description of the data, a description of the model and a specification of the desired outputs. All the commands can appear in any order in the `command file` although it is advisable to stick to the above mentioned order to prevent errors and to simplify finding them if any occur. The `command file` is ended with the command `end`. It tells the program to stop reading the `command file`. All the files that are read by the program have to be in the same directory/folder as the program itself.

Comments

After the `end`-line any amount or form of commentary is allowed; it will be ignored. It is allowed to write commentary anywhere in the `command file`. However, care should be taken that a comment line does not start with any of the commands that are used by the program. To prevent this, it is recommended that comment lines start with , the usual C++ keyword for end-of-line-comments but anything is allowed as long as it does not start with the name of a command used in `Markovfit`.

Command line format

Each (non-comment) line of the `command file` has the following format: `<command name>` followed by the `<command value>`. The `<command name>` and the `<command value>` should be separated by one or more spaces or tabs but they should be on the same line. There are a few exceptions to this, notably the `strings`-command and the `pa`-command that are described below.

The `job`-command

There are three types of analyses possible with `Markovfit`. The type of analysis required is specified with the `command name job` followed by one of the following three `command values`: `fitting`, `explore` or `generate`. Here's a description for each `job`:

`fitting` This routine fits an HMM with a fixed number of (hidden) states to a given data set. Optional outputs include a number of measures of fitness and confidence intervals for the parameters. Optional starting values for the optimization can be specified along with a number of parameters that control optimization.

`explore` This routine fits a series of HMMs with a increasing number of hidden states to a given data set. This should be used when no a priori structure of the model is available or desired. The output of this routine is a list of the fitted models and a number of measures of fitness for each of them in order to facilitate model selection.

`batch` This routine does the same thing as the explore routine for a number of different datasets.

`generate` This routine generates a data set of specified dimensions according to a given model.

In the following three sections the required and optional commands for each of these `jobs` will be listed along with an example `command file`.

B.2.3 Standard run: `fitting`

To fit an HMM the `command file` should have the line: `job fitting`. Apart from that it has to contain data description commands and model description commands and, optionally, output specifications.

Table B.1: Data description commands

Command	Status	Possible values	Description/use
`datafile`	obligatory	`<filename>`	specifies the file that contains the data
`categories`	obligatory	positive integer	the number of response categories in the data
`datapoints`	optional	positive integer	number of datapoints of a single timeseries
`strings`	optional	positive integer	number of separate sequences/strings

Data description commands

Depending on the type of data there are the following data description commands.

Either one of the `datapoints` or the `strings` is required to describe the data. If a number of individual sequences is to be fitted the `strings`-command is used. After the command line that has `strings <number of strings>` the lengths of each of the strings or sequences have to be listed, separated by white space.

Model description commands

These commands define the model to be fitted.

Table B.2: Model description commands

Command	Status	Possible values	Description/use
`states`	obligatory	positive integer	the number of (hidden) states of the HMM
`modelfile`	optional	HMM parameters	starting values for optimization can be provided here, see details below
`pa`	optional	see details below	with this command values for parameters that should be fixed or constrained to be equal to another parameter can be provided

Starting values: ***`modelfile`*** The file that contains the starting values should list the number of states, the number of observation categories (both positive integers) and then the three model matrices, `transition matrix`, `observation matrix` and `initial state distribution` vector, in this order. See section B.1 for the definition of these parameters.

Fixing parameters: ***pa*** When parameters have to be fixed on certain values this can be done with the pa. Also parameters that should be estimated equal can be set using this command. After the pa-command, on the next lines, the three model matrices should be listed with zeroes for the parameters that should be fixed and with non-zero integer values for the parameters that are to be estimated free. If parameters need to be estimated equal they should be given the same positive integer value in this listing of parameters. For more on the order of the parameters, see the section about that. When parameter should be fixed, the value of that parameter that is read from the `modelfile` is the value at which it will be fixed. The other values in the `modelfile` are used as starting values.

Output specifications

The output of the `fitting`-routine is written to a file called `<datafile>.out`, in `fitting`. Outputs are appended to this file if it already exists. Outputs that are always given are the following (in that order in the output file):

1. Starting values of the model (parameters in the same order as described above for the `modelfile`).
2. Number of datapoints or number of strings and their lengths.
3. Fitted model (parameters in the same order as described for the `modelfile`).
4. Number of iterations until convergence.
5. Loglikelihood of the data given the model.
6. Number of parameters of the model.
7. Number of free parameters: number of parameters minus the number of constraints imposed. This is the number of degrees of freedom in likelihood ratio χ^2-tests. It is also used for computing the AIC and BIC measures of fitness.
8. AIC
9. BIC
10. Adjusted free parameters: same as the number of free parameters minus the number of parameters that is estimated at zero (in general for large models many parameters are estimated zero and hence the adjusted number of parameters may be quite different from the number of free parameters).
11. Adjusted AIC, same as AIC but using adjusted free parameters instead of the free parameters.
12. Adjusted BIC, same as BIC but using adjusted free parameters instead of the free parameters.
13. Lempel-Ziv entropy of the data.

14. Lempel-Ziv entropy of the model (based on ten resampled datasets of the same dimensions as used for fitting the model) (not yet).

All the following output specification commands are optional. These outputs, if requested, appear in the order specified here, except for the most likely state sequence which is written to a separate file.

Table B.3: Output specification commands

Command	Possible values	Default	Description use
`profile`	0/1	0	1 for profile likelihood confidence intervals (this is not implemented)
`boots`	0/1	0	1 for bootstrapped confidence intervals
`samples`	positive integer	100	this is the number of bootstrap samples used to compute bootstrapped confidence intervals
`errors`	0/1	0	1 for computing χ^2 prediction error measures
`viterbi`	0/1	0	1 for computing the most likely state sequence
`mod`	0/1	0	put a 1 for loglikelihood ratio modification indices

Likelihood profiles ***profile*** Likelihood profile based confidence intervals are not implemented in this version.

Bootstrapping: ***boots*** When the `boots`-command is given value 1 confidence intervals will be bootstrapped. Outputs that are then produced are the following: the value of the parameter, the mean of the fitted values of that particular parameter, the standard error, i.e. the standard deviation of the distribution from the bootstrap samples, the confidence interval, i.e. $1.96 \times stderr$ and the t-ratio of the parameter. The t-ratio is an indication if the parameter value is significant. The t-ratio given here is the parameter value divided by its standard error. As a rule of thumb it should be more then 2 for a parameter value to be significant. Also see the paragraph on modification indices. All the bootstrap samples are written to a file called `bootstrapsamples`.

Bootstrapping: ***samples*** When bootstrapped confidence intervals are requested with the `boots`-command, it is possible to set the number of bootstrap samples using the `samples`-command. Usually one hundred bootstrap samples gives results that are precise enough for practical purposes but more samples can be requested.

Prediction errors: ***errors*** By setting the `errors`-command to 1 prediction errors will be listed in the output file. These prediction errors are χ^2 differences between observed and predicted frequencies of single observation symbols, pairs of observation symbols, triples et cetera. Prediction errors can only be computed for ergodic models or models without absorbing states. This is checked by the program and error computation is skipped if this is the case.

Most likely state sequence: ***viterbi*** For timeseries it is often useful to know the most likely state sequence, that is, the fitted sequence of hidden states that optimizes the loglikelihood. It can be requested by putting `viterbi 1` in the **`command file`**. It can also be used for multiple sequences. The result is written to a file called `<datafile>.vit`.

Modification indices: ***mod*** When the `mod` is set to 1, modification indices are computed for the parameters. The modification index is the loglikelihood ratio of the fitted model and the constrained model that is construed by setting a parameter to zero. The loglikelihood ratio that is reported is:

$$R = -2 \times (L_c - L_f),$$

where L_c is the loglikelihood for the constrained (with a specific parameter set to zero), and L_f is the fitted model. Equality constraints are incorporated in computing R, that is, when two parameters are constrained to be equal, they will both be set to zero for computing their modification index.

Modification indices are not in general available for all parameters. First, they are not computed for fixed parameters. Second, setting some parameter to zero might lead to an inadmissible model, i.e the data can not be described anymore with the data. Third, sometimes setting a parameter to zero leads to a model that is either not identified or not ergodic anymore, although the model may be an admissible model. In theory this would lead to a loglikelihood of minus infinity. When this is the case this is reported in the output file and the loglikelihood ratio is reported as *nan* (not-a-number).

Order of parameters

In two output routines the parameters are numbered consecutively in a special order. These are the bootstrap and modification indices routines. The parameters are numbered as follows: first the transition matrix parameters, for example, 1 through 4 in case of two-state model. After that the observation matrix parameters are numbered and then the initial state distribution. For a two-state, three-category model, parameter number 9 is thus the probability of observing a symbol of category 2 in (hidden) state 2.

Sample ***command file***

This is a sample **`command file`** using most of the commands described above. Note that the line numbers are not part of the actual **`command file`**.

```
1  title testing Markovfit 1.6
2  job fitting
3
4  //data description
5  datafile testmodel.dat
6  strings 10
7  100 100 100 100 100   100 100 100 100 100
8  categories 3
9
10 //model description
11 states 2
12 modelfile testmodel
13
14 pa
15 1 2
16 3 4
17
18 5 0 6
19 7 8 5
20
21 0
22 0
23
24 //requested outputs
25 boots 1
26 errors 0
27 end
28 any comments are allowed here
```

Here's a description of each line and what it results in:

1. This line is ignored since title is not a keyword used in `Markovfit 1.6`
2. This line specifies the `job` to be performed, in this case simply fitting a model.
3. Empty lines have no effect whatsoever but they do make the `command file` more readable.
4. This is a comment line for purposes of clarity.
5. Name of the file that the data has to be read from.
6. This line indicates that the data consists of 10 separate strings of observations.
7. This line lists the lengths of each of these 10 separate strings, in this case they all have the same length but this is not necessary.
8. This line gives the number of categories in the data, alternatively called the number of observation symbols.

9. Empty lines have no effect whatsoever but they do make the `command file` more readable.

10. Comment line (not starting with a `command` or keyword.

11. The number of states in the model.

12. The name of the file with the starting values of the model.

13. Empty line ...

14. The `pa`-command indicates that some parameters of the model should be estimated to be equal and/or fixed

15. This line and the next are the transition parameters, they are all estimated free.

16. They have no equality constraints imposed on them.

17. Empty line ...

18. This line and the next tell `Markovfit` that parameters b_{11} and b_{23} should be estimated equal.

19. Parameter number 6, b_{12} should be fixed.

20. Empty line ...

21. This line and the next indicate that the initial state probabilities have no equality constraints imposed on them.

22. ... and both are fixed.

23. Empty line ...

24. Comment line ...

25. The `boots`-command indicates that bootstrapped confidence intervals are requested (with the default number of samples since samples is not in the command file).

26. The `errors`-command indicates that no prediction errors are requested. Since this is the default this line is actually superfluous.

27. The `end` tells `Markovfit` to stop reading the command file here.

28. After the `end`-command the `command file` is not read anymore so any comments are allowed.

*B.2.4 Explorative fitting of models: **explore***

In order to exploratively fit HMMs the `command file` should have: `job explore`. Apart from that it has to contain data description commands and model description commands and, optionally, output specifications. The `explore`-routine will fit a series of HMMs according to specifications given in the command file. Usually this means fitting a series of models with an increasing number of hidden states. For each number of hidden states this can be done a number of times, which is recommendable since not always the same models are found given different starting values. See section B.2.4 below on starting values.

Data description commands

Data description is the same as it is when using the normal `fitting`-routine. Refer to section B.2.3 for a complete description of the required commands.

Model description commands

These commands define the models to be fitted. When exploratively fitting HMMs it is required to specify the range of model to be fitted.

Table B.4: Model description commands in `explore`

Command	Status	Possible values	Description/use
`statbegin`	optional	positive integer	see details below
`statend`	obligatory	positive integer	see details below
`fitsPerState`	optional	positive integer	see details below

*Number of states: **statbegin** and **statend*** With the `statbegin` the number of hidden states to begin exploring is specified. It is an optional command. When it's left out of the `command`-file, exploring is started with an HMM with one hidden state. Exploring stops with `statend` number of hidden states. For each number of hidden states, `fitsPerState` number of models are fitted, with the exception of the single hidden state model, since the single hidden state model is uniquely identified. It is in fact the model where the entries of the `observation matrix` correspond to the proportion of each observed category. Although it is a trivial model, it is good to have it as a reference model in model selection.

Starting values In explorative fitting of models, starting values can not be provided by the user. For each model to be fitted ten sets of random starting values are generated. Each of these sets of starting values are iterated through the EM-algorithm `iterStart` times; see section B.2.7 below for a complete description of this command.

Fixing parameters Fixing parameters is not possible in routine `explore` for obvious reasons.

Output specifications

Output options in the `explore`-routine are the same as in the `fitting`-routine; they are, however, written to different file called `<datafile>.models`. Outputs are appended to this file if it already exists. Before each model's outputs the numer of the fitted model is given. That is, models that are fitted are numbered consecutively starting with the models with the least number of states. Outputs that are always given are the same ones as in the `fitting`-routine, see section B.2.3.

All the following output specification commands are optional. These outputs, if requested, appear in the order specified here. The only difference with outputs in the `fitting`-routine is that the most likely state sequence can not be requested. All others can be but it should be noted that all of them, except the modification indices, are computationally demanding. When fitting a long series of models it is recommendable to not specify any of these outputs to save on computation time. For detailed descriptions of what all these commands amount, refer to the respective paragraphs in section B.2.3.

Table B.5: Output specification commands

Command	Possible values	Default	Description use
`profile`	0/1	0	1 for profile likelihood confidence intervals (this is not implemented)
`boots`	0/1	0	1 for bootstrapped confidence intervals
`samples`	positive integer	100	this is the number of bootstrap samples used to compute bootstrapped confidence intervals
`errors`	0/1	0	1 for computing χ^2 prediction error measures
`mod`	0/1	0	put a 1 for loglikelihood ratio modification indices

Apart from the models file, another output file is created in the `explore`-routine which has all the information needed for model selection. This file is called `<datafile>.fits`. Outputs are appended to this file if it already exists. The has file contains one row for each fitted model. Each row has the following information:

exp nr The number of the model which is the same as in the output file that contains the fitted models.

it The number of iterations needed until convergence.

states The number of (hidden) states of the model.

pol The loglikelihood of the data given the model.

AIC Akaike's Information Criterion.

A-AIC Adjusted AIC, see section B.2.3.

BIC Bayesian Information Criterion.

A-BIC Adjusted BIC, see section B.2.3.

pars The number of parameters of the model.

free The number of free parameters of the model, see section B.2.3.

A-free The number of adjusted free parameters of the model, see section B.2.3.

B.2.5 Explorative fitting of models: `batch`

This routine does essentially the same as the `explore` routine. However, it does it for a number of datasets. As a consequence the `command` file is essentially identical the `command` file for the `explore` routine. There are two differences. First, the `datafile` should specify a file which contains the names of the files that have the datasets to be analysed. Second, there should be an extra command named `inputfiles` which specifies the number of datasets to be processed. From there on, the rest is the same as for the explore routine. Each dataset that is analysed gets its own output files. Note that this only works for datasets that are identical, that is, they should have the same number of datapoints and the same number of categories.

B.2.6 Generating data: `generate`

Generating data according to a given HMM can be done with the command line `job generate`. After this command line the `command file` should have the data description commands that are again identical to those in the `fitting` and `explore` routines. See section B.2.3 for a full description of all the optional and required commands. After the data description commands the model according to which the data should be generated is to be specified using the `states` and `modelfile` commands. The generated dataset is written to a file called `<modelfile>.data`. Note that this file is overwritten if it already exists.

B.2.7 Other commands

There are some other commands that relate to optimizing models which can be used with either the `fitting` or `explore` routines. They are the following:

`iterMax` The maximum number of iterations used in optimizing models. It has a default value of 999.

`iterStart` The minimal number of iterations used in optimizing models. It has a default value of 6. `iterStart` iterations are done before the loglikelihood criterion is tested. This number of iterations is also used in selection of random starting values in the `explore` routine. In the `explore` routine ten sets of random starting values are generated. Each of the resulting models are iterated `iterStart` times and then the best of these models is selected, that is, the model with the highest loglikelihood.

`kmeans` If this command is specified and given a value of one, K-means estimation of the model will be done at the start of optimization. This can speed up optimization considerably but in some cases also results in suboptimal solutions. See Rabiner (1989) for a description of the algorithm. At the end of optimization another K-means estimate of the model is generated and written to the output file.

`optCriterion` With this command the optimization criterion can be specified. Its default value is $1.0\,10^{-9}$.

`setzero` By default at every step of the EM-algorithm parameters are checked for sigfnicance. If they are found to be non-significant parameters they are set to zero and optimization continues from there. The reason for doing this is that it speeds up optimization considerably. Although some checks are performed to see if setting the parameter to zero leads to an impossible or otherwise inadmissable model, these checks are not 100% foolproof. As a result it is possible that setting a parameter to zero leads to a loglikelihood that is not a number (nan on the console output). Setting the `setzero` to 0 in the command file takes care that optimization is done without setting parameters to zero. This can result in parameter estimates that are very close to zero but nevertheless do not reach zero before optimization is over. Also this can lead to slow optimization and hence it may be necessary to increase the maximum number of allowed iterations.

`verbose` If this command is specified and given a value of one, screen outputs will be provided during optimization and exploration.