Journal of Statistical Software

August 2016, Volume 72, Issue 3. doi: 10.18637/jss.v072.i03

Analyzing State Sequences with Probabilistic Suffix Trees: The PST R Package

Alexis Gabadinho
NCCR LIVES, University of Geneva

Gilbert Ritschard
NCCR LIVES, University of Geneva

Abstract

This article presents the **PST** R package for categorical sequence analysis with probabilistic suffix trees (PSTs), i.e., structures that store variable-length Markov chains (VLMCs). VLMCs allow to model high-order dependencies in categorical sequences with parsimonious models based on simple estimation procedures. The package is specifically adapted to the field of social sciences, as it allows for VLMC models to be learned from sets of individual sequences possibly containing missing values; in addition, the package is extended to account for case weights. This article describes how a VLMC model is learned from one or more categorical sequences and stored in a PST. The PST can then be used for sequence prediction, i.e., to assign a probability to whole observed or artificial sequences. This feature supports data mining applications such as the extraction of typical patterns and outliers. This article also introduces original visualization tools for both the model and the outcomes of sequence prediction. Other features such as functions for pattern mining and artificial sequence generation are described as well. The **PST** package also allows for the computation of probabilistic divergence between two models and the fitting of segmented VLMCs, where sub-models fitted to distinct strata of the learning sample are stored in a single PST.

Keywords: state sequences, categorical sequences, sequence visualization, sequence data mining, variable-length Markov chains, probabilistic suffix trees, R.

1. Introduction

The analysis of categorical sequences is involved in various fields, including biology, computer science, and behavioral and social sciences. Several methodological approaches are available for analyzing such sequences. One possible approach for comparing whole sequences and finding frequent patterns in sequential databases uses sequence alignment methods. Pairwise sequence alignment, widely used for biological sequences, is based on algorithms that generate

pairwise edit distances. This method, known as optimal matching (OM), has been extended to the social sciences (Abbott and Forrest 1986; Abbott 1995) to study categorical sequences representing careers (life trajectories) and many other types of social processes. The pairwise distance matrix is normally input to a clustering procedure aimed at finding typical existing patterns. Although the method has yielded meaningful empirical results in this field, critics have argued that the assumptions underlying OM distances are not transposable from biological sciences to social sciences, where sequences unfold (grow step by step) with time (Levine 2000) as the result of a dynamic probabilistic process.

Another methodological approach, based on Markovian models, considers categorical sequences as the result of a stochastic process. Unlike the pairwise distance approach described above, it is based on the modeling of the underlying process that generates the observed data. The estimated model provides transition probabilities between the distinct states in the sequences. Many processes studied in various scientific fields, from natural sciences to social sciences (Bartholomew 1973), fall under this framework and have long been studied using Markov chain models (MCMs). However, for several reasons, this approach is not commonly used to analyze complex sequences such as individual life trajectories.

The Markov hypothesis states that the conditional probability of a state in a sequence depends on a fixed number of past states, which is defined by the order (memory) L of the chain. The number of parameters to be estimated increases exponentially with L; thus, fitting high-order models becomes infeasible. Most applications of Markov chains use first- or low-order models. In the social sciences, for example, MCMs have been typically used to study intergenerational (from parents to children) educational or social mobility and geographic migration, the input data being an aggregated cross table representing flows between states at two time points.

However, a realistic model of real-world processes must consider high-order statistical correlations (Raftery 1985; Berchtold and Raftery 2002; Langeheine and Van de Pol 2000; Ching, Fung, and Ng 2004). Among the various Markovian model extensions, variable-length Markov chains (VLMCs) are an interesting alternative for this purpose. In such generative probabilistic models, the length of the memory depends on the *context*, i.e., L varies with the particular series of previous states in the sequence. Thus, these models are especially suited for capturing high-order dependencies in (sets of) sequences when such dependencies exist for particular contexts only. The ability of VLMCs to model complex sequential data, as well as the fact that they can be learned easily and efficiently from the data without requiring complex estimation procedures, makes them superior to both MCMs and widely used hidden Markov models (HMMs) (Seldin, Bejerano, and Tishby 2001; Begleiter, El-Yaniv, and Yona 2004). The structures used to store these models, referred to as probabilistic suffix trees (PSTs), make both learning and *a posteriori* optimization of the model intuitive and straightforward.

The potential applications of PSTs are numerous and manifold. They can be used not only for exploring the process generating the analyzed sequences but also for machine learning applications. Examples include supervised and unsupervised classification in various domains such as proteomics, genomics, linguistics, and classification of musical styles (Galves and Löcherbach 2008). PSTs also have significant potential for pattern mining, i.e., identification of typical (sub)sequences or outliers in sequence databases.

Social sciences sequences are mainly built from longitudinal survey data and represent individual life trajectories or careers. These sequences are usually complex and potentially exhibit high-order dependencies. Although VLMCs are well suited for analyzing such data, these

models, to the best of our knowledge, have not been used in social sciences thus far. The sole existing R (R Core Team 2016) package for fitting VLMCs (**VLMC**; Mächler 2015) can be used only with single categorical time series and does not consider case weights or missing values, which limits its applications. In this article, we present an implementation of VLMCs and PSTs as an R package that extends the **TraMineR** package (Gabadinho, Ritschard, Müller, and Studer 2011a) for categorical sequence analysis. The package is specifically adapted to the field of social sciences, as it allows for PSTs to be learned from sets of individual sequences possibly containing missing values; in addition, the package is extended to account for case weights. However, the use of the package is by no means limited to social sciences sequences.

Furthermore, this article shows how the **PST** package (Gabadinho 2016) can be used to learn discrete-time stationary models. In addition, it describes the various tools and features provided by **PST** to analyze sequence data with these models, i.e., new visualization tools and functions for model optimization, sequence prediction, artificial sequence generation, and context and pattern mining. The package also allows for the fitting of segmented VLMCs, where conditional transition probabilities are estimated and stored for each value of a covariate. The package is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=PST`.

The remainder of this article is organized as follows. Section 2 introduces VLMCs. Section 3 presents a simple example to demonstrate the process of growing and pruning a PST that stores a VLMC model. Section 4 presents a real-world example to show how to (i) learn a PST on a set of weighted social sciences sequences containing missing data, (ii) tune and optimize the model, and (iii) measure sequence prediction quality. Section 5 describes tools for exploring PSTs and pattern mining. Section 6 explains how to fit segmented models and compare data partitions using such models. Finally, Section 7 summarizes our findings and concludes the article.

2. Markov chain models

Let A be a finite alphabet of size $|A|$. A state (categorical) sequence $x = x_1, x_2, \ldots, x_\ell$ of length ℓ is an ordered list of ℓ elements taken from A. The indexes $1, \ldots, \ell$ refer to the positions in the sequence, that correspond to time units in most applications of Markov chain models (MCMs). We consider the sequence x as the result of a discrete-time stochastic process, i.e., as successive realizations of a random variable X. The probability of x can be computed by using the chain rule

$$\mathsf{P}(x) = \mathsf{P}(x_1) \times \mathsf{P}(x_2|x_1) \times \mathsf{P}(x_3|x_1x_2) \times \ldots \times \mathsf{P}(x_\ell|x_1 \ldots x_{\ell-1}), \tag{1}$$

where the probability of each state in the sequence is conditional on the past observed states.

2.1. The Markov hypothesis

The Markov hypothesis states that the whole past is summarized by a limited number L of past states, i.e., the period $x_{t-L}, \ldots, x_{t-1}$. In a first-order Markov chain, the past is summarized by the previous state, i.e., $L = 1$, and we have

$$\mathsf{P}(x) = \mathsf{P}(x_1) \times \mathsf{P}(x_2|x_1) \times \mathsf{P}(x_3|x_2) \times \cdots \times \mathsf{P}(x_\ell|x_{\ell-1}). \tag{2}$$

A MCM of order L is summarized by a series of $|A|^L \times |A|$ conditional probabilities $\mathsf{P}(X_t = \sigma|X_{t-L}, \ldots, X_{t-1})$, $\sigma \in A$, called transition probabilities. The fixed number L of previous states on which the state probabilities are conditional is referred to as the *memory*. In this paper, we focus on discrete-time stationary Markov chains, i.e., we assume time-homogeneous transition probabilities.

MCMs are well suited for analyzing a wide variety of categorical sequences in different scientific fields, for example, DNA sequences in biology, behavioral sequences in ethology, individual or collective social or geographical mobility in social sciences, and developmental processes in psychology (e.g., Bartholomew 1973; Singer and Spilerman 1976; Berchtold 1998, 2010; Avery and Henderson 1999; Kaplan 2008). Nonetheless, classical MCMs suffer from a lack of flexibility, which limits their application to real-life data. The main problem is the assumptions required to ensure that the model has a reasonable number of parameters.

One important drawback is that the number of free parameters of a fixed-length MCM, $K = (|A| - 1)|A|^L$, increases exponentially with L. Capturing high-order statistical correlations existing in the sequences requires complex models and usually involves the estimation of a large number of parameters relative to the size of the learning sample. The complexity is further increased when a realistic modeling requires to consider heterogeneity, i.e., variation in the transition probabilities across distinct strata of the analyzed data, or to relax the stationarity assumption. Consequently, to keep the model reasonably simple, most applications use low-order models, even if they represent an oversimplification of the real-world scenario.

Several extensions of standard MCMs with greater flexibility overcome some of these drawbacks, e.g., hidden Markov models (Rabiner 1989), mixture Markov models and latent mixed Markov models (Langeheine and Van de Pol 1990), double chain Markov models (DCMM; Berchtold and Sackett 2002), or mixture transition distribution model (MTD; Berchtold and Raftery 2002). Variable-length Markov chains (VLMCs; Ron, Singer, and Tishby 1996; Bühlmann and Wyner 1999), also known as variable-order Markov chains (VOMCs), are a class of models that exhibit many interesting characteristics. They are especially effective in capturing particular high-order dependencies in sequences while remaining parsimonious and simple to estimate. Thus, VLMC models can be learned efficiently even on small data sets. These features make VLMCs useful alternatives to HMMs (Ron *et al.* 1996; Bejerano and Yona 2001; Seldin *et al.* 2001), e.g., in biological sequence analysis.

2.2. Variable-length Markov chains

The *context* is the observed subsequence[1] $x_1, \ldots, x_{t-1}$ preceding a state x_t, $1 \leq t \leq \ell$.[2] In VLMC models, the number of considered past states, L, varies with each particular context, i.e., L is a function of the past. For each distinct context $c = c_1, \ldots, c_k$ there exists a length $L, 0 \leq L \leq k$, such that

$$\mathsf{P}(\sigma|c_1, \ldots, c_k) \approx \hat{\mathsf{P}}(\sigma|c_{k-L}, \ldots, c_k), \tag{3}$$

which was described by Ron *et al.* (1996) as the *short memory property*.

[1] As in the case of most studies on Markov modeling, here, we define a subsequence y of a sequence $x = x_1, \ldots, x_t$ as a series of adjacent symbols $y = x_i, \ldots, x_j$ taken from x, where $i \geq 1$ and $j \leq \ell$.

[2] Here we assume that sequences are of finite size, and the maximal context length is $L = \ell - 1$, where ℓ is the maximal sequence length.

Assuming that the process generating the learning sample is a VLMC of maximal context length $L_{\max}$, the VLMC learning algorithm estimates the length L for each context appearing in the data. Contexts of maximum length are first extracted from the data, and the algorithm successively compares the conditional probability distribution of a context to that of its longest *suffix*. For a particular context $c = c_1, c_2, \ldots, c_k$, if

$$\hat{\mathsf{P}}(\sigma|c) \simeq \hat{\mathsf{P}}(\sigma|\,\mathrm{suf}(c)),\ \sigma \in A, \tag{4}$$

where $\mathrm{suf}(c) = c_2, \ldots, c_k$, keeping $\mathsf{P}(\sigma|c)$ in the model would increase the number of parameters without improving information. Therefore, a VLMC is efficiently stored and represented as a structure known as a probabilistic suffix tree (PST), which is sometimes also referred to as a prediction suffix tree. The next section explains how to learn a VLMC model with the **PST** package.

3. Basic functions of the PST package

The first step of sequence analysis with VLMCs is to learn (fit) a model from a learning sample, i.e., a set of one or several categorical sequences. The main PST learning algorithms described in the literature are the *context* (Rissanen 1983; Bühlmann and Wyner 1999) and the *Learn-PSA* (Ron *et al.* 1996; Bejerano and Yona 2001) algorithms. The **PST** package implements existing approaches under a general and flexible framework, thereby allowing for easy tuning and comparison of models. Learning a VLMC on one or more categorical sequences with the package involves two main stages. In the first stage, the next-symbol conditional probability distributions are computed for the contexts appearing in the data and stored in a PST. In the second stage, the tree is pruned to remove non-significant information. Once the model is learned, it can be used to compute the likelihood of any particular sequence built on the alphabet of the learning sample; this feature is known as sequence prediction, and its principle is explained in the second part of this section.

3.1. Learning the model

To illustrate the learning process, we consider a single example sequence of length $\ell = 27$ built on the alphabet $A = \{a, b\}$. A sequence object is created with the `seqdef()` function (Gabadinho *et al.* 2011a):

```
R> data("s1", package = "PST")
R> s1 <- seqdef(s1)
R> s1

    Sequence
[1] a-b-a-a-b-a-a-b-a-a-b-b-b-b-a-b-a-b-b-a-a-a-b-a-b-b-b
```

Growing the tree

The PST is built by successively adding contexts of increasing length k. A node labeled with the context $c = c_1, \ldots, c_k$ stores the empirical conditional probability $\hat{\mathsf{P}}(\sigma|c)$ of observing a

symbol $\sigma \in A$ after the subsequence c, i.e.,

$$\hat{\mathsf{P}}(\sigma|c) = \frac{N(c\sigma)}{\sum_{\omega \in A} N(c\omega)}, \tag{5}$$

where

$$N(c) = \sum_{i=1}^{\ell} \mathbb{1}\left[x_i, \ldots, x_{i+|c|-1} = c\right], \; x = x_1, \ldots, x_\ell, \; c = c_1, \ldots, c_k \tag{6}$$

is the number of occurrences of the subsequence c in the sequence x and $c\sigma$ is the concatenation of the subsequence c and the symbol σ. The empirical probability distributions $\hat{\mathsf{P}}(\sigma|c)$ are the maximal likelihood estimations of the true distributions $\mathsf{P}(\sigma|c)$, i.e., the estimations of the model's parameters.

The root node is labeled with the empty string `e` and contains the zeroth-order empirical probabilities $\hat{\mathsf{P}}(a) = 13/27 = 0.48$ and $\hat{\mathsf{P}}(b) = 14/27 = 0.52$. The counts are returned by the `cprob()` function:

```
R> cprob(s1, L = 0, prob = FALSE)
```

```
   a  b [n]
e 13 14  27
```

The first-order probabilities, $\hat{\mathsf{P}}(\sigma|\mathtt{a})$ and $\hat{\mathsf{P}}(\sigma|\mathtt{b})$, $\sigma \in \{\mathtt{a}, \mathtt{b}\}$, are estimated from 26 states because the last sequence position (`b`) in `s1` is not followed by any symbol:

```
R> cprob(s1, L = 1, prob = TRUE)
```

```
       a      b [n]
a 0.3846 0.6154  13
b 0.5385 0.4615  13
```

The nodes labeled by `a` and `b` have the root node `e` as parent (the longest proper suffix[3] of a string of length 1 is the empty string `e`). The four distinct subsequences of length 2 (`a-a`, `a-b`, `b-a`, and `b-b`) appearing in `s1` are then added to the tree, with the following conditional probabilities:

```
R> cprob(s1, L = 2, prob = TRUE)
```

```
         a      b [n]
a-a 0.2000 0.8000   5
a-b 0.6250 0.3750   8
b-a 0.5714 0.4286   7
b-b 0.4000 0.6000   5
```

[3] A proper suffix of a string is the longest suffix that is not the string itself.

Each of the four nodes is connected to the node labeled with its longest suffix: `a-a` and `b-a` are children of `a`, while `a-b` and `b-b` are children of `b`.

By default, the growing stage stops when every distinct subsequence c of length $k \leq \ell - 1$ found in the data is added to the tree. The process can be controlled by two optional parameters: L, the maximal depth of the tree, i.e., the maximal allowed context length, and *nmin*, the required minimal frequency $N(c)$ of a subsequence c in the data to add it in the tree.[4]

Probability smoothing

Estimating the next-symbol conditional probability distributions from empirical counts may cause a null probability to be assigned to patterns that do not appear in the training data. This is a problem for predicting sequences that do not belong to the learning sample (see Section 3.2) and also for the pruning stage described below. Probability smoothing ensures that the PST contains no null probability. Given a parameter *ymin*, the probability of each symbol $\sigma \in A$ is smoothed as follows[5]

$$\hat{\mathsf{P}}(\sigma|c) = (1 - |A| \times ymin)\hat{\mathsf{P}}(\sigma|c) + ymin. \tag{7}$$

In the following example, we build a PST with maximal depth $L = 3$, $nmin = 1$, and smoothing parameter $ymin = 0.001$.

```
R> S1 <- pstree(s1, L = 3, ymin = 0.001)
R> print(S1, digits = 2)

--(e)-[ p=(0.48,0.52) - n=27 ]
  `--(a)-[ p=(0.38,0.62) - n=13 ]
     `--(a-a)-[ p=(0.20,0.80) - n=5 ]
        `--(a-a-a)-[ p=(0.001,0.999) - n=1 ]--|
        `--(b-a-a)-[ p=(0.25,0.75) - n=4 ]--|
     `--(b-a)-[ p=(0.57,0.43) - n=7 ]
        `--(a-b-a)-[ p=(0.60,0.40) - n=5 ]--|
        `--(b-b-a)-[ p=(0.50,0.50) - n=2 ]--|
  `--(b)-[ p=(0.54,0.46) - n=13 ]
     `--(a-b)-[ p=(0.62,0.38) - n=8 ]
        `--(a-a-b)-[ p=(0.75,0.25) - n=4 ]--|
        `--(b-a-b)-[ p=(0.33,0.67) - n=3 ]--|
     `--(b-b)-[ p=(0.40,0.60) - n=5 ]
        `--(a-b-b)-[ p=(0.33,0.67) - n=3 ]--|
        `--(b-b-b)-[ p=(0.50,0.50) - n=2 ]--|
```

In the resulting tree, $\hat{\mathsf{P}}(\mathtt{a}|\mathtt{a\text{-}a\text{-}a}) = 0.001$, while the empirical probability of observing `a` after the string `a-a-a` is null in the learning sample. A *leaf* is a node that has no child, which occurs when the maximum context length is reached. Leaves are identified with the symbol '`-|`' when the tree is displayed, as shown in the example above.

[4] An alternative considered by Largeron (2003) is to use $\varphi \geq pmin$, where φ is the empirical probability of observing the subsequence c in the sequence x.

[5] An alternative smoothing procedure has been proposed by Kermorvant and Dupont (2002).

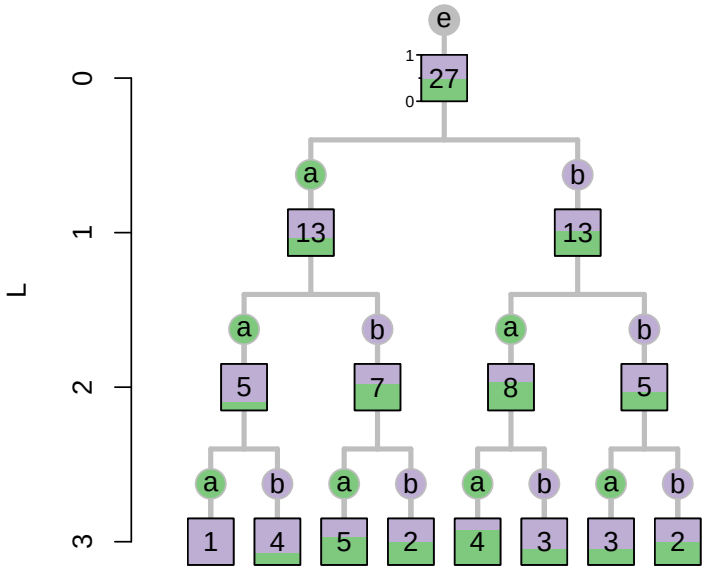

Figure 1: Default graphical representation of a PST provided by the **PST** package. The context c labeling a node is the path in the tree up to the root node `e`. The probability distribution $\mathsf{P}(\sigma|c)$ in each node is displayed as a bar plot, and the number of observations on which the distribution is estimated (i.e., the number of times the pattern c precedes a symbol) is displayed inside the bar plot.

Plotting the tree

The `plot` method[6] produces a graphical display of the tree. By default, a bar plot representing the probability distribution stored in each node is displayed. Figure 1 is obtained with

```
R> plot(S1, axis = TRUE, withlegend = FALSE)
```

The node at the bottom left of the tree stores the probability distribution $\hat{\mathsf{P}}(\sigma|\texttt{a-a-a}), \sigma \in \{a, b\}$, and the node immediately to its right stores $\hat{\mathsf{P}}(\sigma|\texttt{b-a-a})$. Thus, the context labeling the node is the path in the tree up to the root node `e`. Figure 2 shows a more classical representation of a PST that can be obtained by setting the `nodePar` and `edgePar` arguments as follows:

```
R> plot(S1, nodePar = list(node.type = "path", lab.type = "prob",
+     lab.pos = 1, lab.offset = 2, lab.cex = 0.6),
+     edgePar = list(type = "triangle"), withlegend = "FALSE")
```

Pruning

The initial growing stage may yield an overly complex model containing all contexts of maximal length L and frequency $N(c) \geq nmin$ found in the learning sample. The pruning stage potentially reduces the number of nodes in the tree, and thus, the model complexity. It compares the conditional probabilities associated with a node labeled by a subsequence

[6]The package is written using S4 classes.

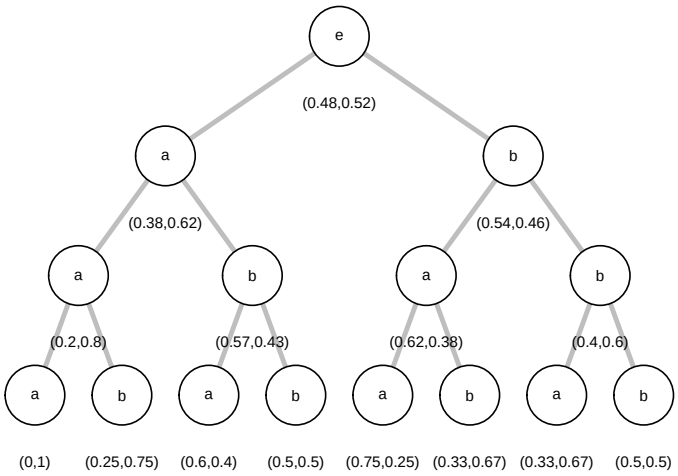

Figure 2: More classical view of the PST. The probability distribution in each node is displayed as text.

$c = c_1, c_2, \ldots, c_k$ to the conditional probabilities of its parent node labeled by the longest suffix of c, $\text{suf}(c) = c_2, \ldots, c_k$. The general idea is to remove a node if it does not contribute additional information with respect to its parent in predicting the next symbol, i.e., if $\hat{\mathsf{P}}(\sigma|c)$ is not *significantly* different from $\hat{\mathsf{P}}(\sigma|\,\text{suf}(c))$ for all $\sigma \in A$.

The pruning procedure starts from the terminal nodes and is applied recursively until all terminal nodes remaining in the tree represent an information gain relative to their parent. A gain function whose outcome will determine the pruning decision is used to compare the two probability distributions. The gain function is driven by a cutoff, and the complexity of the tree varies with the value of this cutoff. Methods for selecting the pruning cutoff are described in Section 4.

The first implemented gain function, which is used by the *Learn-PSA* algorithm, is based on the ratio between $\hat{\mathsf{P}}(\sigma|c)$ and $\hat{\mathsf{P}}(\sigma|\,\text{suf}(c))$ for each $\sigma \in A$. A node represents an information gain if, for any symbol $\sigma \in A$, the ratio is greater than the cutoff C or less than $1/C$, i.e., if

$$G_1(c) = \sum_{\sigma \in A} \mathbb{1}\left[\frac{\hat{\mathsf{P}}(\sigma|c)}{\hat{\mathsf{P}}(\sigma|\,\text{suf}(c))} \geq C \text{ or } \frac{\hat{\mathsf{P}}(\sigma|c)}{\hat{\mathsf{P}}(\sigma|\,\text{suf}(c))} \leq \frac{1}{C}\right] \geq 1, \tag{8}$$

where C is a user-defined cutoff value.[7] Nodes that do not satisfy the above condition are pruned. For $C = 1$, no node is removed because even a node whose next probability distribution is similar to that of its parent does not satisfy the pruning condition.

The PST built for our example sequence with[8] $nmin = 2$ and pruned with $C = 1.20$ is obtained as follows:

```
R> S1 <- pstree(s1, L = 3, nmin = 2, ymin = 0.001)
R> S1.p1 <- prune(S1, gain = "G1", C = 1.2, delete = FALSE)
```

[7] In their application for the prediction of protein families, Bejerano and Yona (2001) used $C = 1.05$. Largeron (2003) compared results with $C = 1$, $C = 1.05$, and $C = 1.2$.

[8] The node labeled `a-a-a` is no longer present in the tree because this subsequence appears only once in the data.

The `delete = FALSE` argument prevents the deletion of the pruned nodes, which appear crossed out by red lines in the graphical representation of the tree (see Figure 3).

The *context* algorithm uses another gain function, i.e.,

$$G_2(c) = N(c) \sum_{\sigma \in A} \hat{\mathsf{P}}(\sigma|c) \log \left(\frac{\hat{\mathsf{P}}(\sigma|c)}{\hat{\mathsf{P}}(\sigma|\,\mathrm{suf}(c))} \right) > C, \tag{9}$$

where c is the context labeling the terminal node and $N(c)$ is the number of occurrences of c in the training data. The cutoff C is specified on the scale of χ^2 quantiles (Mächler and Bühlmann 2004),

$$C_\alpha = \frac{1}{2}\,\mathrm{qchisq}(1-\alpha, v), v = |A| - 1 \tag{10}$$

where $\mathrm{qchisq}(p = 1-\alpha, v)$ is the quantile function of a χ^2-distribution with v degrees of freedom. Typical values of α are 5% and 1%, yielding $p = 0.95$ and $p = 0.99$, respectively. The cutoff C is a threshold for the difference in deviance between a tree S^1 and its subtree S^2 obtained by pruning the terminal node c. The deviance of a model S is

$$D(S) = -2\,\mathrm{logLik}(S), \tag{11}$$

where $\mathrm{logLik}(S)$ is the log-likelihood of the data for model S, i.e.,

$$\mathrm{logLik}(S) = \sum_{j=1}^{n} \log \mathsf{P}^S(x^j), \tag{12}$$

where $\mathsf{P}^S(x^i)$ is the probability of the ith observed sequence as predicted by S.

Unlike $G_1(c)$, the $G_2(c)$ gain function returns a value to be compared to the cutoff C that is proportional to $N(c)$. It is intended to be applied to an initial tree grown with $nmin = 2$ (Bühlmann and Wyner 1999; Mächler and Bühlmann 2004), whereas trees pruned with $G_1(c)$ are usually grown with $nmin = 30$. In our example, $v = 1$, and setting $C_{\alpha=0.05} = 1.92$ yields a tree where all the nodes up to the root node are removed (Figure 3). The small number of occurrences of each context is not sufficient for the difference between probability distributions to be significant, whereas only two nodes are removed when pruning with the $G_1(c)$ function.

```
R> C95 <- qchisq(0.95, 1) / 2
R> S1.p2 <- prune(S1, gain = "G2", C = C95, delete = FALSE)
```

3.2. Sequence prediction

One particularly interesting feature of VLMCs is that a model S built over an alphabet A allows for the computation of the probability (likelihood) of any particular sequence $x \in A^\ell$, $\ell \in \mathbb{N}$. Indeed, S is a generative model representing a probability distribution for any complete set of sequences A^ℓ, i.e.,

$$\forall \ell \in \{0, 1, 2, \ldots\} : \sum_{x \in A^\ell} \mathsf{P}^S(x) = 1. \tag{13}$$

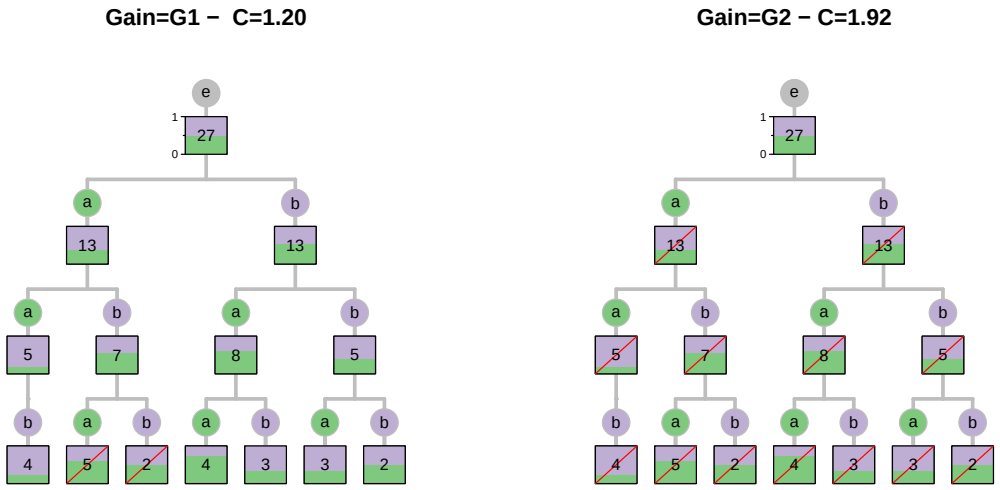

Figure 3: PST for the `s1` example sequence, pruned with the G_1 and G_2 gain functions.

i	$\mathsf{P}(x_i \mid x_1 \dots x_{i-1})$	Node	$\mathsf{P}^S(x_i)$
1	$\mathsf{P}(a)$	`e`	0.48
2	$\mathsf{P}(b \mid a)$	`a`	0.62
3	$\mathsf{P}(a \mid ab)$	`a-b`	0.62
4	$\mathsf{P}(a \mid aba)$	`b-a`	0.57
5	$\mathsf{P}(b \mid abaa)$	`b-a-a`	0.75

Table 1: Computing the probability of the sequence *abaab* using the PST `S1.p1`.

This holds for any length $\ell \in \mathbb{N}$ because according to the *consistency* property, the sum of the probabilities of all continuations of a sequence equals the probability of the initial sequence (see Wiewiora 2008), i.e.,

$$\forall \ell \in \{0, 1, 2, \dots\}, \forall x \in A^\ell : \mathsf{P}(x) = \sum_{\sigma \in A} \mathsf{P}(x\sigma), \tag{14}$$

where A^0 is the empty string `e`, and $\mathsf{P}(\mathtt{e}) = 1$.

The computation of the likelihood of a particular sequence given a model S is called *sequence prediction*. Let us, for example, compute the likelihood $\mathsf{P}^S(x)$ of the sequence $x =$ `a-b-a-a-b`, given the PST `S1.p1` learned previously. $\mathsf{P}^S(x)$ represents the probability that the model S generates x, which is obtained by computing the conditional probability of each state in x, i.e.,

$$\mathsf{P}^S(abaab) \quad = \quad \mathsf{P}^S(a) \times \mathsf{P}^S(b|a) \times \mathsf{P}^S(a|ab) \times \mathsf{P}^S(a|aba) \times \mathsf{P}^S(b|abaa), \tag{15}$$

where $S =$ `S1.p1`.

The probability of each of the states is retrieved from the PST, as shown in Table 1. To get, for example, $\mathsf{P}^S(a|aba)$, the tree is scanned for the node labeled with the string `a-b-a`, and if this node does not exist, the tree is scanned for the node labeled with the longest suffix of this string, i.e., `b-a`, and so on. Because the node `a-b-a` has been removed in the pruning stage, the longest suffix of `a-b-a` in `S1.p1` is `b-a`:

```
R> query(S1.p1, "a-b-a")

 [>] context: b-a
        a      b
S1 0.5714 0.4286
```

The sequence likelihood is returned by the `predict()` function. By setting `decomp = TRUE`, the output is a matrix containing the probability of each symbol in the sequence, i.e., the probabilities in Equation 15:

```
R> x <- seqdef("a-b-a-a-b")
R> predict(S1.p1, x, decomp = TRUE)

       [1]    [2]    [3]    [4]  [5]
[1] 0.4815 0.6154 0.625 0.5714 0.75
```

$\mathsf{P}^S(x)$ can be transformed into a more convenient prediction quality measure such as the *average log-loss*

$$\text{logloss}(S, x) = -\frac{1}{\ell}\sum_{i=1}^{\ell}\log_2 \mathsf{P}^S(x_i|x_1 \ldots x_{i-1}) = -\frac{1}{\ell}\log_2 \mathsf{P}^S(x). \tag{16}$$

The value is the average log-loss of each state in the sequence, which allows for the predictions of sequences of unequal lengths to be compared. The average log-loss can be interpreted as a residual, i.e, the distance between the prediction of a sequence by a PST S and the perfect prediction $\mathsf{P}(x) = 1$ yielding $\text{logloss}(\mathsf{P}^S, x) = 0$. The lower the value of $\text{logloss}(\mathsf{P}^S, s)$ the better is the prediction of the sequence. A difference δ in the average log-loss of two sequences implies that the sequence having the lower value is 2^δ times more likely.

Average log-loss has been used, for example, by Begleiter *et al.* (2004) to compare the prediction performance of several types of VLMC models. Other related measures (i.e., similarity measures) are used to compare predictions of single sequences by several distinct VLMC models in applications such as supervised (Bejerano and Yona 2001) and unsupervised (Yang and Wang 2003) sequence classification (see also Section 5.2).

In the following example, we compute the average log-loss for predicting three sample sequences with the tree `S1.p1`:

```
R> ex2 <- c("a-a-b", "a-b-a-a-b", "b-b-b-b-a")
R> ex2 <- seqdef(ex2)
R> predict(S1.p1, ex2, output = "logloss")

    logloss
[1]  0.9183
[2]  0.7311
[3]  0.9600
```

4. Learning a model from a real data set

To illustrate more advanced aspects of sequence analysis using VLMCs, we now consider a set of individual sequences that contain missing values and have attached case weights, as in many social science longitudinal data sets. This section describes the solutions implemented in the **PST** package for learning a model on such data sets and shows how the model can be optimized, i.e., how to set the growing and pruning parameters.

4.1. Data

The `SRH` example data set[9] contains sequences of self-rated health (SRH) for 2612 respondents of a survey conducted by the Swiss Household Panel (SHP), aged between 20 and 80 years at the start of the survey. The `SRH.seq` sequence object is defined using **TraMineR**'s `seqdef()` function and is also included in the package (see `help(SRH)`).

SRH data is collected at each yearly wave of the survey; the question *How do you feel right now?* is posed to the respondents. There are 5 possible ratings that constitute the alphabet of the sequences: very well (G1), well (G2), so-so (average) (M), not very well (B2), and not well at all (B1). The sequences include the responses of an individual over 11 years, starting in 1999, with possibly missing states due to no response at one or more waves. Case weights are used to account for the sampling scheme and missing responses. The first sequences are plotted[10] in Figure 4 together with the state distribution at each sequence position.

Although SRH is a complex construct combining physical, psychological, and social well-being (Smith, Shelley, and Dennerstein 1994), it is known to accurately represent an individual's health status and to be a good predictor of morbidity and mortality (Idler and Benyamini 1997). Health at a given point in time can be considered as the result of a process that includes an erosion component, i.e., a decline in health due to ageing. Although the SRH data is collected at intervals of one year and covers only a limited period of the whole process, we use it to illustrate how VLMCs and PSTs can provide insights into the process as well as to model (possibly) high-order dependencies in SRH sequences.

4.2. Growing the tree

We now consider a set of 2612 sequences of equal length, $\ell = 11$. Our aim is to learn a single model from this data, i.e., we hypothesize that all sequences were generated by the same model. An important issue is how to deal with missing states in some of the sequences.[11] One option is to build the tree by considering only contexts without missing states. We build such an initial PST with maximal possible depth $L = \ell - 1 = 10$ (i.e., the memory goes up to the 10 previous waves) and the growing parameters[12] described in Section 3.

```
R> data("SRH", package = "PST")
R> SRH.pst.L10 <- pstree(SRH.seq, L = 10, nmin = 2, ymin = 0.001)
```

[9] The data set has been included in the package with the authorization of the Swiss Household Panel.

[10] The height of the bar in the sequence index plot is proportional to the sequence weight (Gabadinho *et al.* 2011a).

[11] Gabadinho *et al.* (2011a) describe how missing values are dealt with in sequence objects.

[12] The constraint *nmin* on the number of occurrences of the contexts is checked before substrings followed by a missing state are removed. Therefore, some of the final counts could be less than *nmin*.

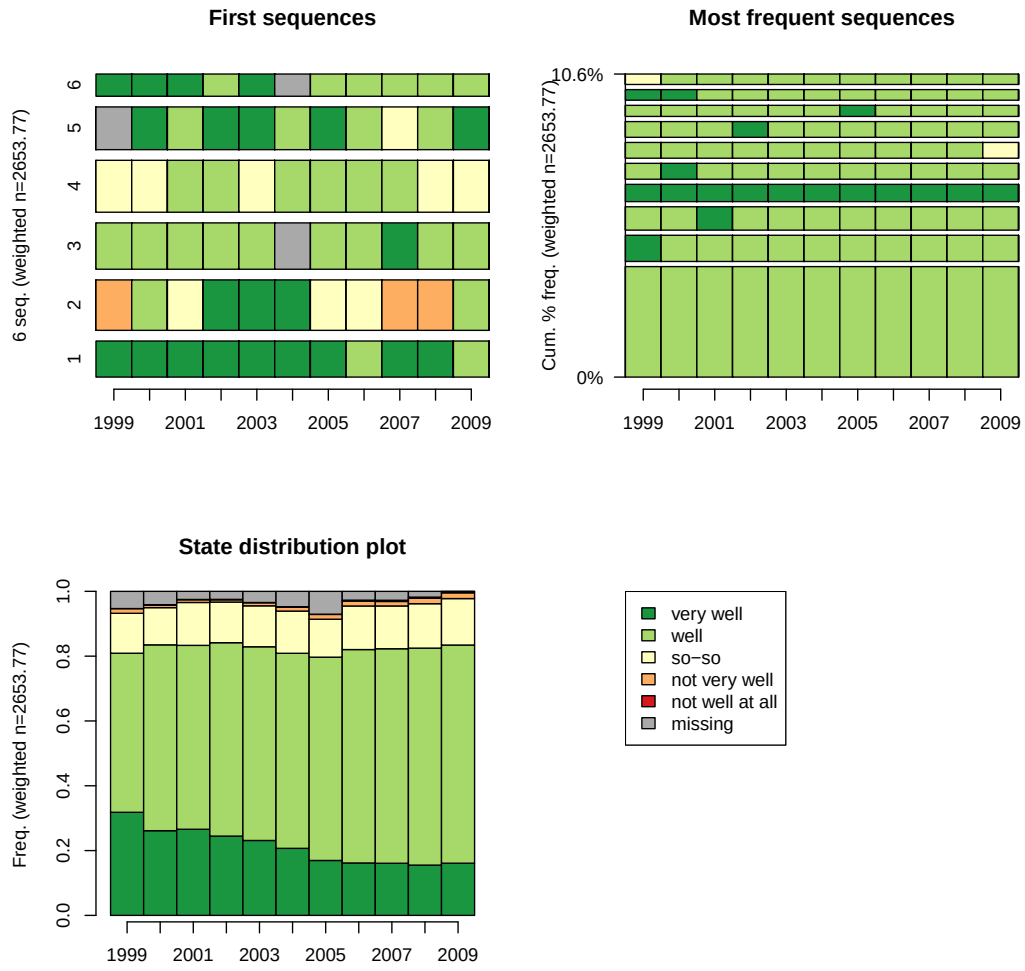

Figure 4: First sequences, most frequent sequences, and position-wise state distribution in the `SRH.seq` sequence object.

The resulting model describes the process generating the sequences, using all available valid information in the data. However, ignoring missing states has several drawbacks. Let us consider the following sequence of our data set:

```
R> SRH.seq[3, ]

  Sequence
3 G2-G2-G2-G2-G2-*-G2-G2-G1-G2-G2
```

The probability of the state at position 6, and thus, of the whole sequence, cannot be predicted by the PST (Table 2). Moreover, the probability of the state at position 7 is retrieved from the root node, because neither the context `G2-G2-G2-G2-G2-*` nor any of its suffixes are present in the tree. Similarly, the prediction of states from position 8 onward can use the memory only from position 7 onward.

For predicting sequences that include missing values, we can first impute the missing states. One simple solution for imputation is to use the `impute()` function provided by the package.

	G2	G2	G2	G2	G2	*	G2	G2	G1	G2	G2
A	0.63	0.74	0.78	0.81	0.82		0.63	0.74	0.12	0.66	0.75
B	0.63	0.74	0.78	0.81	0.82	0.84	0.85	0.88	0.05	0.60	1.00
C	0.61	0.72	0.76	0.79	0.80	0.03	0.65	0.82	0.04	0.40	0.76

Table 2: Probability of successive states in sequence 3 using models fitted with and without missing values: (A) PST built without missing states, (B) A + imputation, and (C) PST built with missing states.

When a sequence includes a missing state, the procedure searches the PST for the context preceding the missing state and imputes the state according to the conditional distribution associated with the context. The imputation can be done either with the state having the highest probability or randomly (`method = "prob"`), as in the following example:

```
R> SRH.iseq <- impute(SRH.pst.L10, SRH.seq, method = "prob")
R> SRH.iseq[3, ]

  Sequence
3 G2-G2-G2-G2-G2-G2-G2-G2-G1-G2-G2
```

Note that such an imputation is based on very simple assumptions, and a more sophisticated model that considers the non-response mechanism may be required. The imputed state at position 6 can now be predicted by the model. Most of the states after position 6 now have a higher probability (Table 2), because past states can be used for prediction.

However, with the imputation solution, it is not possible to compute the likelihood of the whole learning sample given a model S, and thus, the goodness of fit of two alternative models, which is required for model selection (see Section 4.3). To compute the likelihood of sequences that include missing states, we can add the missing state to the alphabet. This is done by using the `with.missing = TRUE` argument:

```
R> SRH.pst.L10m <- pstree(SRH.seq, nmin = 2, ymin = 0.001,
+    with.missing = TRUE)
R> C99m <- qchisq(0.99, 6 - 1) / 2
R> SRH.pst.L10m.C99 <- prune(SRH.pst.L10m, gain = "G2", C = C99m)
```

The probability of the missing state at position 6, $\mathsf{P}(*|\texttt{G2-G2-G2-G2-G2}) = 0.03$ can now be extracted from the PST:

```
R> query(SRH.pst.L10m, "G2-G2-G2-G2-G2")

 [>] context: G2-G2-G2-G2-G2
        G1     G2       M       B2    B1       *
S1 0.07366 0.8138 0.07958 0.004986 0.001 0.02698
```

The PST now includes contexts with missing states, and the probability of the state at position 7, i.e., $\mathsf{P}(\texttt{G2}|\texttt{G2-G2-G2-G2-G2-*})$, is retrieved as follows:

```
R> query(SRH.pst.L10m, "G2-G2-G2-G2-G2-*")

 [>] context: G2-G2-G2-G2-G2-*
       G1     G2       M    B2    B1      *
S1 0.06825 0.6455 0.03346 0.001 0.001 0.2507
```

Case weights

Longitudinal weights for wave 11 provided in the `SRH` data set make the (remaining) respondents in wave 11 representative of the initial population at wave 1 (1999). With a weighted sample of m sequences having weights $w^j,\ j = 1 \dots m$, the function $N(c)$ in Equation 5 is

$$N(c) = \sum_{j=1}^{m} w^j \sum_{i=1}^{\ell} \mathbb{1}\left[x_i^j, \dots, x_{i+|c|-1}^j = c\right], \tag{17}$$

where $x^j = x_1^j, \dots, x_\ell^j$ is the jth sequence in the training data.

Weights attached to the `SRH.seq` sequence object are used by default in the calculation of next-symbol probability distributions,[13] as shown in the following output:

```
R> cprob(SRH.seq, L = 1, prob = FALSE, with.missing = TRUE)

          G1        G2       M     B2     B1        *   [n]
*   206.624   517.303  104.38  15.99  1.929 132.8937   876
B1    1.192     6.572   11.86  10.61 10.252   0.6424    42
B2    9.503    99.264  140.11  48.42 13.085  10.8811   326
G1 2608.958  2744.144  236.37  11.67  0.000 172.4600  5783
G2 2341.022 11499.921 1662.90 102.29 12.087 425.9847 15862
M   189.513  1660.854 1276.36 140.66 13.750  97.2840  3231
```

Note that column `[n]` lists the (unweighted) number of occurrences $N(c)$ (Equation 6), which are used by the $G_2(c)$ gain function (Equation 9).

4.3. Model selection

The parameters controlling the growing and pruning stages define the shape of the final tree, and thus, the characteristics of the underlying VLMC model. The question is how to set the values of the parameters to obtain the optimal final model. A VLMC model can be optimized by measuring its performance in the particular application for which it is used. For example, if a PST is used for sequence classification, the model having the best classification performance on test data sets is selected (see, for example, Bejerano and Yona 2001). Another approach that is more statistically grounded considers optimization as the search for a true model generating the data, and uses information criteria. This approach is described next.

Growing and pruning parameters

We consider as reference (null model) a memoryless model (i.e., of order $L = 0$) where the probability of each state in a sequence is the frequency of that state in the whole data

[13] The `weighted = FALSE` option can be used to produce unweighted results.

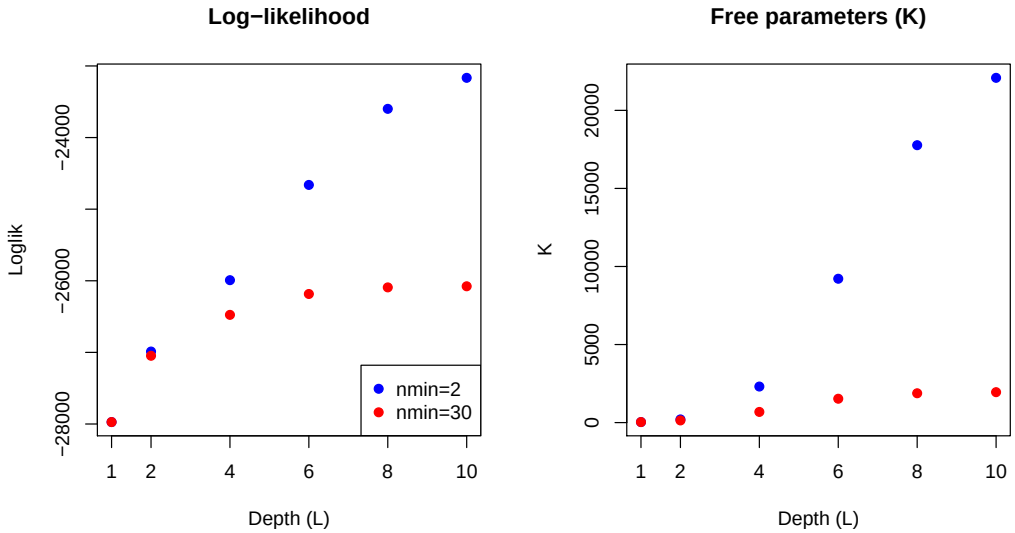

Figure 5: Log-likelihood and number of free parameters of PSTs built with varying L and *nmin* values.

(regardless of the position). The goodness of fit of this model is measured by its log-likelihood (Equation 12), as returned by the generic `logLik()` function:

```
R> SRH.pst.L0m <- prune(SRH.pst.L10m, L = 0)
R> logLik(SRH.pst.L0m)

'log Lik.' -30618 (df=5)
```

Figure 5 shows that increasing the maximal tree depth L and reducing *nmin* increases the number of free parameters,[14] and consequently, the goodness of fit of the model.

Because the shape of the initial tree is determined by the contexts found in the data, the total number of free parameters of the most complex PST ($L = 10$, $nmin = 2$) is significantly smaller than that of a fixed-length Markov chain of order 10, $(|A| - 1)|A|^L = 5 \times 6^{10}$. This tree has as a storage requirement of 26.44 Mb and its building time is around 16 seconds on a 3.6 GHz CPU. The complexity and building time of the initial tree increases with the sequence lengths and the size of the alphabet. When it is not feasible to set L equal to the maximum sequence length, as in our example application, the user will have to use a smaller value (see Rissanen 1983, page 660, on how to set maximal context length).

The deviance of a VLMC model (Equation 11) can be expressed as the sum of position-wise deviances:

$$D(S) = \sum_{i=1}^{\ell} D_i(S) = \sum_{i=1}^{\ell} \sum_{j=1}^{n} -2 \log \mathsf{P}^S(x_i^j), \tag{18}$$

where $\mathsf{P}^S(x_j^i)$ is the predicted state probability at position i in the jth sequence of the learning sample. Figure 6 shows the deviance at each sequence position $i = 1, \ldots, \ell$ for two models

[14] The number of free parameters of a PST S is $(|A| - 1) \times |S|$, where $|A|$ is the size of the alphabet and $|S|$ is the number of nodes (internal nodes and leaves) in the tree. The number of free parameters is $(|A| - 1)$ for each node in the tree, because we have a distribution with $|A|$ probabilities that sum up to 1.

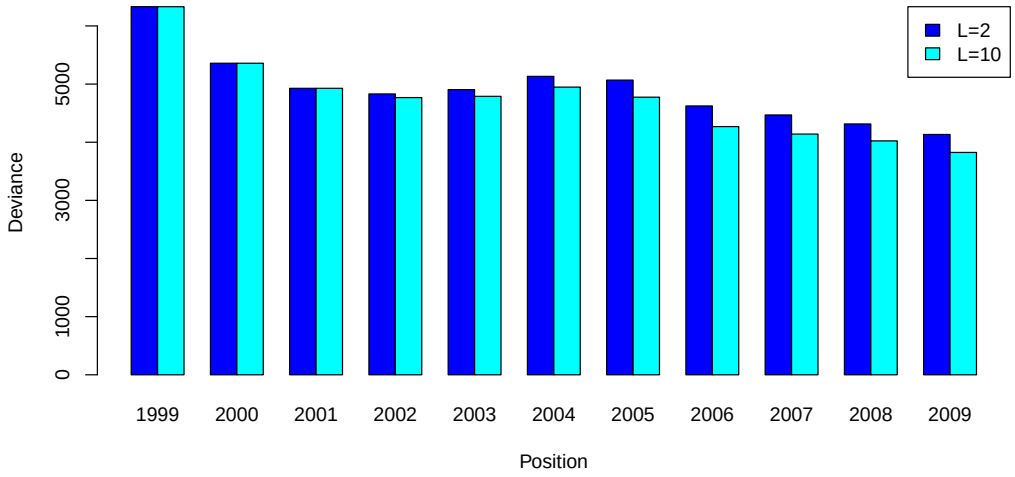

Figure 6: Deviance at positions 1–11 obtained with two unpruned PSTs grown with $nmin = 30$ and depths $L = 2$ and $L = 10$, for the SRH data set.

grown with $nmin = 30$ and depths $L = 2$ and $L = 10$. This figure is obtained with the following code:

```
R> cLL <- NULL
R> for (L in c(2, 10)) {
+    tmp <- prune(SRH.pst.L10m.nmin30, L = L)
+    prob <- predict(tmp, SRH.seq, decomp = TRUE, output = "prob")
+    prob <- -2 * log(prob)
+    cLL <- rbind(cLL, colSums(prob))
+  }
R> barplot(cLL, beside = TRUE, col = c("blue", "cyan"), xlab = "Position",
+    ylab = "Deviance")
R> legend("topright", c("L=2", "L=10"), fill = c("blue", "cyan"))
```

Compared to the model grown with $L = 2$, the model grown with $L = 10$ improves the prediction quality of the states at positions $i > 3$, whereas the probabilities of the states at positions $i \leq 3$ remain unchanged. Further, note that prediction quality is lower at the beginning of the sequences[15] because less memory is available and the state probabilities are retrieved from the internal nodes of the tree. This is particularly true for the first sequence position $i = 1$.

An initial tree grown with $nmin = 2$ and pruned with the G_2 gain function yields the most parsimonious models, with nearly four times less free parameters than trees pruned using the G_1 gain function, but having a lower goodness of fit consequently (Table 3). The log-likelihood of models yielded by the G_1 gain function is nearly the same as that of the unpruned model, because only a few nodes are removed during the pruning stage.

Figure 7 enables us to compare the outcomes of the gain functions used for pruning decisions. The `ppplot()` function displays the probability distributions of two leaves and all their parent

[15] Bejerano and Yona (2001) used a second PST built on the reversed sequences.

L	Lp	$nmin$	Gain	C	LogLik	K	n/K	AIC	AIC_c	BIC
0		2			−30617.6	5	5746.4	61245.1	61245.1	61286.5
10		2			−23165.6	22085	1.3	90501.3	237287.1	273050.7
10		30			−26076.6	1950	14.7	56053.1	56337.2	72171.4
10	10	30	G1	1.05	−30617.6	1945	14.8	65125.1	65407.8	81202.1
10	10	30	G1	1.20	−30617.6	1940	14.8	65115.1	65396.3	81150.7
10	7	2	G2	5.54	−26445.9	510	56.3	53911.7	53930.2	58127.3
10	7	2	G2	7.54	−26588.5	340	84.5	53856.9	53865.1	56667.3

Table 3: PSTs including missing values. The following parameters are listed: depth of the initial tree (L); depth of the pruned tree (Lp); gain function (Gain); cutoff (C) used for pruning; likelihood (LogLik); number of free parameters (K); AIC, AIC_c, and BIC criteria.

nodes (suffixes) in the tree built with $nmin = 2$. When a gain function and a vector of pruning cutoffs are given as optional arguments, the graphic displays the outcomes of the gain function.[16] The first graph in Figure 7 is obtained with the following command:

```
R> node <- rep("G2", 10)
R> alpha <- c(0.05, 0.01, 0.001)
R> C.list <- qchisq(1 - alpha, df = 6 - 1) / 2
R> ppplot(SRH.pst.L10m, node, gain = "G2", C = C.list)
```

In the lower part of each plot, the red circles indicate that starting from the leaf, the nodes do not provide an information gain, whereas the green circles indicate that the nodes provide an information gain in comparison with their parents.

Model selection using information criteria

More complex trees have higher goodness of fit (Table 3), but over-fitting can occur. Therefore, the question is how to achieve the best trade-off between goodness of fit and model complexity. For a fixed-length Markov chain, model selection involves determining its optimal order, which can be realized according to the Akaike information criterion (AIC, Tong 1975)

$$\text{AIC}(S) = -2\,\text{logLik}(S) + 2K \tag{19}$$

or the Bayesian information criterion (BIC, Katz 1981)

$$\text{BIC}(S) = -2\,\text{logLik}(S) + K \times \log(n), \tag{20}$$

where K is the number of free parameters in the model S and logLik is the log-likelihood (Equation 12).

In the case of VLMCs, the memory length (order) can be different for each context, and it is not feasible to test all possible models, which are indeed sub-models of a fixed (high)-order Markov chain. One solution is to optimize with respect to the pruning cutoff,[17] i.e., to select

[16] Note that what is displayed is not the outcome of the pruning stage for the whole tree. The pruning of a leaf is based on a single outcome of the gain function, but the pruning of an internal node depends on the information gain of all its offspring.

[17] An alternative is to use a bootstrap method.

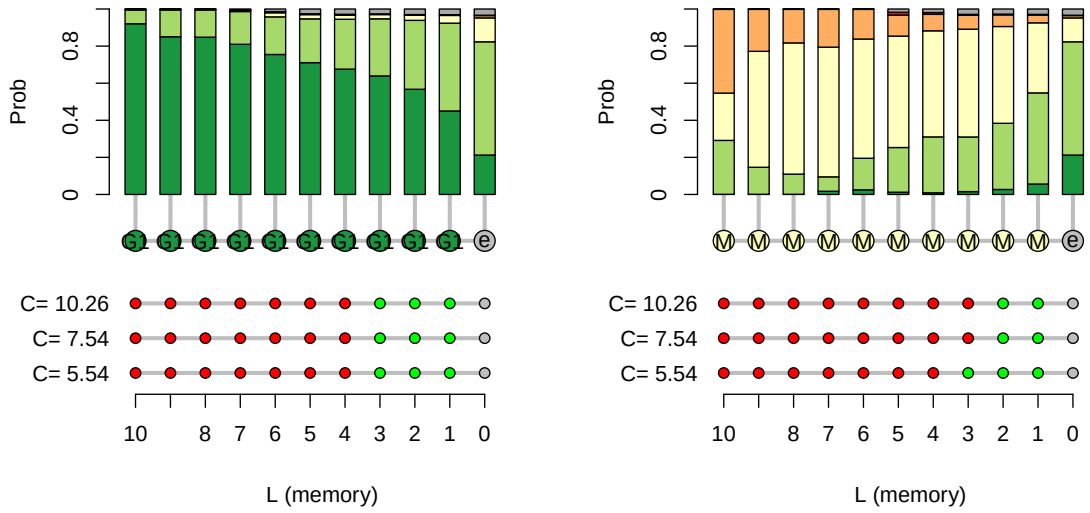

Figure 7: Tree branch corresponding to two contexts and results of the gain function testing for divergence between successive probability distributions.

the cutoff value yielding the best compromise between goodness of fit and model complexity. Mächler and Bühlmann (2004) proposed the use of AIC.[18] In the framework of the context algorithm (using the G_2 gain function), a cutoff estimated using AIC aims to minimize the Kullback-Leibler (KL) divergence between the true underlying process and the fitted VLMC model.

When comparing models with a significant number K of parameters relative to the sample size n, a modified AIC criterion for small samples should be used (Burnham and Anderson 2004)

$$\mathrm{AIC}_c(S) = -2\,\mathrm{logLik}(S) + 2K + \frac{2K(K+1)}{n-K-1}. \tag{21}$$

The rule is to use AIC_c instead of AIC when $n/K \leq 40$ for the model with the largest value of K, which is the case for our example (Table 3). Here, n is the number of observed sequence positions, which is returned by the generic `nobs()` function:

```
R> nobs(SRH.pst.L10m)

[1] 28732
```

The difference between AIC_c and AIC increases with the number of parameters, and it becomes very large for the unpruned tree grown with $nmin = 2$. In this case, using the AIC, AIC_c, or BIC criterion would lead to the selection of the same model, obtained by pruning with cutoff $C_{\alpha=0.01} = 7.54$.

The **PST** package includes a model selection function based on AIC, AIC_c, or BIC, which can return either the selected PST or the statistics for each model. In the following example, we provide a list of cutoff values for the G_2 gain function, ranging from $\alpha = 0.05$ to $\alpha = 0.001$:

[18] The AIC or BIC value for a PST can be obtained with the generic `AIC()` and `BIC()` R functions.

```
R> alpha <- c(0.05, 0.04, 0.03, 0.02, 0.01, 0.001)
R> C.list <- qchisq(1 - alpha, df = 6 - 1) / 2
R> C.tuned <- tune(SRH.pst.L10m, gain = "G2", C = C.list, output = "stats",
+    criterion = "AICc")
```

The output below shows that the model having the lowest AIC_c value, and thus, the highest support (***), is obtained with $\alpha = 0.01$:

```
R> C.tuned
```

```
  Model      C Nodes Leaves Freepar  AICc Support
1     1  5.535    51     51     510 53930
2     2  5.822    49     49     490 53914
3     3  6.187    41     45     430 53880
4     4  6.694    39     42     405 53871       *
5     5  7.543    29     39     340 53865     ***
6     6 10.258    13     25     190 53932
```

By letting $\Delta_i = \text{AIC}_i - \text{AIC}_{\text{min}}$, models with $\Delta_i \leq 2$ can be considered as having substantial support, whereas models with $\Delta_i \geq 10$ can be considered as having essentially no support[19] (Burnham and Anderson 2004).

5. Sequence analysis using a probabilistic suffix tree

Once a VLMC model is learned from a learning sample as described in the previous section, the corresponding PST can be used to gain insights into the process generating the observed sequences. The process is described by examining the parameters of the model, i.e., the conditional probability distributions stored in the nodes of the tree. In this section, we review the tools provided by the **PST** package for this purpose. In addition to elucidating the generating process, the sequence prediction feature of PSTs can be used in numerous applications. Here, we illustrate some of these applications, such as extracting typical sequences and outliers from the learning sample, as well as the use of PSTs for pattern mining with the dedicated functions provided by the package.

5.1. Exploring the generating process

The selected model is stored in the `SRH.pst` object and its first two levels are shown in Figure 8. The PST is the representation of the underlying process that generates the data. The `SRH` sequence data set exhibits high-order dependencies, because the maximal depth of the selected model is 7 (Table 3, page 19). This means that some contexts representing the SRH for the past seven waves provide information to predict the next health status. We can identify them by displaying all the nodes at depth 7:

```
R> nodenames(SRH.pst, L = 7)
```

[19] Models having $\Delta_i \leq 2$ are tagged with ** and models having $2 < \Delta_i < 10$ are tagged with * in the output of the `tune` function.

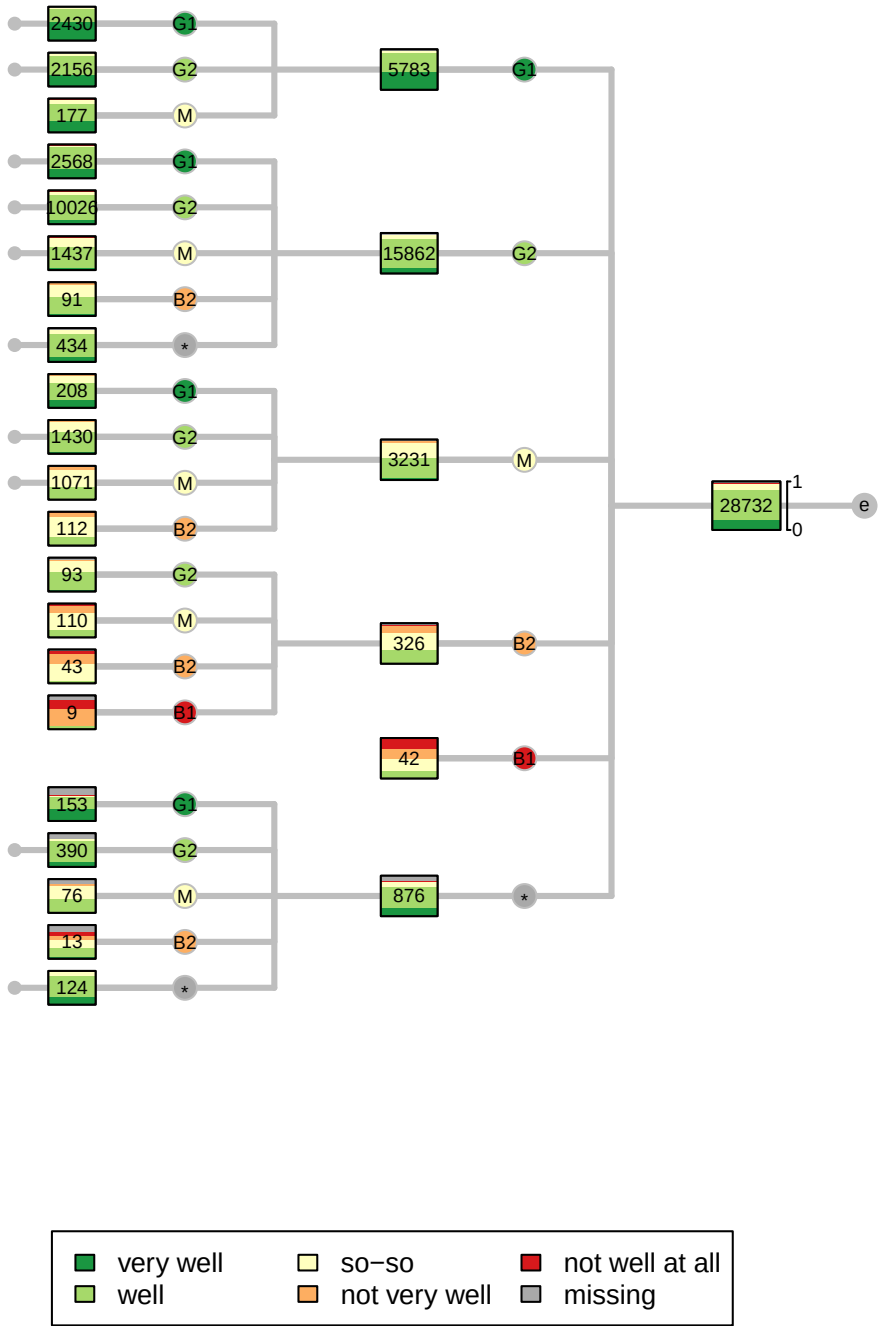

Figure 8: First two levels of the PST learned from the SRH data set (including missing states) having the lowest AIC_c value.

```
[1] "G2-G1-G2-G2-G2-G2-G2" "M-G2-G2-G2-G2-G2-G2"
```

Visualization tools enable us to gain a better understanding and describe the model. In addition to the graphical display of the tree itself, which can become increasingly difficult to read when displaying high-order branches, the **PST** package provides several plotting methods to

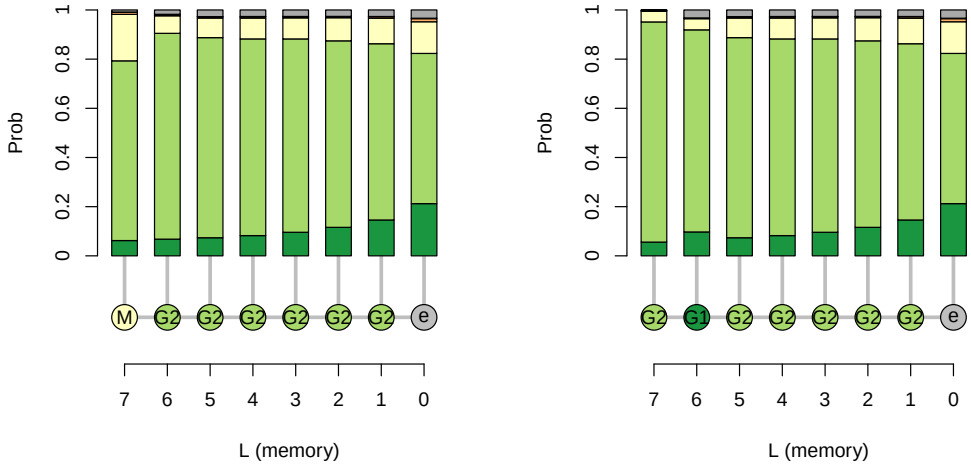

Figure 9: Tree branch corresponding to two contexts.

represent selected parts of the tree, such as `ppplot()`. Figure 9, obtained with the command below, shows the two leaves of order 7 and all their parent nodes:

```
R> par(mfrow = c(1, 2))
R> ppplot(SRH.pst, "M-G2-G2-G2-G2-G2-G2")
R> ppplot(SRH.pst, "G2-G1-G2-G2-G2-G2-G2")
```

One can see how increasing the memory length to account for additional past states can modify, in certain cases, the next-symbol probability distribution, thereby providing an information gain. Using $\mathsf{P}(\sigma|\texttt{G2-G2-G2-G2-G2-G2})$ instead of $\mathsf{P}(\sigma|\texttt{M-G2-G2-G2-G2-G2-G2})$ would cause a significant amount of information to be ignored. The result can be expressed as follows: Knowing that an individual answered "so-so" to the SRH question at time $i-7$ (seven waves ago) increases the probability that he answers "so-so" at time i, even if he answered "good" during the six previous waves $i-6,\ldots,i-1$.

Mining contexts

To gain further insights into the generating process, the **PST** package provides the context mining function `cmine()`, which searches the tree for nodes fulfilling certain characteristics. For example, we would like to know which contexts are highly likely to be followed by a given state. In the following example, we search for all contexts yielding a probability of at least $pmin = 0.5$ for the state `G1` (very good health).

```
R> cmine(SRH.pst, pmin = 0.5, state = "G1")

 [>] context: G1-G1
      G1     G2       M       B2    B1       *
S1 0.5672 0.3716 0.02751 0.002683 0.001 0.03002

 [>] context: G1-G1-G1
     G1     G2       M       B2    B1       *
S1 0.639 0.3081 0.02289 0.002762 0.001 0.02619
```

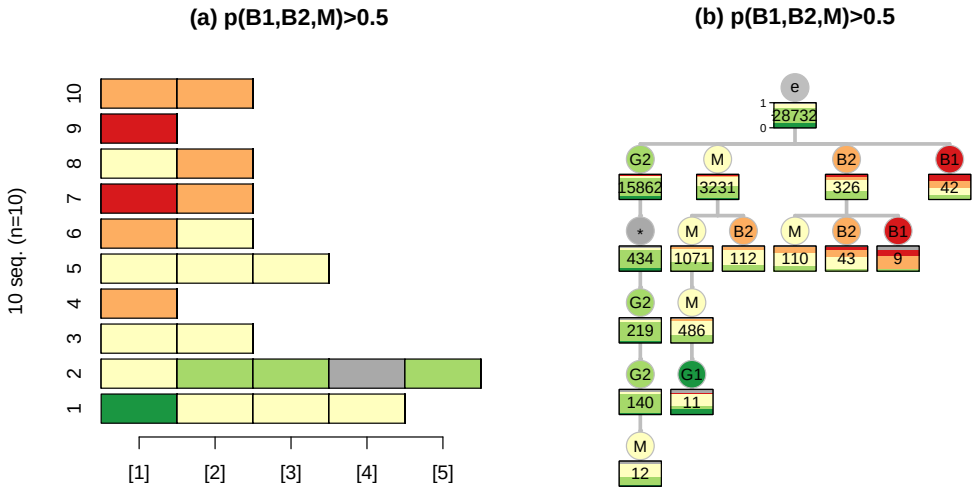

Figure 10: Contexts associated with a probability $\mathsf{P}(\sigma \in \{\texttt{B1,B2,M}\}) > 0.5$: (a) contexts sorted according to the probability $\mathsf{P}(\sigma \in \{\texttt{B1,B2,M}\})$ and (b) initial PST where only the selected contexts are retained.

Only contexts of at least two previous states of very good SRH lead to $\mathsf{P}(\sigma = \texttt{G1}) \geq 0.5$. Thus, cumulated previous states of good and very good SRH increase the probability of good and very good SRH.

One can also mine for contexts corresponding to a minimum or maximum probability for several states together. In the following example, we search for contexts c associated with a probability $\mathsf{P}(\sigma \in \{\texttt{B1, B2, M}\}|c) > 0.5$, i.e., high probability of medium or lower SRH (as compared to the overall probability of these three states, which is $\mathsf{P}(\sigma \in \{\texttt{B1,B2,M}\}) = 0.14$):

```
R> cm2 <- cmine(SRH.pst, pmin = 0.5, state = c("B1", "B2", "M"))
```

The output of the `cmine()` function can be plotted (Figure 10a). By using the `as.tree = TRUE` option, the `cmine()` function returns a tree where only the selected nodes (together with their parents) are retained (Figure 10b).

5.2. Typical sequences and outliers

Predicting the sequences in the `SRH.seq` learning sample with the fitted PST yields the average log-loss distribution shown in Figure 11a. Sequences having a low likelihood, i.e., a large average log-loss, are located in the right queue, whereas the most likely (typical) sequences are located in the left queue of the distribution. Figures 11b and 11c show the 15 unique sequences (1.1% of the 1901 unique sequences) with the highest and lowest average log-loss. The `pqplot()` function represents the log-loss for each state of sample sequences as a series of bar plots (Figure 12). Sequences containing transitions from good to poor and very poor health statuses, as well as missing values, have a low likelihood. In contrast, the most likely pattern (index 1 in Figure 11c) is a stable sequence of good health (G2).

An outlier is defined as a sequence that is unlikely to have been generated by the model. Outliers can be extracted by setting a threshold in the prediction quality distribution such that sequences having scores below the threshold will be considered as outliers. Existing applications propose two solutions that are adapted to different settings.

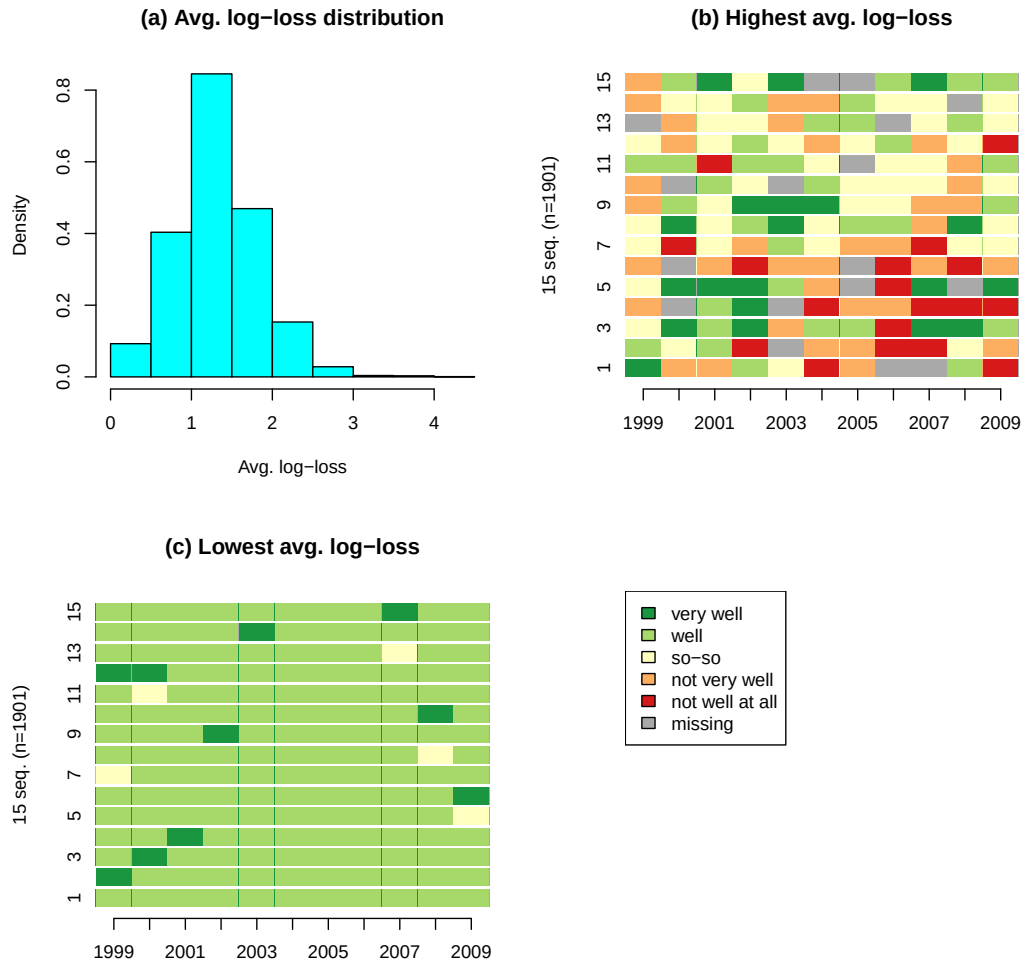

Figure 11: Distribution of the average log-loss for sequences in the SRH.seq data, and sequences having highest and lowest average log-loss.

The first application (Sun, Chawla, and Arunsalam 2006) uses a learning sample of sequences known to belong to a protein family. A VLMC is learned from this sample, and the model is used to compute the score[20] $SIMn(S, x) = \log \frac{\mathsf{P}^S(x)}{\ell}$ of query sequences. If $SIMn < t$, where t is a threshold computed using the Bienaymé-Chebyshev inequality, it is assumed that the query sequence does not belong to the protein family. In the second application (Low-Kam, Laurent, and Teisseire 2009), with a similar setting as in our example, the learning sample includes potential outliers. Here, Bennett's inequality is used to define the threshold t.

5.3. Pattern mining

When selecting sequences with the highest or lowest average log-loss, we select sequences whose average state probability is the lowest or highest, respectively. However, as seen previously, the first states in the sequences are usually less well predicted by the model because less memory is available for prediction. A sequence may not appear as very likely if its first

[20]The score $SIMn(S, x)$ is in fact equivalent to the average log-loss since $SIMn(S, x) = -\text{logloss}(S, x)$.

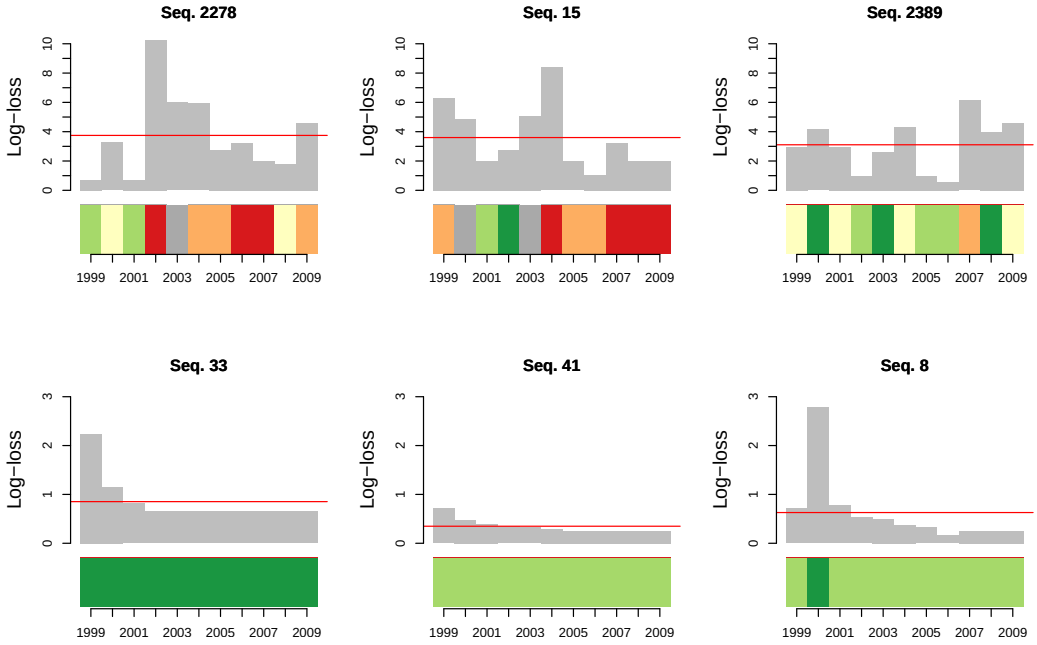

Figure 12: Position-wise log-loss for sample sequences having low and high likelihood. The red line indicates average log-loss.

state has a low relative frequency,[21] even if the model predicts high probabilities for the states at higher positions. The **PST** package provides the `pmine()` function for advanced pattern mining with user-defined parameters.

Using lags

The `pmine` function is controlled by the `lag` and `pmin` arguments. By setting $lag = 2$ and $pmin = 0.40$, we select all sequences with average[22] state probability at positions $lag+1, \ldots, \ell$ above $pmin$. Instead of considering the average state probability at positions $lag + 1, \ldots, \ell$, it is possible to select frequent patterns that do not contain any state with probability below the threshold. Thus, sequences having many states with high probability are not selected when they contain one or several states with low probability. The command is stated below and selected sequences are shown in Figure 13a:

```
R> pmine.A <- pmine(SRH.pst, SRH.seq, pmin = 0.40, lag = 2, average = FALSE)
```

Some patterns not appearing in the most likely sequences are selected, for example `M / 12`. Actually, the probabilities of the first and second `M` (so-so) state, $\mathsf{P}(\mathtt{M}) = 0.13$ and $\mathsf{P}(\mathtt{M}|\mathtt{M}) = 0.38$, respectively, are below the threshold. However, all the following states are predicted with $\mathsf{P}(\mathtt{M}|\mathtt{M\text{-}M}) = 0.52$ (`M-M` is the longest suffix of `M-M-M` and thus, of `M-M-M-...-M` in the tree), which is above the threshold and results in the selection of the pattern. In other words, once a spell (two successive states) of intermediate SRH is entered, the probability of remaining in this status increases and becomes the most likely next state.

[21] The probabilities in the root node used to predict the first state are computed from the distribution of the states in the whole data.

[22] The geometric mean is used.

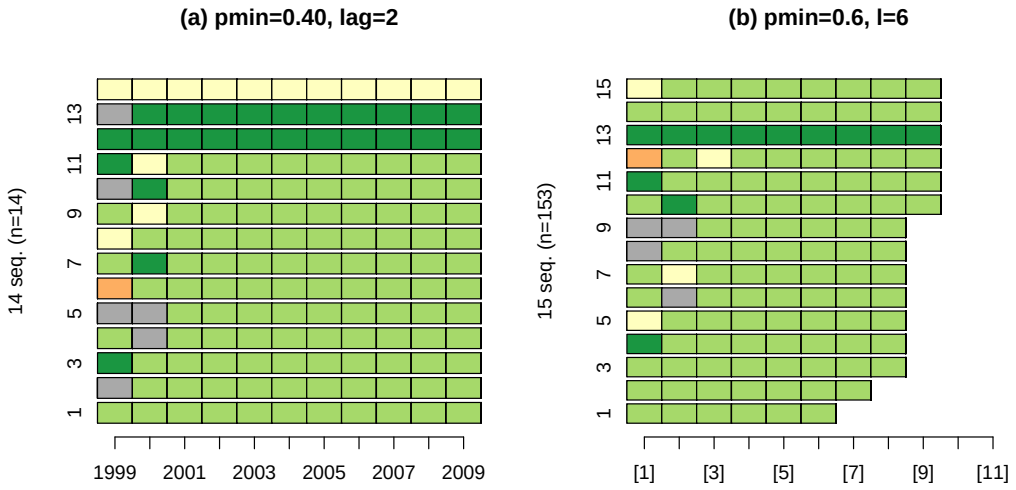

Figure 13: Outcomes of pattern mining: (a) sample sequences with $pmin = 0.4$ and $lag = 2$ and (b) patterns of length $l = 6$ with $pmin = 0.6$.

Frequent subsequences

It is also possible to mine the sequence data for frequent patterns of length $\ell_j < \ell$, regardless of their position in the sequence. By using the `output = "patterns"` argument, the `pmine()` function returns the patterns (as a sequence object) instead of the whole set of distinct sequences containing the patterns. Because the probability of a pattern can vary with the context (previous states), the returned subsequences also contain the context preceding the pattern. In the next example, we extract all patterns of length $\ell_j = 6$ with $pmin = 0.6$

```
R> pmine.B <- pmine(SRH.pst, SRH.seq, l = 6, pmin = 0.6, output = "patterns")
```

The first 15 selected patterns (among 153) and their contexts are shown in Figure 13b. The patterns are the last six states in the displayed bars. The first (index 1) pattern, `G2-G2-G2-G2-G2-G2`, with $pmin = 0.6$ is not preceded by any context, meaning that it appears at position $1, \ldots, 6$.

5.4. Generating sequences

Because a PST represents the model that generates the observed sequences, it can be used to generate artificial sequences. This feature can be used to check whether the model accurately represents the training data. In addition, it can be used for simulation purposes and to compute between-model distances, as explained in Section 6 below. Sequences are built by generating the states at each successive position. The process is similar to sequence prediction (see Section 3), except that the retrieved conditional probability distributions provided by the PST are used for generating a symbol rather than computing the probability of an existing state. In the following example, we generate a set of 1000 sequences of length $\ell = 11$ (the same length as the observed sequences):

```
R> gen1.seq <- generate(SRH.pst, l = 11, L = 10, n = 1000)
```

	G1	G2	M	B2	B1	*
Position 1	0.32	0.49	0.12	0.01	0.00	0.05
Root node	0.21	0.61	0.13	0.01	0.00	0.03

Table 4: State probability distribution at position 1 and zeroth-order probability stored in the root node.

Note that the model also generates missing states. The two most frequent sequences in the artificial set (Figure 14) are the same as those in the original data (Figure 4, page 14).

However, the state distributions at each sequence position between the real and the generated data sets are different, especially at the first positions (Figure 14). Indeed, the probability distribution retrieved from the root node of the PST used to generate the starting state of the sequences is based on the probabilities computed over all positions of the sequence data set. This distribution is somewhat different from the state distribution at the first position (Table 4).

If we use the `p1` argument to set the first state probability distribution to be equal to the one observed in the data,[23] we get a much closer state distribution (Figure 14c).

Geez. The similarity between segmented PST and clustering

6. Segmented models and comparison of PSTs

Thus far, we have assumed that a single model generated all the sequences in the learning sample. For our illustrative data, this means that the SRH trajectories of the whole Swiss resident population aged between 20 and 80 years in 1999 were generated by the same model. However, especially when analyzing sequences in the social sciences, we expect the model to vary across subgroups (strata) in the population defined, for example, with some socio-economic or socio-demographic characteristics.

Thus, in such cases, we need a method for considering population heterogeneity. This issue has been addressed for fixed-length Markov chains in the social sciences literature (Spilerman 1972; Singer and Spilerman 1973; Langeheine and Van de Pol 1990). For VLMCs, we propose two approaches that are explained in this section. One solution is to fit a separate model for each *subgroup* in the population, and then compare the models. In the second part of this section, we describe a method for measuring the distance between two distinct VLMC models. The second solution provided by the **PST** package is to compute the next-symbol probability distributions for each of the groups separately and store them in a single *segmented* PST, for which we can measure the prediction performance. This approach is described below.

6.1. Learning a segmented PST

Going back to our illustrative data, we can expect, in particular, that age at the beginning of the observed trajectory will influence the pattern. At the aggregate level, health status declines slightly but regularly with age and usually starts to decline more significantly at older ages. Here, we consider the following three age groups (119 sequences were removed from the original data set owing to missing information on age),

[23] Note that we have to include the missing state in the probability distribution because it is part of the PST's alphabet.

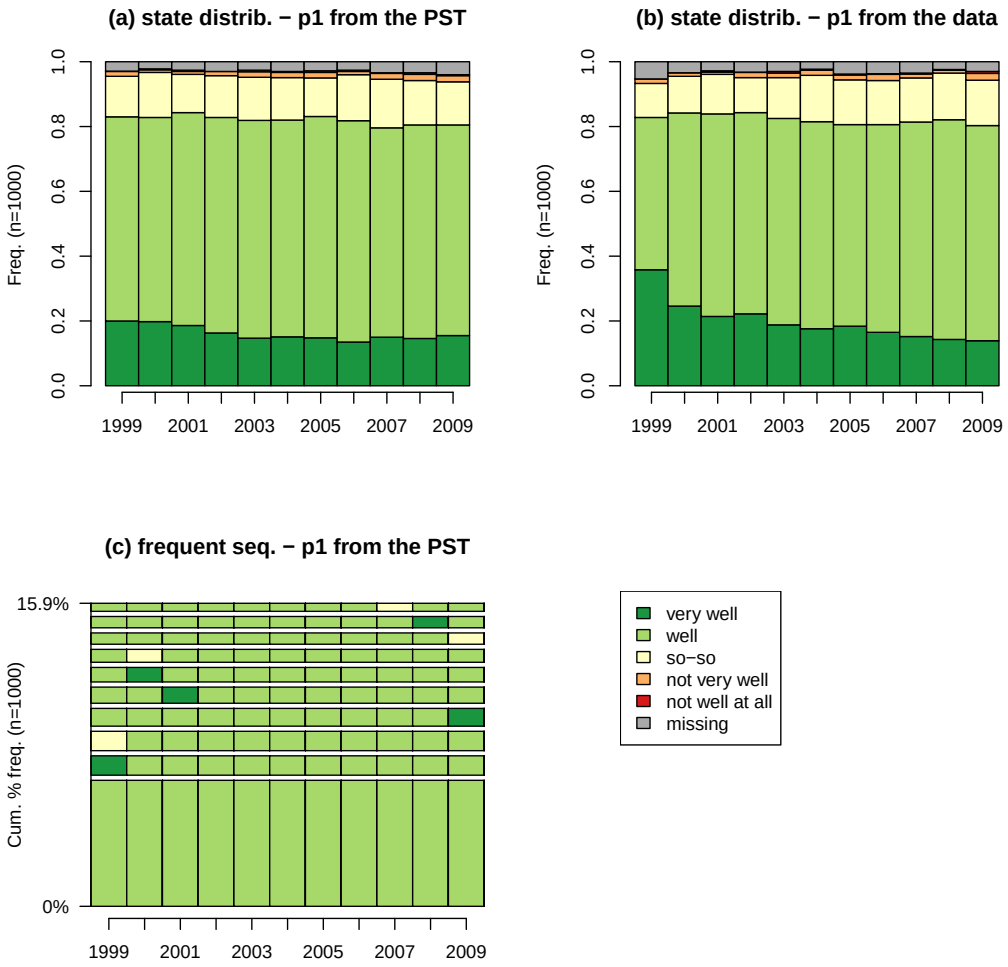

Figure 14: Generating sequences: Position-wise state distribution (a) in the generated data set and (b) in the data set generated with the observed first state probability distribution; (c) the 10 most frequent sequences in the generated data set.

```
R> table(gage10)

gage10
20-39 40-59 60-79
  984  1085   424
```

and we hypothesize that distinct models have generated the sequences in each of the groups. Instead of separately learning one model for each group, the **PST** package allows for a segmented PST to be built. This is achieved by providing the vector of group membership as the `group` argument:

```
R> SRH.pst.L10mg <- pstree(SRH.anm.seq, group = gage10, with.missing = TRUE,
+    nmin = 2, ymin = 0.001)
R> SRH.pst.L10mg.C99 <- prune(SRH.pst.L10mg, gain = "G2", C = C99m)
```

A segmented PST consists of one independent PST for each group, and each submodel is pruned separately during the pruning stage. It is always possible to extract the model for one particular group with the `subtree()` function.

One advantage of a segmented PST is that the conditional probability distributions for each of the submodels can easily be compared. In the graphical representation (Figure 15), a node contains up to three probability distributions. In some nodes, the probability distribution for one or several groups may be missing because contexts are either not found in the sequences of this group or pruned out afterward. The root node indicates that the probability of observing good or very good SRH decreases with age (the distribution for the youngest age group is the left-most one).

Single nodes can be compared by means of the `cplot()` function. One can see in Figure 16 that except for the node `B1`, first-order next-symbol probability distributions evolve in a similar way with age, with an increasing probability of worse health statuses.

Segmented vs. unsegmented model

Considering the segmented PST as a single model, we can measure the degree of improvement in its prediction of the whole sequence data set as compared to the unsegmented model. To predict a set of sequences with a segmented PST, it is necessary to provide a vector with the group membership of each sequence:

```
R> SRH.prob.gage <- predict(SRH.pst.L10mg.C99, SRH.anm.seq, group = gage10)
```

A sequence belonging to group g is then predicted with the submodel learned from the sequences of this group.

The segmented and unsegmented models are learned on the same data and can be compared by means of information criteria. The likelihood of the segmented model is lower, and it contains more parameters than the unsegmented model. This leads to a significantly higher AIC (and AIC_c) value, and thus, to the rejection of the segmented model.

```
R> AIC(SRH.pst.L10m.C99, SRH.pst.L10mg.C99)
```

```
                    df   AIC
SRH.pst.L10m.C99   325 51112
SRH.pst.L10mg.C99 405 51438
```

6.2. Comparison of PSTs

Now, we separately consider each of the three age group models and compare them by using multiple predictions, i.e., by predicting the sequences with the three models and comparing the resulting scores. Let S_1, S_2, and S_3 be the PST learned on sequences belonging to individuals in the first (20–39), second (40–59), and third (60–79) age groups, respectively. Model S_1, for example, is extracted from the segmented model with:

```
R> SRH.pst.g1 <- subtree(SRH.pst.L10mg.C99, group = 1)
```

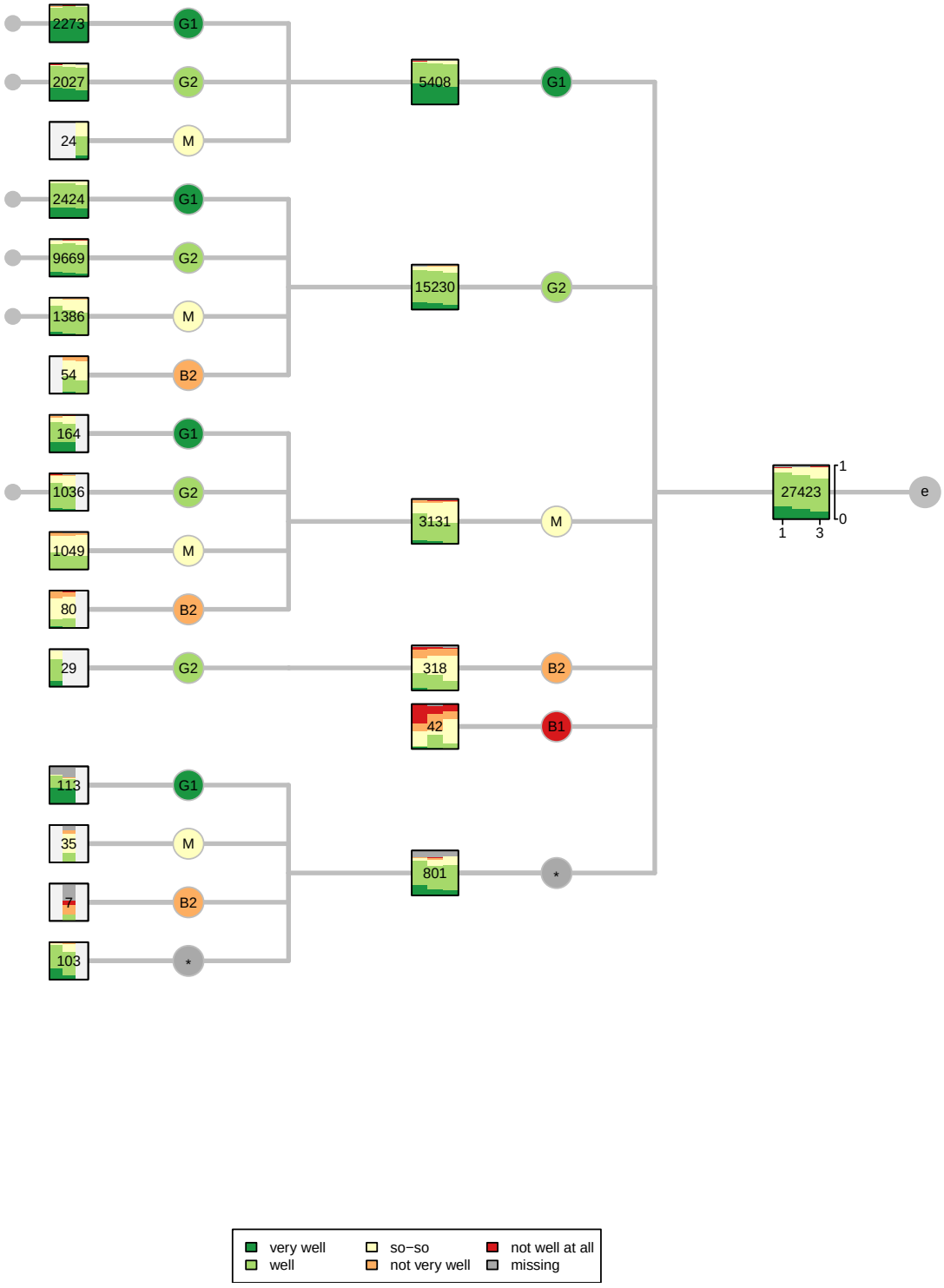

Figure 15: Self-rated health; PST segmented by age group.

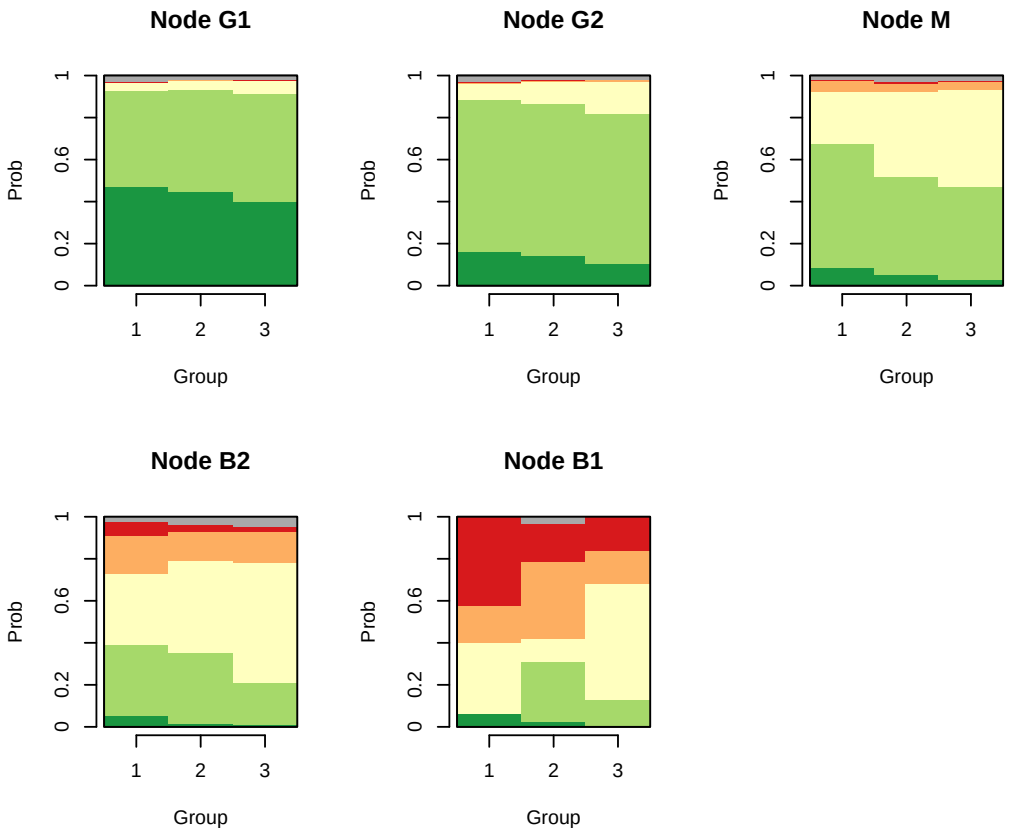

Figure 16: Probability distribution for single nodes; PST segmented by age group.

Sequence	Model S_1	Model S_2	Model S_3
G1/11	0.93	0.94	1.08
G2/11	0.37	0.38	0.41
M/11	1.52	1.11	1.03

Table 5: Average log-loss for sample sequences computed with a PST segmented by age group.

If we use a segmented PST as an argument of the `predict()` function without passing a `group` argument, the output consists of multiple predictions, i.e., each sequence is predicted with each of the submodels. Table 5 lists the results for three sample sequences. For each of the three sequences, the score regularly evolves with the age groups. The probability of the sequence `G1 / 11` (very good SRH during the whole observation period) decreases with age, and thus, the average log-loss increases. In contrast, the likelihood of a sequence of fair (so-so) SRH during the 11 waves increases with age. The likelihood of the sequence `G2 / 11` (good health during 11 waves) slightly declines with age, but this sequence remains the most likely, regardless of the age group.

Figure 17 shows detailed prediction results for two sequences for which the difference among the three prediction scores is the highest. The probability of status `M` (fair health) at the beginning of sequence 2516 is higher with model S_3 (age group 60–79), which is also the case for the status `M` that immediately follows the status `G1` (very good health) at position 5.

Now, we would like to make an overall comparison of the models. One way to assess how

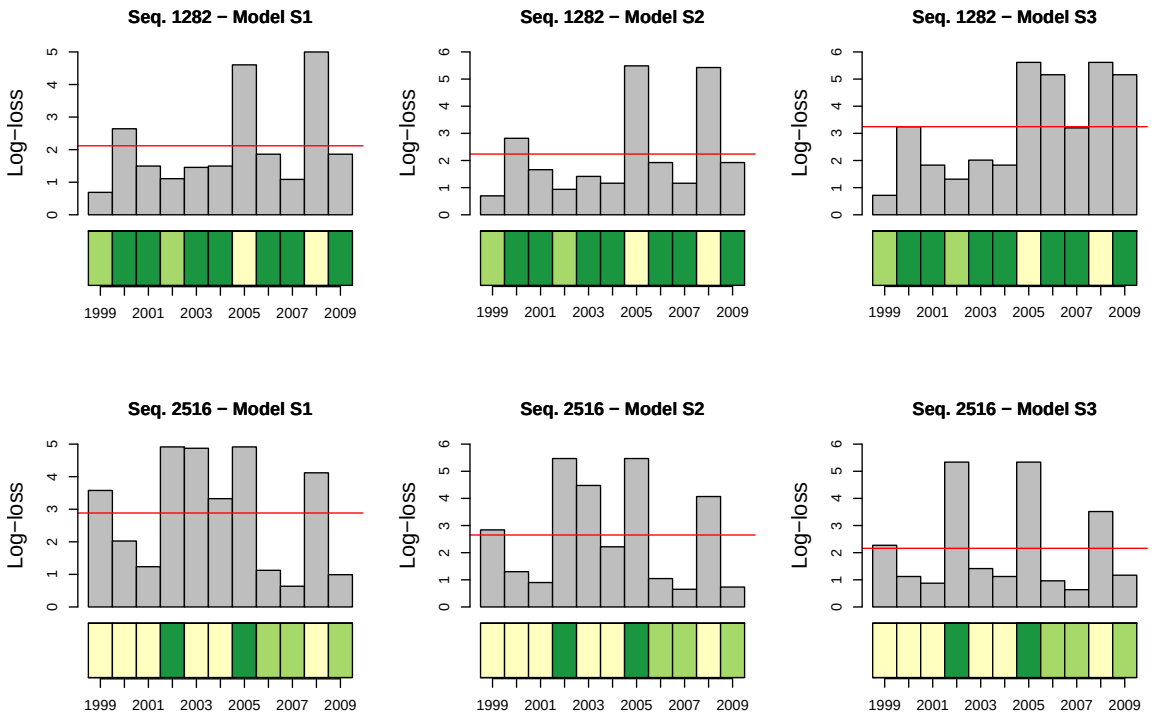

Figure 17: Log-loss for sample sequences computed with PST for each age group.

different a process is in two groups or distinct populations is to define a divergence measure between the two models. A possible approach is to compare the probability distributions in each of the common nodes in the PSTs (Largeron 2003; Mazeroff, Gregor, Thomason, and Ford 2008). Another approach, which we propose below, is based on the comparison of the underlying probability distributions for the complete set of sequences of length ℓ (Equation 13) represented by each of the models, and it uses the ability of a VLMC model to generate sequences.

Therefore, we use a probabilistic divergence[24] measure originally proposed by Juang and Rabiner (1985) for the comparison of two (hidden) Markov models S_A and S_B:

$$d(S_A, S_B) = \frac{1}{\ell}[\log \mathsf{P}^{S_A}(x) - \log \mathsf{P}^{S_B}(x)] = \frac{1}{\ell} \log \frac{\mathsf{P}^{S_A}(x)}{\mathsf{P}^{S_B}(x)}, \tag{22}$$

where $x = x_1, \ldots, x_\ell$ is a sequence generated by model S_A, $\mathsf{P}^{S_A}(x)$ is the probability of x given model S_A, and $\mathsf{P}^{S_B}(x)$ is the probability of x given model S_B. The ratio between the two sequence likelihoods is a measure of how many times the sequence x is more likely to have been generated by S_A than by S_B.

As the number n of generated sequences on which the measure is computed (or the length of a single sequence) approaches infinity, the expected value of $d(S_A, S_B)$ converges to $d_{KL}(S_A, S_B)$ (Falkhausen, Reininger, and Wolf 1995; He, Kwong, Man, and Tang 2000), i.e., the Kullback-Leibler (KL) divergence (also called information gain) used in information theory to measure the difference between two probability distributions.

[24]The word *distance* is used in the original article, but here we use *divergence* instead.

The `pdist()` function of **PST** uses the following procedure to compute the divergence between two PSTs S_A and S_B:

- Generate a random sample of n sequences (of length ℓ) with model S_A using the `generate` method (see Section 5).
- Predict the sequences with S_A and S_B.
- Compute
$$d_i(S_A, S_B) = \frac{1}{\ell}[\log \mathsf{P}^{S_A}(x_i) - \log \mathsf{P}^{S_B}(x_i))], \; i = 1, \ldots, n. \tag{23}$$
- The expected value
$$\mathsf{E}(d(S_A, S_B)) \tag{24}$$
is the divergence between S_A and S_B, and it is estimated as
$$\hat{\mathsf{E}}(d(S_A, S_B)) = \frac{1}{n}\sum_{i=1}^{n} d_i(S_A, S_B). \tag{25}$$

Because the calculation of the divergence is based on a randomly generated sample, with each calculation, the obtained expected value varies around the true value that would be obtained with a sample of infinite size. Thus, the number n of generated sequences influences the precision of the measure.[25]

The divergence $d(S_A, S_B)$ is not symmetric, i.e., sequences generated by model S_A can be better predicted by S_B than sequences generated by model S_B can be predicted by S_A (and vice-versa). A symmetric version of the measure can be computed as proposed by Juang and Rabiner (1985)
$$d_{\text{sym}}(S_A, S_B) = \frac{d(S_A, S_B) + d(S_B, S_A)}{2}. \tag{26}$$

In the following example, we compute the pairwise probabilistic divergence between submodels S_1 (age group 20–39) and S_2 (age group 40–59) using 5000 sequences (default value) of length $\ell = 11$ generated by each of the models:

```
R> d1_2 <- pdist(SRH.pst.g1, SRH.pst.g2, l = 11, method = "cp",
+    symetric = TRUE, output = "mean")
```

The pairwise divergences are listed in Table 6. The divergence $d(S_1, S_2)$ is similar to $d(S_2, S_3)$, while the divergence between models S_1 and S_3 is around twice as much as that between models S_2 and S_3. This suggests that the process that generates SRH sequences evolves with age. However, the obtained distance measure does not provide information about the statistical significance of the divergence between two models. Therefore, a statistical test of the difference between two models is needed. One such test, which compares the probability distributions in the nodes of the two PSTs, has been proposed by Largeron (2003).

[25] With the default value $n = 5000$, a simulation that computes the distances 100 times yields a coefficient of variation ranging from 2.2% to 4.3% depending on the two considered models. With $n = 10000$, the coefficient of variation ranges from 1.7% to 2.8%.

	Model S_1	Model S_2	Model S_3
Model S_1	0.000	0.039	0.085
Model S_2	0.039	0.000	0.041
Model S_3	0.085	0.041	0.000

Table 6: Symmetric pairwise divergence matrix for models S_1 (age group 20–39), S_2 (age group 40–59), and S_3 (age group 60–79).

7. Conclusion

The features of the **PST** package described in this article provide a framework for analyzing categorical sequences with variable-length Markov chains (VLMCs). The package includes original tools for model selection, model visualization, and pattern mining. The procedure for learning VLMCs can account for weights in sequence data sets as well as missing values. These features make the package well suited for the analysis of social science longitudinal data sets stemming from individual-level longitudinal surveys.

The package provides an alternative to the dissimilarity-based methods provided by the **TraMineR** package (Gabadinho *et al.* 2011a). The use of VLMCs is based on the modeling of the process generating the observed sequences. Whole sequences can then be compared by calculating the likelihood that they were generated by the VLMC model. This allows for numerous data mining tasks based on sequence likelihood. One example is the extraction of typical patterns from sequence databases, an important objective of sequence analysis. This task, which requires nontrivial heuristic procedures when using pairwise dissimilarities (Gabadinho, Ritschard, Studer, and Müller 2011b), can also be achieved with sequence prediction. Another important and promising application is the analysis of the influence of covariates on the patterns. This can be realized with **PST** by fitting segmented PSTs and computing the probabilistic divergence between models.

The **PST** package enables us to explore the possibilities of categorical sequence analysis using VLMCs, and it can facilitate the development of new methods. In addition to the simple examples presented in this article, the possible applications of PSTs are numerous and include, in particular, unsupervised sequence classification (clustering). Although the scope of this paper is limited to the analysis of stationary processes, the modeling of non-stationary processes is available as an experimental feature of the package. This feature increases the scope for applying VLMCs to many real-world processes that are not realistically modeled with stationary models.

Acknowledgments

This publication benefited from the support of the Swiss National Centre of Competence in Research LIVES – Overcoming vulnerability: Life course perspectives, which is financed by the Swiss National Science Foundation. The authors are grateful to the Swiss National Science Foundation for its financial assistance. The authors also acknowledge Reto Bürgin for his insightful comments.

References

Abbott A (1995). "Sequence Analysis: New Methods for Old Ideas." *Annual Review of Sociology*, **21**, 93–113. doi:10.1146/annurev.soc.21.1.93.

Abbott A, Forrest J (1986). "Optimal Matching Methods for Historical Sequences." *Journal of Interdisciplinary History*, **16**, 471–494. doi:10.2307/204500.

Avery P, Henderson D (1999). "Fitting Markov Chain Models to Discrete State Series Such as DNA Sequences." *Journal of the Royal Statistical Society C*, **48**(1), 53–61. doi:10.1111/1467-9876.00139.

Bartholomew D (1973). *Stochastic Models for Social Processes*. Wiley Series in Probability and Mathematical Statistics, 2nd edition. John Wiley & Sons.

Begleiter R, El-Yaniv R, Yona G (2004). "On Prediction Using Variable Order Markov Models." *Journal of Artificial Intelligence Research*, **22**, 385–421.

Bejerano G, Yona G (2001). "Variations on Probabilistic Suffix Trees: Statistical Modeling and Prediction of Protein Families." *Bioinformatics*, **17**(1), 23–43. doi:10.1093/bioinformatics/17.1.23.

Berchtold A (1998). *Chaînes de Markov et Modèles de Transition*. Hermès, Paris.

Berchtold A (2010). "Sequence Analysis and Transition Models." In M Breed, J Moore (eds.), *Encyclopedia of Animal Behavior*, pp. 139–145. Academic Press, Oxford. doi:10.1016/b978-0-08-045337-8.00233-3.

Berchtold A, Raftery A (2002). "The Mixture Transition Distribution Model for High-Order Markov Chains and Non-Gaussian Time Series." *Statistical Science*, **17**(3), 328–356. doi:10.1214/ss/1042727943.

Berchtold A, Sackett G (2002). "Markovian Models for the Developmental Study of Social Behavior." *American Journal of Primatology*, **58**(3), 149–167. doi:10.1002/ajp.10056.

Bühlmann P, Wyner A (1999). "Variable Length Markov Chains." *The Annals of Statistics*, **27**(2), 480–513. doi:10.1214/aos/1018031204.

Burnham K, Anderson D (2004). "Multimodel Inference." *Sociological Methods and Research*, **33**(2), 261–304. doi:10.1177/0049124104268644.

Ching W, Fung E, Ng M (2004). "Higher-Order Markov Chain Models for Categorical Data Sequences." *Naval Research Logistics*, **51**(4), 557–574. doi:10.1002/nav.20017.

Falkhausen M, Reininger H, Wolf D (1995). "Calculation of Distance Measures Between Hidden Markov Models." In *EUROSPEECH'95 – Fourth European Conference on Speech Communication and Technology*, pp. 1487–1490. URL http://www.isca-speech.org/archive/eurospeech_1995/e95_1487.html.

Gabadinho A (2016). ***PST**: Probabilistic Suffix Trees and Variable Length Markov Chains*. R package version 0.90, URL https://CRAN.R-project.org/package=PST.

Gabadinho A, Ritschard G, Müller N, Studer M (2011a). "Analyzing and Visualizing State Sequences in R with **TraMineR**." *Journal of Statistical Software*, **40**(4), 1–37. doi:10.18637/jss.v040.i04.

Gabadinho A, Ritschard G, Studer M, Müller N (2011b). "Extracting and Rendering Representative Sequences." In A Fred, J Dietz, K Liu, J Filipe (eds.), *Knowledge Discovery, Knowlege Engineering and Knowledge Management*, volume 128 of *Communications in Computer and Information Science*, pp. 94–106. Springer-Verlag, Berlin. doi:10.1007/978-3-642-19032-2_7.

Galves A, Löcherbach E (2008). "Stochastic Chains with Memory of Variable Length." arXiv:0804.2050 [math.PR], URL http://arxiv.org/abs/0804.2050.

He Q, Kwong S, Man KF, Tang KS (2000). "An Improved Maximum Model Distance Approach for HMM-Based Speech Recognition Systems." *Pattern Recognition*, **33**(10), 1749–1758. doi:10.1016/s0031-3203(99)00144-2.

Idler E, Benyamini Y (1997). "Self-Rated Health and Mortality: A Review of Twenty-Seven Community Studies." *Journal of Health and Social Behavior*, **38**(1), 21–37. doi:10.2307/2955359.

Juang B, Rabiner L (1985). "A Probabilistic Distance Measure for Hidden Markov Models." *ATT Technical Journal*, **64**(2), 391–408. doi:10.1002/j.1538-7305.1985.tb00439.x.

Kaplan D (2008). "An Overview of Markov Chain Methods for the Study of Stage-Sequential Developmental Processes." *Developmental Psychology*, **44**(2), 457–467. doi:10.1037/0012-1649.44.2.457.

Katz R (1981). "On Some Criteria for Estimating the Order of a Markov Chain." *Technometrics*, **23**(3), 243–249. doi:10.2307/1267787.

Kermorvant C, Dupont P (2002). "Improved Smoothing for Probabilistic Suffix Trees Seen as Variable Order Markov Chains." In T Elomaa, H Mannila, H Toivonen (eds.), *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pp. 135–145. Springer-Verlag, Berlin. doi:10.1007/3-540-36755-1_16.

Langeheine R, Van de Pol F (1990). "A Unifying Framework for Markov Modeling in Discrete Space and Discrete Time." *Sociological Methods and Research*, **18**, 416–441. doi:10.1177/0049124190018004002.

Langeheine R, Van de Pol F (2000). "Fitting Higher Order Markov Chains." *Methods of Psychological Research Online*, **5**(1). URL http://www.dgps.de/fachgruppen/methoden/mpr-online/issue9/art2/article.html.

Largeron C (2003). "Prediction Suffix Trees for Supervised Classification of Sequences." *Pattern Recognition Letters*, **24**(16), 3153–3164. doi:10.1016/j.patrec.2003.08.002.

Levine J (2000). "But What Have You Done for Us Lately?" *Sociological Methods and Research*, **29**(1), 34–40. doi:10.1177/0049124100029001002.

Low-Kam C, Laurent A, Teisseire M (2009). "Détection de Séquences Atypiques Basée Sur un Modèle de Markov d'Ordre Variable." In *Extraction et Gestion des Connaissances (EGC'2009), Actes, Strasbourg, 27 au 30 Janvier 2009*, pp. 217–228. URL http://editions-rnti.fr/?inprocid=1000766.

Mächler M (2015). ***VLMC**: Variable Length Markov Chains Models*. R package version 1.4-1, URL https://CRAN.R-project.org/package=VLMC.

Mächler M, Bühlmann P (2004). "Variable Length Markov Chains: Methodology, Computing, and Software." *Journal of Computational and Graphical Statistics*, **13**(2), 435–455. doi:10.1198/1061860043524.

Mazeroff G, Gregor J, Thomason M, Ford R (2008). "Probabilistic Suffix Models for API Sequence Analysis of Windows XP Applications." *Pattern Recognition*, **41**(1), 90–101. doi:10.1016/j.patcog.2007.04.006.

Rabiner LR (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE*, **77**(2), 257–286. doi:10.1109/5.18626.

Raftery A (1985). "A Model for High-Order Markov Chains." *Journal of the Royal Statistical Society B*, **47**(3), 528–539.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rissanen J (1983). "A Universal Data Compression System." *IEEE Transactions on Information Theory*, **29**(5), 656–664. doi:10.1109/tit.1983.1056741.

Ron D, Singer Y, Tishby N (1996). "The Power of Amnesia: Learning Probabilistic Automata With Variable Memory Length." *Machine Learning*, **25**(2–3), 117–149. doi:10.1007/bf00114008.

Seldin Y, Bejerano G, Tishby N (2001). "Unsupervised Sequence Segmentation by a Mixture of Switching Variable Memory Markov Sources." In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pp. 513–520. Morgan Kaufmann.

Singer B, Spilerman S (1973). "Social Mobility Models for Heterogeneous Populations." *Sociological Methodology*, **5**, 356–401. doi:10.2307/270841.

Singer B, Spilerman S (1976). "The Representation of Social Processes by Markov Models." *American Journal of Sociology*, **82**(1), 1–54. doi:10.1086/226269.

Smith A, Shelley J, Dennerstein L (1994). "Self-Rated Health: Biological Continuum or Social Discontinuity?" *Social Science & Medicine*, **39**(1), 77–83. doi:10.1016/0277-9536(94)90167-8.

Spilerman S (1972). "The Analysis of Mobility Processes by the Introduction of Independent Variables into a Markov Chain." *American Sociological Review*, **37**(3), 277–294. doi:10.2307/2093468.

Sun P, Chawla S, Arunsalam B (2006). "Mining for Outliers in Sequential Databases." In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pp. 94–105. doi:10.1137/1.9781611972764.9.

Tong H (1975). "Determination of the Order of a Markov Chain by Akaike's Information Criterion." *Journal of Applied Probability*, **12**(3), 488–497. doi:10.1017/s0021900200048294.

Wiewiora E (2008). *Modeling Probability Distributions with Predictive State Representations*. Ph.D. thesis, University of California, San Diego.

Yang J, Wang W (2003). "CLUSEQ: Efficient and Effective Sequence Clustering." In *Proceedings of the 19th International Conference on Data Engineering (ICDE), March 5–8, 2003, Bangalore, India*, pp. 101–112.

Affiliation:

Alexis Gabadinho, Gilbert Ritschard
NCCR LIVES
Institute for Life Course and Demographic Studies
University of Geneva
CH-1211 Geneva 4, Switzerland
E-mail: alexis.gabadinho@unige.ch
URL: http://mephisto.unige.ch/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2016, Volume 72, Issue 3 *Submitted:* 2012-11-13
doi:10.18637/jss.v072.i03 *Accepted:* 2015-07-16