PROCDATA: AN R PACKAGE FOR PROCESS DATA ANALYSIS

Xueying Tang

university of arizona

Susu Zhang

university of illinois at urbana-champaign

Zhi Wang, Jingchen Liu and Zhiliang Ying

columbia university

June 29, 2021

Correspondence should be sent to

E-Mail: jcliu@stat.columbia.edu

PROCDATA: AN R PACKAGE FOR PROCESS DATA ANALYSIS

Abstract

Process data refer to data recorded in log files of computer-based items. These data, represented as timestamped action sequences, keep track of respondents' response problem-solving behaviors. Process data analysis aims at enhancing educational assessment accuracy and serving other assessment purposes by utilizing the rich information contained in response processes. The `R` package `ProcData` presented in this article is designed to provide tools for inspecting, processing, and analyzing process data. We define an `S3` class '`proc`' for organizing process data and extend generic methods `summary` and `print` for '`proc`'. Feature extraction methods for process data are implemented in the package for compressing information in the irregular response processes into regular numeric vectors. `ProcData` also provides functions for making predictions from neural-network-based sequence models. In addition, a real dataset of response processes from the climate control item in the 2012 Programme for International Student Assessment is included in the package.

Key words: process data analysis, multidimensional scaling, autoencoder, sequence model

1. Introduction

With the advancement of technology, computer-based assessments have become popular in measuring complex human skills such as problem solving skills. In these assessments, participants are often asked to accomplish real-life tasks in a simulated environment. As a participant interacts with a computer to complete the tasks, the entire interaction process is recorded in a log file. Each process includes actions such as mouse clicks and keystrokes and timestamps at which these actions took place. Such data describe the process of responding to an item and are thus called response process data, or in short, process data.

Naturally, process data contain substantially more information concerning each respondent than traditional dichotomous or polytomous item responses. A response process not only indicates whether the final answer to the item is correct or not but also details how the problem is solved. The detailed information enables researchers to compare behavior patterns in successful and unsuccessful responses, detect abnormal behaviors, and study other problems that are otherwise difficult or even impossible to study (Ren et al., 2019; Stadler, Fischer, & Greiff, 2019; C. Wang, Xu, Shang, & Kuncel, 2018). It has also been quantitatively verified that the variables constructed from process data have higher prediction power for a wide range of variables that are directly or indirectly related to problem-solving behaviors (X. Tang, Wang, He, Liu, & Ying, 2020; X. Tang, Wang, Liu, & Ying, 2020; Zhang, Tang, He, Liu, & Ying, 2021). Moreover, with an appropriate design of the scoring rule, process-data-based assessment substantially improves assessment reliability compared to the item-response-theory-based (IRT-based) scores (Zhang, Wang, Qi, Liu, & Ying, 2021). In particular, the process-data-based score of one item is as accurate as the IRT-based score of three items. In addition, differential item functioning may be reduced or even completely removed by looking into the response processes and making use of the information appropriately. There are many other ways that researchers may make use of process data and they are yet to be discovered.

Process data analysis has gained prominence in the past few years. Various methods have been developed. He and von Davier (2016) and Qiao and Jiao (2018) found that n-grams of action sequences are useful for predicting the item performance and for clustering respondents.

Chen, Li, Liu, and Ying (2019) analyzed process data through an event history analysis approach. Recently, two generic feature extraction methods have been developed to automatically construct informative features from process data (X. Tang, Wang, He, et al., 2020; X. Tang, Wang, Liu, & Ying, 2020). These methods transform response processes into real-valued vectors (features) without losing too much information. One of the two methods is based on multidimensional scaling and the other is based on a type of neural networks called autoencoder. Besides autoencoders, other neural networks such as recurrent neural networks (RNNs) have also been found potentially useful for analyzing process data (S. Tang, Peterson, & Pardos, 2016).

Although the methodology development has been prosperous, software for analyzing process data has not been developed in parallel. Due to the special structure of response processes, few existing packages for analyzing real-valued or categorical time series are applicable to process data analysis. Recently, a `Python` library `glassPy` (Hao, Smith, Mislevy, von Davier, & Bauer, 2016) is developed for analyzing game/simulation-based assessment log files. This library focuses on processing and summarizing the raw log files in extensive markup language (XML). It also includes tools for visualizing the summarized information. As far as the authors' knowledge, there is currently no software implementing advanced statistical and machine learning methods designed for process data. The lack of software implementation hinders the applications of the state-of-the-art methods and slows down the roll-out of scientific discovery. We try to fill in this gap by developing an `R` package `ProcData`. This package provides easy-to-use tools for basic exploration and operation of process data. It also includes functions implementing the state-of-the-art methods for process data analysis. `ProcData` is available at the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=ProcData` and under development on GitHub at `https://github.com/xytangtang/ProcData`.

The remaining sections of this article are organized as follows. In Section 2, we describe the state-of-the-art methods implemented in `ProcData`. We select a set of important elements of the `ProcData` package and present them in Section 3. A case study of the climate control item in PISA 2012 is presented in Section 4 to demonstrate the usage of the package[1]. A summary is

[1] `ProcData 0.2.5` is used to produce the results in the case study. This version is available on

included in Section 5.

2. Methods

In this article, a response process is represented by $\boldsymbol{o} = (\boldsymbol{s}, \boldsymbol{t})$. Each response process consists of two sequences, an action sequence $\boldsymbol{s} = (s_1, \ldots, s_L)$ and a timestamp sequence $\boldsymbol{t} = (t_1, \ldots, t_L)$ where L denotes the length of the response process, that is, the total number of actions taken during the process. The action sequence $\boldsymbol{s}$ records the actions taken by the respondent to solve the item in order. Each element in $\boldsymbol{s}$ is one of the M possible actions, $a_1, \ldots, a_M$, in the item. We call $\mathcal{A} = \{a_1, \ldots, a_M\}$ the action set of the item. The timestamp sequence $\boldsymbol{t}$ records the time of actions in $\boldsymbol{s}$ from inception and therefore $0 \leq t_1 \leq t_2 \leq \cdots \leq t_L$. For a set of response processes $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_N$ of N respondents, the length of a process is likely to vary across observations. We write $\boldsymbol{o}_i = (\boldsymbol{s}_i, \boldsymbol{t}_i)$, $\boldsymbol{s}_i = (s_{i1}, \ldots, s_{iL_i})$, and $\boldsymbol{t}_i = (t_{i1}, \ldots, t_{iL_i})$, where L_i denotes the length of the response process for $\boldsymbol{o}_i$, $i = 1, \ldots, N$.

A major technical difficulty for process data analysis is that response processes cannot be conveniently organized as a matrix. Thus standard statistical methods are not directly applicable to process data. Feature extraction methods have been shown useful for exploratory process data analysis. These methods compress response processes into vectors of a prespecified dimension. These vectors are deemed as features of the response processes and are readily incorporated in traditional statistical models such as regression models. Furthermore, the extracted features are less noisy than the original response processes whose high noise level constitutes another challenge of direct analysis of process data. In this section, we describe two unsupervised feature extraction methods (X. Tang, Wang, He, et al., 2020; X. Tang, Wang, Liu, & Ying, 2020) and an RNN-based supervised learning model for response processes. We begin the discussion by briefly introducing RNNs, a key building block in these methods. We refer readers to Goodfellow, Bengio, and Courville (2016) and Patterson and Gibson (2017) for a more thorough description of neural networks.

CRAN at `https://cran.r-project.org/src/contrib/Archive/ProcData/` or on Github at `https://github.com/xytangtang/ProcData/tree/95a3658?`.

2.1. Introduction to neural networks

Artificial neural networks or, in short, neural networks are nonlinear models describing relationships among variables. Each neural network is essentially a parameterized family of nonlinear functions. Its structure is specially designed to achieve a specific modeling aim while being flexible.

==========

Insert Figure 1 about here

==========

RNNs are a class of artificial neural networks often used for processing sequential information. Unlike the feedforward neural network (FNN; Goodfellow et al., 2016, Chapter 6) that treats an input as a simple vector, RNNs have a special structure to utilize the sequential information in the data. As depicted in Figure 1, an RNN has three components: inputs, hidden states, and outputs, each of which is a multivariate time series. The inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ are $K^{(\mathrm{I})}$-dimensional vectors. The hidden states $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_L$ are $K^{(\mathrm{H})}$-dimensional and can be viewed as the memory that helps process the input information sequentially. The hidden state evolves as the elements in the input are processed. Each $\boldsymbol{m}_l$ summarizes the information in the input sequence up to step l by integrating the current information $\boldsymbol{x}_l$ with the previous memory $\boldsymbol{m}_{l-1}$, that is, $\boldsymbol{m}_l$ is a function of $\boldsymbol{x}_l$ and $\boldsymbol{m}_{l-1}$

$$\boldsymbol{m}_l = f(\boldsymbol{x}_l, \boldsymbol{m}_{l-1}), \tag{1}$$

for $l = 1, \ldots, L$. The initial hidden state $\boldsymbol{m}_0$ is often set to be the zero vector. To extract the information useful for subsequent tasks from the memory, a $K^{(\mathrm{O})}$-dimensional output vector $\boldsymbol{y}_l$ is produced as a function of the hidden state $\boldsymbol{m}_l$ at each step l,

$$\boldsymbol{y}_l = g(\boldsymbol{m}_l). \tag{2}$$

Both f and g are specified as a parametric family of functions with parameters to be estimated based on data. Various choices of f and g have been proposed to compute the hidden states and

the outputs. Two most widely used ones are the long-short-term-memory (LSTM) unit and the gate recurrent unit (GRU). They are designed to mitigate the vanishing or exploding gradient problem of a basic RNN (Bengio, Simard, & Frasconi, 1994). We refer the readers to Hochreiter and Schmidhuber (1997) and Cho et al. (2014) for the expressions of LSTM and GRU.

RNNs take input process sequences of different lengths. Note that the functions f and g in (1) and (2) are the same across time steps. Therefore, the total number of parameters for an RNN does not depend on the sequence length.

2.2. Autoencoder Feature Extraction

Autoencoders (Goodfellow et al., 2016, Chapter 14) are a class of neural networks aiming at reconstructing the input in the output. It consists of two components (Figure 2), an encoder that maps the complex and/or high dimensional input to a low dimensional vector and a decoder that reconstructs the input from the low dimensional vector. The low-dimensional vector produced by the encoder contains condensed information to rebuild the input. Therefore, its elements can be treated as features of the input data.

==========

Insert Figure 2 about here

==========

X. Tang, Wang, Liu, and Ying (2020) proposed to extract features from the action sequences in process data by means of a sequence autoencoder (SeqAE) . The structure of the action SeqAE is depicted in Figure 3. Its encoder consists of three steps. In the first step, each unique action in $\mathcal{A}$ is associated with a numeric vector (embedding) in $\mathbb{R}^K$ so that an input action sequence is transformed into a sequence of K-dimensional embeddings. In the second step, an RNN is used to sequentially summarize the information in the embedding sequence up to a time step. In the last step, the last vector in the output of the encoder RNN is retained as the feature vector. The decoder of the action SeqAE also consists of three steps. In the first step, the feature vector is repeated to form a sequence of vectors which are then passed into another RNN in the second step to obtain another sequence of vectors. Each vector in this latter sequence contains the

information of the action at the corresponding time step. In the last step, for $l = 1, \ldots, L$, a probability distribution $\hat{s}_l = (\hat{s}_l^{(1)}, \hat{s}_l^{(2)}, \ldots, \hat{s}_l^{(M)})$ on $\mathcal{A}$ is constructed at time step l using a multinomial logit model (MLM). The output vector $\boldsymbol{y}_l$ of the decoder RNN is used as the covariates in the MLM. The MLMs at different time steps share the parameters. In both the encoder RNN and the decoder RNN, $K^{(\mathrm{I})} = K^{(\mathrm{H})} = K^{(\mathrm{O})} = K$.

==========

Insert Figure 3 about here

==========

X. Tang, Wang, Liu, and Ying (2020) focused on the action sequences in process data. The idea of action SeqAE can be easily extended to the time sequences or both action and time sequences to construct a time SeqAE or an action-time SeqAE. As shown in Figure 4, the structure of a time SeqAE resembles that of an action SeqAE. RNNs are used in both encoder and decoder to summarize the sequential information. However, because timestamps are numeric, in a time SeqAE, an embedding step is not needed in the encoder and the timestamps are reconstructed through a linear model instead of a multinomial logit model. An action-time SeqAE can be constructed by combining an action SeqAE and a time SeqAE as shown in Figure 5.

==========

Insert Figure 4 about here

==========

==========

Insert Figure 5 about here

==========

Given the structure of a SeqAE, the parameters of the autoencoder, including the embeddings, the parameters in the encoder and decoder RNNs, and the parameters in the multinomial logit model and the linear model, are estimated by minimizing the discrepancy between the input sequences and the output sequences. To be more specific, the estimated

parameter vector $\hat{\boldsymbol{\eta}}$ is obtained by minimizing

$$F(\boldsymbol{\eta}) = \sum_{i \in \Omega} \delta(\boldsymbol{o}_i, \hat{\boldsymbol{o}}_i; \boldsymbol{\eta}), \tag{3}$$

where $\hat{\boldsymbol{o}} = (\hat{\boldsymbol{s}}, \hat{\boldsymbol{t}})$ denotes the output of a SeqAE for input $\boldsymbol{o}$, $\delta(\boldsymbol{o}, \hat{\boldsymbol{o}})$ is a function measuring the discrepancy between $\boldsymbol{o}$ and $\hat{\boldsymbol{o}}$, and $\boldsymbol{\eta}$ is a vector containing all parameters to be estimated in the SeqAE. For an action SeqAE,

$$\delta(\boldsymbol{o}, \hat{\boldsymbol{o}}) = \delta_{\mathrm{a}}(\boldsymbol{s}, \hat{\boldsymbol{s}}) = -\frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{M} \mathbf{1}\{s_l = a_j\} \log(\hat{s}_l^{(j)}).$$

For a time SeqAE,

$$\delta(\boldsymbol{o}, \hat{\boldsymbol{o}}) = \delta_{\mathrm{t}}(\boldsymbol{t}, \hat{\boldsymbol{t}}) = \sum_{l=1}^{L} (t_l - \hat{t}_l)^2.$$

For an action-time SeqAE,

$$\delta(\boldsymbol{o}, \hat{\boldsymbol{o}}) = w_a \delta_{\mathrm{a}}(\boldsymbol{s}, \hat{\boldsymbol{s}}) + w_t \delta_{\mathrm{t}}(\boldsymbol{t}, \hat{\boldsymbol{t}}),$$

where $\boldsymbol{w} = (w_a, w_t)$ assigns different weights on the discrepancy between action sequences and the discrepancy between time sequences. The objective function (3) is usually minimized via gradient-based stochastic approximation (Hinton, Srivastava, & Swersky, 2014; Kingma & Ba, 2015; Robbins & Monro, 1951; Zeiler, 2012). Since neural networks are often over-parametrized, the algorithm is usually stopped before convergence to avoid overfitting. A widely used early stopping method monitors the value of the objection function on a validation set. After running the algorithm long enough, one uses the parameter value producing the lowest monitored value as the estimate.

2.3. Multidimensional Scaling Feature Extraction

Multidimensional scaling (MDS) (Borg & Groenen, 2005) is a technique often used for dimension reduction and data visualization. Its goal is to embed objects in a space in such a way that the dissimilarity between objects is approximated by the distance between their embedded features. Similar objects are located close to each other while dissimilar objects are far apart. Given a dissimilarity measure that comprehensively characterizes the difference between objects, the object coordinates obtained from MDS can be viewed as features describing the latent

attributes of the objects. Given a dissimilarity measure d and the latent feature dimension K, features of a set of response processes $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n$ are a solution to the optimization problem

$$\min_{\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_N \in \mathbb{R}^K} \sum_{1 \le i < j \le n} (d_{ij} - \|\boldsymbol{\theta}_i - \boldsymbol{\theta}_j\|)^2, \tag{4}$$

where $d_{ij} = d(\boldsymbol{o}_i, \boldsymbol{o}_j)$ is the dissimilarity between response processes $\boldsymbol{o}_i$ and $\boldsymbol{o}_j$, $\boldsymbol{\theta}_i$ is the feature vector of response process $\boldsymbol{o}_i$, and $\|\mathbf{x}\| = \sqrt{\mathbf{x}^\top \mathbf{x}}$.

X. Tang, Wang, He, et al. (2020) consider the optimal symbol similarity (OSS) measure proposed in Gómez-Alonso and Valls (2008). For two response processes $\boldsymbol{o}_i$ and $\boldsymbol{o}_j$, the OSS measure is

$$d_{\text{OSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \frac{f_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j) + g_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j)}{L_i + L_j}, \tag{5}$$

where $f_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j)$ quantifies the dissimilarity among the actions that appear in both $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ and $g_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j)$ is the count of actions appearing in only one of $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. More precisely,

$$f_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \frac{\sum_{a \in C_{ij}} \sum_{k=1}^{K_{ij}^a} |\boldsymbol{s}_i^a(k) - \boldsymbol{s}_j^a(k)|}{\max\{L_i, L_j\}}, \tag{6}$$

and

$$g_{\text{OSS}}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \sum_{a \in U_{ij}} L_i^a + \sum_{a \in U_{ji}} L_j^a, \tag{7}$$

where $\boldsymbol{s}^a$ denotes the sequence consisting of chronologically ordered positions of action a in action sequence $\boldsymbol{s}$, L^a is the length of $\boldsymbol{s}^a$, $\boldsymbol{s}^a(k)$ represents the kth element of $\boldsymbol{s}^a$, and $K_{ij}^a = \min\{L_i^a, L_j^a\}$.

The OSS measure only takes the action sequences into account when characterizing the differences between two response processes. To account for the difference in both action sequences and time sequences, a time-weighted version of OSS (TOSS) can be considered. The length of a response process in OSS is replaced by the total response time of the response process in TOSS and L^a is replaced by the total time spent on a. In (6), $|\boldsymbol{s}_i^a(k) - \boldsymbol{s}_j^a(k)|$ is the difference between the positions of the kth appearance of a in $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. In TOSS, this quantity is replaced by the difference between the two corresponding timestamps. Namely,

$$d_{\text{TOSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \frac{f_{\text{TOSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j) + g_{\text{TOSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j)}{t_{iL_i} + t_{jL_j}}, \tag{8}$$

where

$$f_{\text{TOSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \frac{\sum_{a \in C_{ij}} \sum_{k=1}^{K_{ij}^a} |t_{i\boldsymbol{s}_i^a(k)} - t_{j\boldsymbol{s}_j^a(k)}|}{\max\{t_{iL_i}, t_{jL_j}\}}, \tag{9}$$

and

$$g_{\text{TOSS}}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \sum_{a \in U_{ij}} \sum_{l: s_{il} = a} (t_{il} - t_{i,l-1}) + \sum_{a \in U_{ji}} \sum_{l: s_{jl} = a} (t_{jl} - t_{j,l-1}). \tag{10}$$

Given the dissimilarity function d, the solution to (4) is approximated by performing an eigenvalue decomposition of the matrix $-\frac{1}{2}\mathbf{J}\mathbf{D}^{(2)}\mathbf{J}$ where $\mathbf{J} = \mathbf{I} - \frac{1}{N}\mathbf{1}^\top\mathbf{1}$ and $\mathbf{D}^{(2)} = (d_{ij}^2)$. This is called classical MDS. The computational complexity of the algorithm is $O(N^3)$, which is very expensive if the number of processes N is large. Moreover, the $N \times N$ dissimilarity matrix D consumes a large amount of memory. Paradis (2018) proposed an algorithm to perform MDS for a large N. This algorithm first chooses a small subset Ω of the objects and obtains $\hat{\boldsymbol{\theta}}_i, i \in \Omega$ by performing classical MDS on this subset. Then it minimizes

$$F(\boldsymbol{\theta}_i) = \sum_{j \in \Omega} (d_{ij} - \|\boldsymbol{\theta}_i - \hat{\boldsymbol{\theta}}_j\|)^2 \tag{11}$$

by the BFGS method (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) for each $i \notin \Omega$. In this way, only the dissimilarities for $O(\tilde{N}N)$ pairs of objects are calculated where $\tilde{N}$ is the subset size and the eigenvalue decomposition for a large matrix is avoided.

The MDS feature extraction method requires specifying K, the number of features to be extracted. In X. Tang, Wang, He, et al. (2020), K is selected by k-fold cross-validation. The authors also recommended performing principal component analysis on the extracted feature to obtain more interpretable features.

2.4. Sequence Models

The feature extraction methods described above are designed to capture the variations among response processes. We further present a sequence model that extracts features for some specific variable of interest. As depicted in Figure 6, a sequence model has two parts. The first part is identical to the encoder. It embeds a response process to a fixed dimensional vector $\boldsymbol{\theta}$ via an RNN. In the second part, instead of reconstructing the response process, the sequence model utilizes a generalized linear model (GLM) to predict the target variable y given $\boldsymbol{\theta}$. Thus, the

feature vector is trained to explain the target variable instead of the variations in response processes. Besides GLMs, other supervised learning models such as FNNs can also be used in the second part to describe a more complex relationship between $\boldsymbol{\theta}$ and y.

==========

Insert Figure 6 about here

==========

Figure 6 illustrates the structure of the sequence model when the information in both the action sequence and the timestamp sequence is considered. Similar structures have been used for sentiment analysis in natural language processing (X. Wang, Liu, Sun, Wang, & Wang, 2015).

The parameters in the sequence model includes the embedding, the parameters in the RNN, and the parameters in the GLM or FNN. Given a set of response processes and the corresponding values of the target variable, the parameters can be estimated by the maximum likelihood approach or, equivalently, minimizing the discrepancy between the observed y and the estimated mean value $\hat{y}$ based on the model. More specifically, the estimated value of the parameters minimizes

$$F(\boldsymbol{\eta}) = \sum_{i \in \Omega} \delta(y_i, \hat{y}_i), \tag{12}$$

where $\delta(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$ if y is binary and $\delta(y, \hat{y}) = (y - \hat{y})^2$ if y is continuous. Similar to SeqAE, the objective function (12) can be optimized by stochastic approximation with early stopping. Once the parameters in the sequence model is estimated, the model output $\hat{y}$ is a prediction of the response variable. The output of the first part of the model are the features most relevant to the target response variable.

3. The ProcData Package

We develop the `ProcData` package to provide tools for process data analysis. `ProcData` defines an `S3` object for process data and includes functions for data processing, inspection, and modeling. The main features of `ProcData` is summarized in Table 1. The details of these features are described in Sections 3.1–3.5.

==========

Insert Table 1 about here

==========

As described in Section 2, some methods for process data require construction and training of neural networks. `ProcData` relies on `R` package `keras` for achieving this functionality. The `keras` package is an `R` interface to a high level neural network API developed for fast experimentation of neural networks in `Python`. The functions in `ProcData` that are built on functions in `keras` are marked by † in Table 1. If `keras` is not installed properly, calling these functions will lead to an error while other functions in `ProcData` can still be used normally. The installation guide of the `keras` package can be found at `https://keras.rstudio.com/`. Note that a successful installation of `keras` requires an installation of `Python`.

3.1. Data Input and Output

`ProcData` includes a dataset from the climate control item in PISA 2012. The dataset contains the response processes and the dichotomous response outcomes of 16,763 students. The item interface is described later in Section 4 and is available online [2]. The data is loaded as follows.

```
library("ProcData")
data("cc_data")
```

The data object `cc_data` is a list of two elements, `seqs` and `responses`. The response outcomes are contained in `responses` as a numeric vector. The response processes are stored in `seqs` as an object of class '`proc`', which will be specified in Section 3.2.

==========

Insert Figure 7 about here

==========

[2] http://www.oecd.org/pisa/test-2012/testquestions/question3/

==========

Insert Figure 8 about here

==========

To read and write datasets on hard drive, `ProcData` provides functions for data input and output. Process data are usually stored as comma separated values (CSV) files in either "single" or "multiple" style. In both styles, all the action sequences form a column (action column) and all the timestamp sequences form another column (time column). In the "single" style, the process for one respondent takes up a single row in the CSV file. The entire action sequence of the respondent is stored as one entry of the action column. The timestamp sequence of the respondent is stored in the corresponding entry of the time column. Figure 7 presents a CSV file storing five response processes in the "single" style. In the "multiple" style, each action in the process and its timestamp occupy one row in the CSV file. A process of length L takes up L consecutive rows. A CSV file that stores two response processes in the "multiple" style is displayed in Figure 8. The two response processes are the same as the first two shown in Figure 7.

In `ProcData`, functions `read.seqs()` and `write.seqs()` read and write process data from and to a CSV file. The file style is specified via argument `style` as shown in the sample code below. Reading and writing files require specification of `id_var`, `action_var`, and `time_var`, which are the column names for respondent identity number, action sequence, and timestamp sequence, respectively. The separator of the elements in the action sequence and the timestamp sequence is provided via `step_sep` if the file to read or write is in the single style.

```
write.seqs(cc_data$seqs, file="seqs_format_multiple.csv", style = "multiple",
          id_var="ID", action_var="Action", time_var="Time")
write.seqs(cc_data$seqs, file="seqs_format_single.csv", style = "single",
           id_var="ID", action_var="Action", time_var="Time", step_sep=",")
seqs_multiple <- read.seqs("seqs_format_multiple.csv", style = "multiple",
                           id_var = "ID", action_var = "Action", time_var = "Time")
seqs_single <- read.seqs("seqs_format_single.csv", style = "single", id_var = "ID",
                         action_var = "Action", time_var = "Time", step_sep = ",")
```

3.2. S3 class 'proc'

`ProcData` defines an `S3` class '`proc`' for organizing process data. An object of class '`proc`' is a list of two elements, `action_seqs` and `time_seqs`. In a '`proc`' object storing response processes $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n$, `action_seqs` is a list with elements $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ and `time_seqs` is a list with elements $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$. The names of the elements in `action_seqs` and `time_seqs` are the identity of the respondents. If the timestamp sequences are not available, `time_seqs` is set to `NULL`. The `seqs` element in `cc_data` is a '`proc`' object.

An object of class '`proc`' is printed by the `print` method for class '`proc`', `print.proc()`. The function prints response processes one by one in an easy-to-read format. For each process, the action sequence and the timestamp sequence are aligned and are printed horizontally. One can print either the first several processes or a selected subset of the processes in the '`proc`' object. The format of the printed processes is demonstrated below.

```
ARE000000300079
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9
Event start  1_1_1  reset  0_0_1  reset  0_1_0  reset  1_0_0  end
Time    0.0  113.2  119.1  122.0  135.4  138.5  147.8  149.8  157.0
```

We also extend the `summary` method for class '`proc`'. Function `summary.proc()` returns a list containing various summary statistics of the action sequences and the timestamp sequences in the '`proc`' object.

3.3. Sequence Manipulation

==========

Insert Table 2 about here

==========

Process data often contain a large amount of noise and data cleaning is often necessary before further analysis. Table 2 lists the functions provided in `ProcData` for cleaning data in '`proc`' objects. Their basic usage is demonstrated below.

```
# obtain processes 4, 8, 12, ..., 40 from cc_data
seqs <- sub_seqs(cc_data$seqs, ids = 4*1:10)
# remove repetitive actions in the response processes
seqs1 <- remove_repeat(seqs)
# remove action "0_0_0"
seqs2 <- remove_action(seqs1, actions = "0_0_0")
# rename "reset" with "RESET"
seqs3 <- replace_action(seqs2, old_action = "reset", new_action = "RESET")
# combine actions "0_0_1" and "0_0_2" into a new action "BOTTOM_MOVE_ONE"
seqs4 <- combine_actions(seqs2, old_actions = c("0_0_1", "0_0_2"),
                         new_action = "BOTTOM_MOVE_ONE")
```

3.4. Feature Extraction

3.4.1. Feature Extraction via Multidimensional Scaling

In `ProcData`, `seq2feature_mds()` extracts feature from process data via MDS. It takes a '`proc`' object (`seqs`), the number of features to be extracted (`K`), and some other control arguments to calculate the dissimilarity matrix for the input response processes and then performs MDS.

The dissimilarity measure is specified via `dist_type`. In particular, two dissimilarity measures in Section 2.3 are implemented with `dist_type = "oss_action"` and `dist_type = "oss_both"` standing for OSS and TOSS, respectively. Users may also use their own dissimilarity measure by providing the dissimilarity matrix through `seqs`.

In `seq2feature_mds()`, the objective function (4) is optimized by either calling function `cmdscale()` in `R` to perform the classical MDS or applying the large sample size algorithm (Paradis, 2018). By default, `seq2feature_mds()` uses the classical MDS if there are fewer than 5000 response processes and the large sample size algorithm otherwise. One may also choose the algoritm by setting `method = "small"` for the classical MDS or `method = "large"` for the other algorithm.

Following the recommendation in X. Tang, Wang, He, et al. (2020), `seq2feature_mds()` performs principal component analysis on the extracted features to enhance the interpretability of the features. This can be turned off by setting `pca = FALSE`. By default, `seq2feature_mds()` returns a list containing the latent features (`theta`) and the value of the optimized objective function (`loss`). The dissimilarity matrix (`dist_mat`) is returned if `return_dist = TRUE`. `ProcData` also provides a function `chooseK_mds()` to select the feature dimension by k-fold cross-validation.

3.4.2. Sequence Autoencoder Feature Extraction

In `ProcData`, function `seq2feature_seq2seq()` is provided to extract K features from a given set of response processes (`seqs`) via SeqAEs. The SeqAE type is specified through argument `ae_type` as `"action"` (default), `"time"`, or `"action_time"`. As described in Section 2.2, RNNs are a critical component in SeqAEs. In `seq2feature_seq2seq()`, the choice of the recurrent units (`"lstm"` or `"gru"`) in the encoder and decoder RNNs is set by argument `rnn_type`.

By default, the original timestamp sequences are used for constructing time SeqAEs and action-time SeqAEs in `seq2feature_seq2seq()`. The inter-arrival time sequences are used by setting `cumulative = FALSE`. The natural logorithm of the timestamps or the inter-arrival time is used if `log = TRUE`.

The objective function (3) is minimized by stochastic approximation with early stopping as described in Section 2.2. The algorithms available are stochastic gradient descent (`optimizer_name = "sgd"`; Robbins & Monro, 1951), Adam (`optimizer_name = "adam"`; Kingma & Ba, 2015), AdaDelta (`optimizer_name = "adadelta"`; Zeiler, 2012), and RMSprop (`optimizer_name = "rmsprop"` Hinton et al., 2014). The (baseline) step size is set by argument `step_size`. The training and validation sets are specified in `samples_train` and `samples_valid`, respectively.

The training time of SeqAEs varies from a few minutes to a few hours depending on the number of possible actions in the item and the number of features to be extracted. In `seq2feature_seq2seq()`, one sets `verbose = TRUE` to print out and monitor the training progress.

3.5. Sequence Models

In `ProcData`, function `seqm()` constructs and fits a sequence model described in Section 2.4 with a target `response` variable y whose type is set by `response_type`. Currently, y can be either a binary variable (`"binary"`) or a continuous variable (`"scale"`). The dimensions of the action embeddings and the output of the RNN are specified through `K_emb` and `K_rnn`, respectively. By default, `seqm()` uses a generalized linear model to predict y based on the RNN output. Alternatively, one may use an FNN for this predictive model.

As a default choice, `seqm()` extracts features only from the action sequences. Timestamp sequences, if available, can be incorporated by setting `include_time = TRUE`. In this case, the logarithms of the inter-arrival time sequences are used by default. To use the original timestamp sequences, set `time_interval = FALSE` and `log_time = FALSE`.

Function `seqm()` returns an object of class '`seqm`', which is a list containing the neural network architecture, the estimated parameters, and other information about modeling fitting. The key elements of a '`seqm`' object are a character string describing the neural network structure (`structure`) and a matrix giving the training and validation loss at the end of each epoch for convergence check (`history`). Once a sequence model is fit, prediction and feature extraction are performed by `predict.seqm()`, the `predict` method of '`seqm`' objects.

4. Examples

In this section, we demonstrate the `ProcData` package through a case study of the climate control item in PISA 2012. The dataset `cc_data` is included in the package. In the demonstration, we extract features from the response processes and use them to predict respondents' task accomplishment. Furthermore, we investigate the behavior patterns in response processes related to task accomplishment.

4.1. PISA Climate Control Item

The climate control item is part of the Programme for International Student Assessment (PISA) 2012 for assessing students' problem solving skills. Figure 9 is a screenshot of the item

interface. In the simulated environment, there is a new air conditioner with no instructions. Students are asked to figure out which climate variable, temperature or humidity, that each of the three controls on the air conditioner influences. They can slide the control bars through the simulation interface and read how the temperature and humidity change. In the process, how the controls are moved and what buttons are clicked are recorded in the log files. For example, if a student clicked the "APPLY" button after moving the top control to "+" and the middle control to "−−", then action "1_-2_0" along with the time elapsed since the start of the item, say 5.4 seconds, are recorded. A recorded process with action sequence ("start", '1_0_0", "RESET", "0_0_-2", "end") and timestamp sequence $(0.0, 4.9, 6.3, 10.6, 12.5)$ indicates that the student moved the top control to "+" and clicked "APPLY" 4.9 seconds after the item started. The positions of the three controls were reset by a click of the "RESET" button 1.4 seconds later. Then the bottom control was moved to "−−" and "APPLY" was clicked again 10.6 seconds after the item started. The total response time is 12.5 seconds.

==========

Insert Figure 9 about here

==========

4.2. Data preprocessing

We randomly sample 2000 response processes from dataset `cc_data` for our analysis. Given a vector of randomly sampled indices, subsetting a '`proc`' object can be easily done using `sub_seqs()` provided in `ProcData`. In the subset, 53.7% of the respondents answered the item correctly. The sampled processes are then split into training, validation, and test sets.

```
# randomly sample a subset of size 2000 from cc_data
set.seed(12345)
n <- 2000
idx <- sample(1:length(cc_data$responses), n)
seqs <- sub_seqs(cc_data$seqs, idx)
y <- cc_data$responses[idx]
```

```
# proportion of incorrect (0) and correct (1) answers
table(y) / n
# split sampled processes into training, validation, and test set
n_train <- 1000; n_valid <- 500; n_test <- 500
index_train <- sample(1:n, n_train)
index_valid <- sample(setdiff(1:n, index_train), n_valid)
index_test <- setdiff(1:n, c(index_train, index_valid))
```

The following code demonstrates the `summary` method for '`proc`' objects.

```
seqs_summary <- summary(seqs)
# number of unique actions
seqs_summary$n_action
# action set
seqs_summary$actions
# range of sequence lengths
range(seqs_summary$seq_length)
# action transition probability matrix
trans_mat <- seqs_summary$trans_count
trans_mat <- trans_mat / rowSums(trans_mat)
```

The output shows that there are 128 unique actions in the 2000 response processes. The process length ranges from 3 to 183. Figure 10 presents a 25×25 submatrix of the 128×128 full action transition probability matrix `trans_mat`. The 25 actions corresponding to the submatrix are randomly selected from the action set. The dark diagonal of the submatrix indicates that respondents tend to repeat an action several times before taking a different action.

==========

Insert Figure 10 about here

==========

4.3. Unsupervised feature extraction

We begin the illustration of feature extraction with the intuitive action count features obtained as follows.

```
get_action_counts <- function(x, actions) sapply(actions, function(a) sum(x == a))
theta_action <- t(sapply(seqs$action_seqs, get_action_counts,
                         actions = seqs_summary$actions))
```

The output object `theta_action` is a 2000×128 matrix. The rows are feature vectors for different respondents. The columns correspond to the actions in the action set. The elements in a feature vector are the counts of the corresponding actions in the response process. Using these action features as covariates, we fit a logistic regression for the response outcome via `glm()` on the training and validation set combined. The prediction accuracy of the fitted model on the test set is 0.766.

Next, we extract features from response processes through multidimensional scaling. The number of features are chosen by five-fold cross-validation. We only consider the action sequences here and set `dist_type = "oss_action"`. We set `return_dist = TRUE` and `chooseK_mds()` returns the dissimilarity matrix for later use to avoid repeated calculations. The returned dissimilarity matrix and the selected number of features are then passed to `seq2feature_mds()` for feature extraction.

```
mds_K_res <- chooseK_mds(seqs, 1:10*10, dist_type = "oss_action",
                         return_dist = TRUE)
K <- mds_K_res$K; K
mds_res <- seq2feature_mds(mds_K_res$dist_mat, K = K)
```

As a result, 40 MDS features are extracted. These features are used similarly as the action count features to fit a logistic regression for the response outcome. The prediction accuracy based on MDS features is 0.81, which is slightly higher than that obtained from the model based on action count features. We further investigate the `glm` output and refit a logistic regression with the first MDS only. We find that including this single feature yields a prediction accuracy of 0.8.

Now we examine the behavior patterns associated with the first MDS feature. We order the response processes according to the value of their first feature and use the `print` method for class '`proc`' to display the response processes with the highest and lowest values.

```
o_mds1 <- order(mds_res$theta[,1])
# response processes corresponding to the lowest feature values
print(seqs, index = head(o_mds1))
# response processes corresponding to the highest feature values
print(seqs, index = tail(o_mds1))
```

The output of the above code is included in Appendix. The response processes corresponding to the highest feature values are often very short, meaning that little meaningful interaction with computer interface is made. Respondents with this type of behaviors are unlikely to answer the question correctly. The response processes corresponding to the lowest feature values are often longer. However, their lengths are not the longest according to Figure 11 where the value of the first feature is plotted against the logarithm of the length of the corresponding process. A closer look at the response processes with the lowest values of the feature reveals that these respondents often move one control bar at a time. This "varying-one-thing-at-a-time" strategy is efficient to get the correct answer.

==========

Insert Figure 11 about here

==========

We further extract features by constructing an action SeqAE. Below is an example of extracting action SeqAE features by `seq2feature_seq2seq()` in `ProcData`.

```
ae_res <- seq2feature_seq2seq(seqs, ae_type = "action", K = K,
                              rnn_type = "gru", samples_train = index_train,
                              samples_valid = index_valid, verbose = TRUE)
```

We set `verbose = TRUE` to print out the epoch number during the training of the autoencoder so that we can monitor the training progress. The extracted SeqAE features stored in `ae_res$theta`

are used to predict the task accomplishment. The prediction accuracy is 0.836, slightly higher than that of the model with MDS features.

4.4. Feature extraction with targeted variables via sequence model

We demonstrate the usage of `seqm()` and other relevant functions in this section. We still use the task accomplishment as the target variable. The training and validation sets are used for fitting the sequence model while the testing set is used for evaluating the prediction performance of the fitted model. The response processes used for model fitting and testing are obtained by subsetting the 'proc' object `seqs` with function `sub_seqs()` in `ProcData`.

```
seqs_train <- sub_seqs(seqs, c(index_train, index_valid))
seqs_test <- sub_seqs(seqs, index_test)
y_train <- y[c(index_train, index_valid)]
y_test <- y[index_test]
```

We first consider predicting y from the action sequence in a response process. The sequence model that fulfills this task can be fitted by calling function `seqm()`.

```
seqm_res <- seqm(seqs_train, y_train, response_type = "binary",
                 K_emb = 5, K_rnn = 5, n_epoch = 20,
                 max_len = max(seqs_summary$seq_length),
                 index_valid = n_train + 1:n_valid)
```

Since the response outcome is a binary variable, we specify `response_type = "binary"`. `n_epoch` is the total number of epochs to be run for estimating the parameters. In each epoch, the chosen stochastic approximation algorithm goes through the entire training set once.

The convergence of the training process is examined by a plot of the loss function values at the end of each epoch stored in `seqm_res$history` (Figure 12). We then make predictions based on the fitted model by calling the `predict` method for 'seqm' objects.

```
seq_pred_res <- predict(seqm_res, new_seqs = seqs_test)
mean(as.numeric(seq_pred_res > 0.5) == y_test)
```

The resulting prediction accuracy is 0.836.

==========

Insert Figure 12 about here

==========

To conclude the section, we present an example for both action sequences and time sequences. Time sequences are incorporated in addition to the action sequences by setting `include_time = TRUE` in `seqm()`. Other settings remain.

```
seqm_res2 <- seqm(seqs_train, y_train, response_type = "binary",
                  include_time = TRUE, K_emb = 5, K_rnn = 5, n_epoch = 20,
                  max_len = max(seqs_summary$seq_length),
                  index_valid = n_train + 1:n_valid)
seq_pred_res2 <- predict(seqm_res2, new_seqs = seqs_test)
mean(as.numeric(seq_pred_res2 > 0.5) == y_test)
```

The resulting prediction accuracy is 0.734, much lower than what we obtained previously. Although time sequences can provide more information, they at the same time introduce an excessive amount of noise, which can reduce the prediction accuracy. It is also possible that the current sequence model is not able to efficiently incorporating the information in timestamp sequences, causing a decreased prediction performance.

5. Summary

`ProcData` is an `R` package designed for process data analysis. It is applicable to process data generated by a wide range of human-computer interfaces. `ProcData` includes an `S3` class `"proc"` for organizing response processes. Functions for inspecting and processing `"proc"` objects are also included in the package. More importantly, two unsupervised learning feature extraction methods and the neural-network-based sequence models are implemented in `ProcData`. These tools are easy to use. The functions for each method integrate the major steps of the method while allowing the flexibility to change the settings to some degree. In particular, the functions for

neural-network-based methods wrap several function calls for constructing and training of neural network and thus simplify the application of these methods.

The feature extraction methods implemented in `ProcData` require little knowledge of the item designs. These methods compress the information in response processes into fixed-dimension real-valued vectors that can be analyzed using conventional statistical tools. The sequence model describes the relationship between response processes and a target variable. It can be used to make predictions of the target variable from response processes. This model can also be used to extract from response processes the features that are most relevant to the target variable.

The extracted features can be used in various psychometric applications. For instance, process-data-based scoring rules can be constructed based on the extracted features. The specific functional form is obtained by, for example, regressing a reliable estimate of the target scale on the features (Zhang, Wang, et al., 2021). The extracted features are also useful for exploratory analysis of response processes (Zhang, Tang, et al., 2021). The prediction based framework used in Section 4 can be adapted to exploring whether the response processes provide information on the variable of interest. If positive results are obtained from the preliminary analysis, more efforts can be devoted to examining the exact characteristics in the response processes that contribute to the effect.

Most of the existing methodology development for process data focuses on extracting information from the response processes produced by a single item. `ProcData` follows suit. Although features extracted from multiple items can be aggregated together in the subsequent analysis, methods that directly synthesize information in multiple items are much more preferable and could be helpful for understanding the relationship among items. Process data analysis is an active and quickly rising field. We envision to include more state-of-the-art methods in future versions of `ProcData`.

Appendix

This appendix contains the output of `print()` in Section 4.3.

```
> # response processes corresponding to the lowest feature values
> print(seqs, index = head(o_mds1))
'proc' object of  2000  processes

AUT000007001735
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9 Step 10
Event start  1_0_0  2_0_0  -2_0_0 -1_0_0 0_1_0  0_2_0  0_-1_0 0_-2_0 0_0_1
Time    0.0   57.4   60.6   62.8   67.1   79.6   81.1   83.3   85.3   97.2
      Step 11 Step 12 Step 13 Step 14
Event 0_0_2   0_0_-1  0_0_-2  end
Time   98.6   100.7   102.5   119.5

BRA000083118946
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9 Step 10
Event start  1_0_0  2_0_0  -1_0_0 -2_0_0 reset  0_1_0  0_2_0  0_-1_0 0_-2_0
Time    0.0  141.7  151.2  157.7  164.6  182.1  185.1  188.8  194.2  196.3
      Step 11 Step 12 Step 13 Step 14 Step 15 Step 16
Event reset   0_0_1   0_0_2   0_0_-1  0_0_-2  end
Time  204.0   206.6   209.4   215.9   218.6   240.4

KOR000006202016
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9 Step 10
Event start  2_0_0  1_0_0  -1_0_0 -2_0_0 reset  0_1_0  0_2_0  0_-1_0 0_-2_0
Time   0.0   46.1   52.4   56.4   59.2   62.5   67.3   69.3   70.3   72.4
      Step 11 Step 12 Step 13 Step 14 Step 15 Step 16
Event reset   0_0_1   0_0_2   0_0_-1  0_0_-2  end
```

```
Time  78.9    81.1    84.1    86.6    89.0    94.6

PRT000009602769
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9 Step 10
Event start  1_0_0  2_0_0  -1_0_0 -2_0_0 0_1_0  0_2_0  0_-1_0 0_0_1  0_0_2
Time    0.0   73.1   82.5   89.2   94.3  109.5  114.8  119.8  128.8  131.4
      Step 11
Event end
Time  139.7

ESP000073120436
      Step 1 Step 2 Step 3 Step 4 Step 5 Step 6 Step 7 Step 8 Step 9 Step 10
Event start  1_0_0  2_0_0  -1_0_0 -2_0_0 reset  0_1_0  0_2_0  0_-1_0 0_-2_0
Time    0.0   69.0   73.4   77.0   80.8   94.8  100.3  102.6  105.1  107.1
      Step 11 Step 12 Step 13 Step 14 Step 15 Step 16 Step 17
Event reset   0_0_1   0_0_2   0_0_-1  0_0_-2  reset   end
Time  116.9   118.9   121.9   125.1   127.2   136.0   140.6

> # response processes corresponding to the highest feature values
> print(seqs, index = tail(o_mds1))
'proc' object of  2000  processes

BGR000010603028
      Step 1 Step 2 Step 3
Event start  1_1_1  end
Time   0.0   39.7   49.9

AUS000012802387
      Step 1 Step 2 Step 3
```

```
Event start  1_1_1  end
Time   0.0   56.4   60.8

HRV000007702394
      Step 1 Step 2 Step 3
Event start  1_1_1  end
Time    0.0   67.4  116.9

BGR000010903115
      Step 1 Step 2 Step 3
Event start  1_1_1  end
Time    0.0  211.3  237.4

IRL000006501691
      Step 1 Step 2 Step 3
Event start  1_1_1  end
Time    0.0  125.9  136.6
```

References

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Borg, I., & Groenen, P. J. (2005). *Modern multidimensional scaling: Theory and applications*. New York, NY: Springer Science & Business Media. doi: 10.1007/0-387-28981-X

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, *6*(1), 76–90.

Chen, Y., Li, X., Liu, J., & Ying, Z. (2019). Statistical analysis of complex problem-solving process data: An event history analysis approach. *Frontiers in Psychology*, *10*, 486. doi: 10.3389/fpsyg.2019.00486

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1724–1734). Association for Computational Linguistics. doi: 10.3115/v1/D14-1179

Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, *13*(3), 317–322.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, *24*(109), 23–26.

Gómez-Alonso, C., & Valls, A. (2008). A similarity measure for sequences of categorical data based on the ordering of common elements. In V. Torra & Y. Narukawa (Eds.), *Modeling decisions for artificial intelligence* (pp. 134–145). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: https://doi.org/10.1007/978-3-540-88269-5_13

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hao, J., Smith, L., Mislevy, R., von Davier, A., & Bauer, M. (2016). Taming log files from game/simulation-based assessments: Data models and data analysis tools. *ETS Research Report Series*, *2016*(1), 1–17.

He, Q., & von Davier, M. (2016). Analyzing process data from problem-solving items with

n-grams: Insights from a computer-based large-scale assessment. In Y. Rosen, S. Ferrara, & M. Mosharraf (Eds.), *Handbook of research on technology tools for real-world skill development* (pp. 749–776). Hershey, PA: Information Science Reference. doi: 10.4018/978-1-4666-9441-5.ch029

Hinton, G., Srivastava, N., & Swersky, K. (2014). *RMSProp: Divide the gradient by a running average of its recent magnitude.* https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations.*

Paradis, E. (2018). Multidimensional scaling with very large datasets. *Journal of Computational and Graphical Statistics*, *27*(4), 935–939.

Patterson, J., & Gibson, A. (2017). *Deep learning: A practitioner's approach.* " O'Reilly Media, Inc.".

Qiao, X., & Jiao, H. (2018). Data mining techniques in analyzing process data: A didactic. *Frontiers in Psychology*, *9*, 2231. doi: 10.3389/fpsyg.2018.02231

Ren, Y., Luo, F., Ren, P., Bai, D., Li, X., & Liu, H. (2019). Exploring multiple goals balancing in complex problem solving based on log data. *Frontiers in Psychology*, *10*, 1975. doi: 10.3389/fpsyg.2019.01975

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. doi: 10.1214/aoms/1177729586

Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, *24*(111), 647–656.

Stadler, M., Fischer, F., & Greiff, S. (2019). Taking a closer look: An exploratory analysis of successful and unsuccessful strategy use in complex problems. *Frontiers in Psychology*, *10*, 777. doi: 10.3389/fpsyg.2019.00777

Tang, S., Peterson, J. C., & Pardos, Z. A. (2016). Deep neural networks and how they apply to sequential education data. In *Proceedings of the third (2016) acm conference on learning@*

scale (pp. 321–324). doi: 10.1145/2876034.2893444

Tang, X., Wang, Z., He, Q., Liu, J., & Ying, Z. (2020). Latent feature extraction for process data via multidimensional scaling. *Psychometrika*, 1–20. doi: https://doi.org/10.1007/s11336-020-09708-3

Tang, X., Wang, Z., Liu, J., & Ying, Z. (2020). An exploratory analysis of the latent structure of process data via action sequence autoencoders. *British Journal of Mathematical and Statistical Psychology*. doi: https://doi.org/10.1111/bmsp.12203

Wang, C., Xu, G., Shang, Z., & Kuncel, N. (2018). Detecting aberrant behavior and item preknowledge: a comparison of mixture modeling method and residual method. *Journal of Educational and Behavioral Statistics*, *43*(4), 469–501. doi: 10.3102/1076998618767123

Wang, X., Liu, Y., Sun, C., Wang, B., & Wang, X. (2015, July). Predicting polarities of tweets by composing word embeddings with long short-term memory. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1343–1353). Beijing, China: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P15-1130 doi: 10.3115/v1/P15-1130

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, S., Tang, X., He, Q., Liu, J., & Ying, Z. (2021). External correlates of adult digital problem-solving behavior: Log data analysis of a large-scale assessment. Retrieved from https://arxiv.org/pdf/2103.15036.pdf

Zhang, S., Wang, Z., Qi, J., Liu, J., & Ying, Z. (2021). Accurate assessment via process data. Retrieved from https://arxiv.org/pdf/2103.15034.pdf

Figures

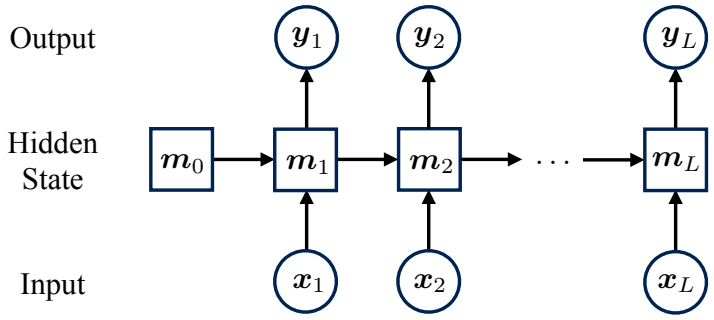

FIGURE 1.

Structure of recurrent neural networks. At each time step, the hidden state $\boldsymbol{m}_t$ is obtained by synthesizing the current $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{m}_{t-1}$.

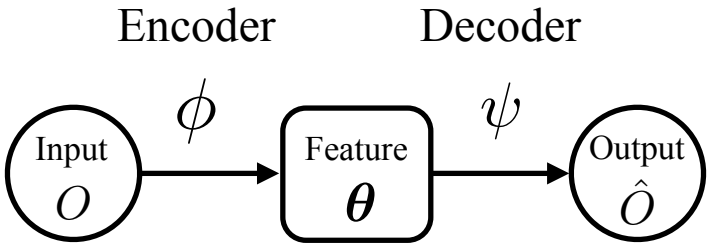

FIGURE 2.

Structure of autoencoders. The encoder ϕ transforms the input $\boldsymbol{o}$ into a low-dimensional vector $\boldsymbol{\theta}$ and the decoder ψ reconstructs $\boldsymbol{o}$ from $\boldsymbol{\theta}$. The low-dimensional vector $\boldsymbol{\theta}$ is the feature vector of $\boldsymbol{o}$.

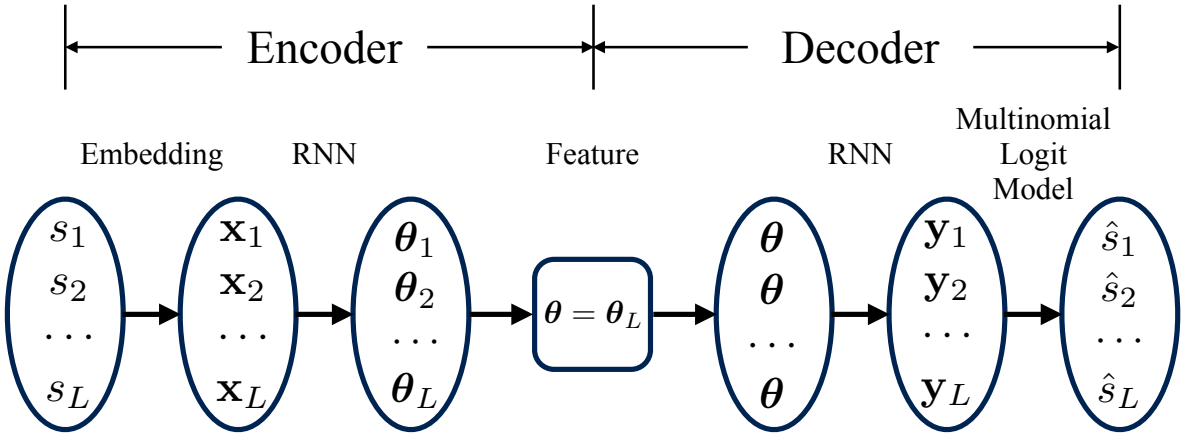

FIGURE 3.

Structure of an action SeqAE that reconstructs an action sequence $\boldsymbol{s} = (s_1, \ldots, s_L)$ as a sequence of probability distributions $\hat{s}_1, \ldots, \hat{s}_L$ on the action set $\mathcal{A}$. The encoder embeds the action sequence into a sequence of real-valued vectors which is then processed by an RNN. The last output vector of the RNN is the feature vector $\boldsymbol{\theta}$ of the action sequence. The decoder reconstructs the action sequence from a sequence of $\boldsymbol{\theta}$'s via an RNN and a multinomial logistic model.

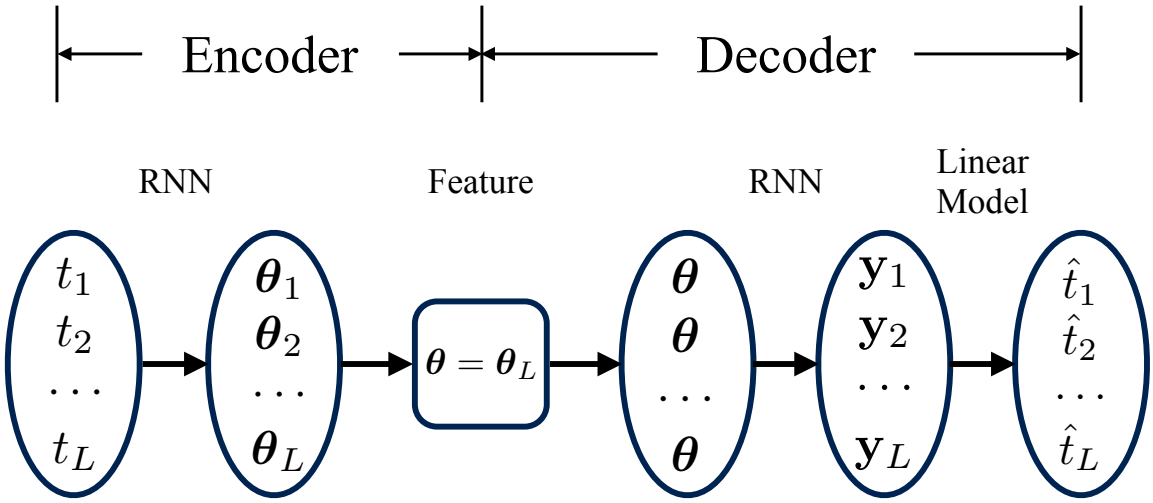

FIGURE 4.

Structure of a time SeqAE that reconstructs a timestamp sequence $\boldsymbol{t} = (t_1, \ldots, t_L)$ as $\hat{\boldsymbol{t}} = (\hat{t}_1, \ldots, \hat{t}_L)$. The encoder processes the timestamp sequence by an RNN. The last output vector of the RNN is the feature vector $\boldsymbol{\theta}$ of the timestamp sequence. The decoder reconstructs the timestamp sequence from a sequence of $\boldsymbol{\theta}$'s via an RNN and a linear model.

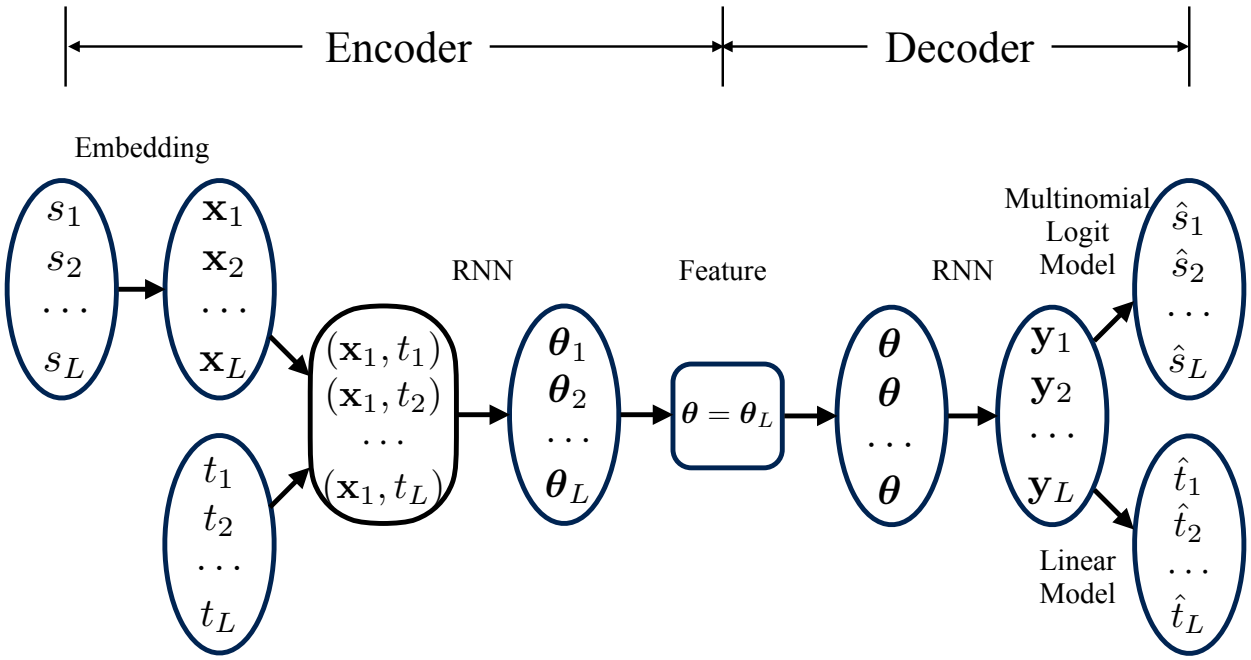

FIGURE 5.

Structure of action-time autoencoders. The encoder embeds the action sequence into a sequence of real-valued vectors, which is combined with the timestamp sequence and passed to an RNN. The last output of the RNN is the feature vector $\boldsymbol{\theta}$ of the action sequence and the timestamp sequence. The decoder reconstructs both the action sequence and the timestamp sequence from a sequence of real-valued vectors processed by an RNN.

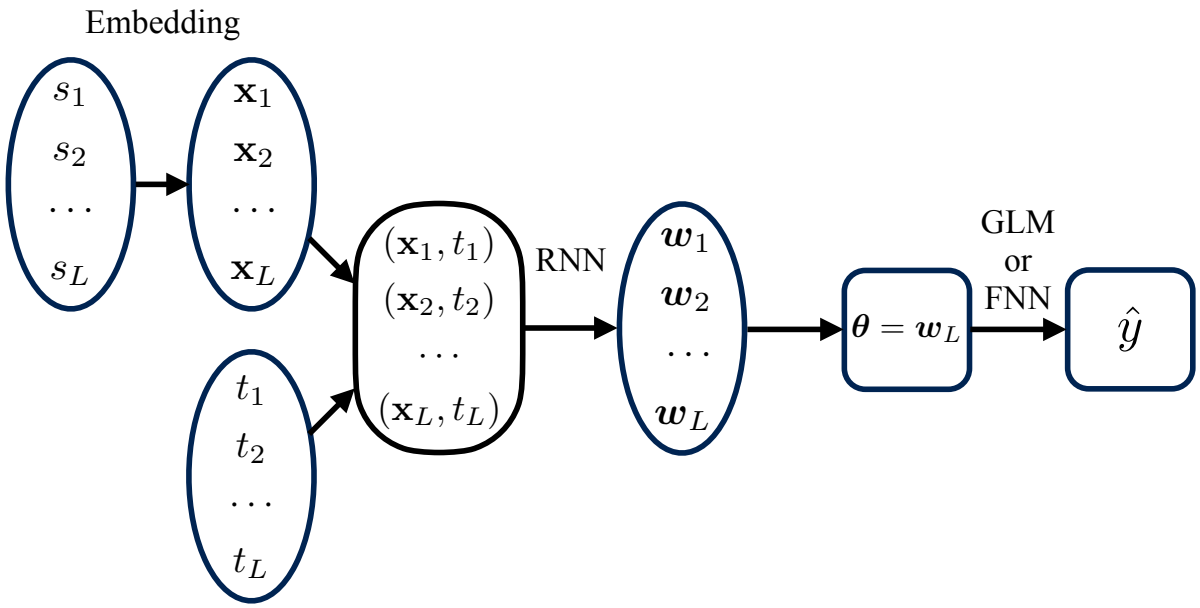

FIGURE 6.

Structure of sequence models. A sequence model embeds an action sequence into a sequence of real-valued vectors and combines it with the timestamp sequence. The combined sequence is then processed by an RNN, the last output of which is the feature vector $\boldsymbol{\theta}$ of the response process related to the response variable. A GLM or FNN is used to describe the relationship between $\boldsymbol{\theta}$ and the response variable y.

	A	B	C	D	E	F
1	Country	Gender	Age	ID	Action	Time
2	US	F	18	101	Start,CHECK_A,End	0,6803,8774
3	US	M	35	102	Start,OPT1_1,OPT2_1,RUN,CHECK_D,End	0,18263,21010,22034,42015,44132
4	US	F	31	103	Start,OPT1_3,OPT2_2,RUN,CHECK_A,End	0,71910,75087,81733,105105,129596
5	JP	M	22	104	Start,OPT1_1,OPT2_2,RUN,OPT1_3,OPT2_2,	0,81733,95466,98860,105105,106560,111(
6	JP	F	40	105	Start,CHECK_C,End	0,3325,3568
7						

FIGURE 7.

Screenshot of a CSV file storing response processes in "single" style.

	A	B	C	D	E	F
1	Country	Gender	Age	ID	Action	Time
2	US	F	18	101	Start	0
3	US	F	18	101	CHECK_A	6803
4	US	F	18	101	End	8774
5	US	M	35	102	Start	0
6	US	M	35	102	OPT1_1	18263
7	US	M	35	102	OPT2_1	21010
8	US	M	35	102	RUN	22034
9	US	M	35	102	CHECK_D	42015
10	US	M	35	102	End	44132

FIGURE 8.

Screenshot of a CSV file storing response processes in "multiple" style.

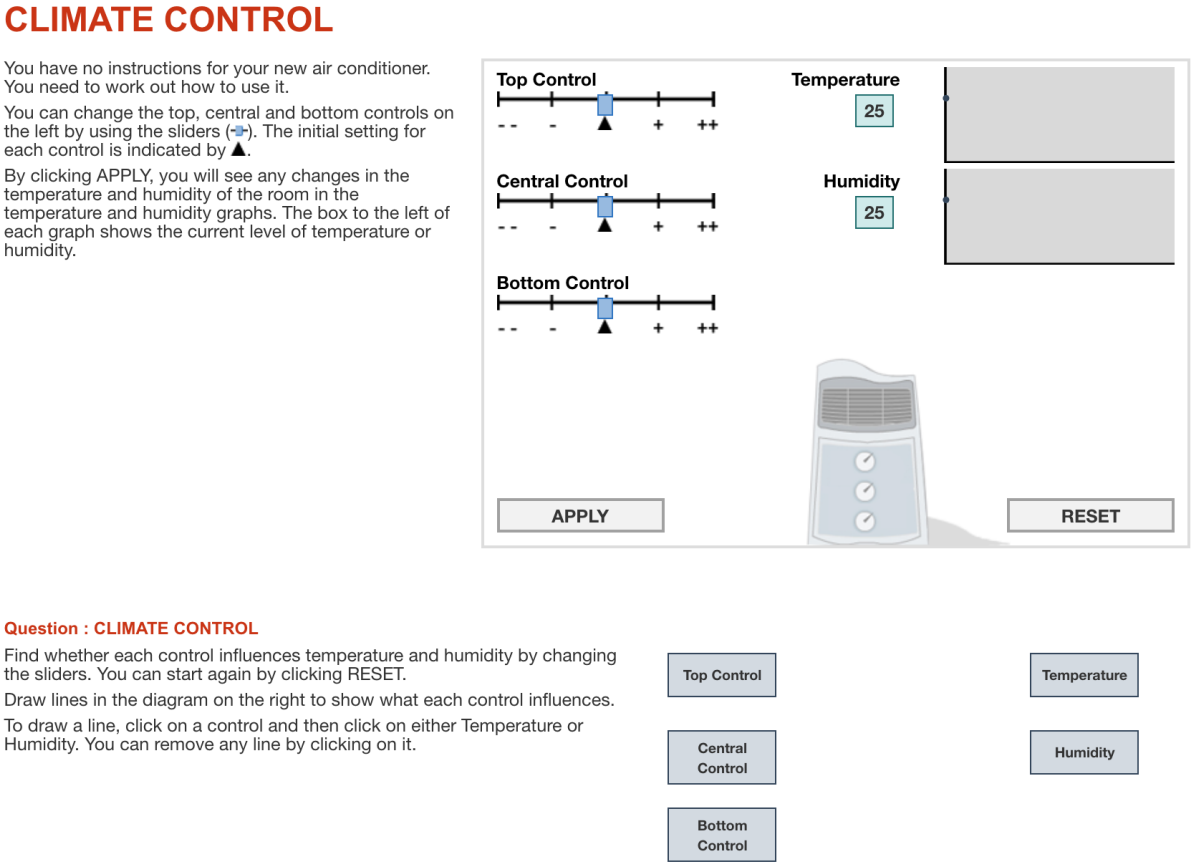

FIGURE 9.

Screenshots of the climate control item in PISA 2012.

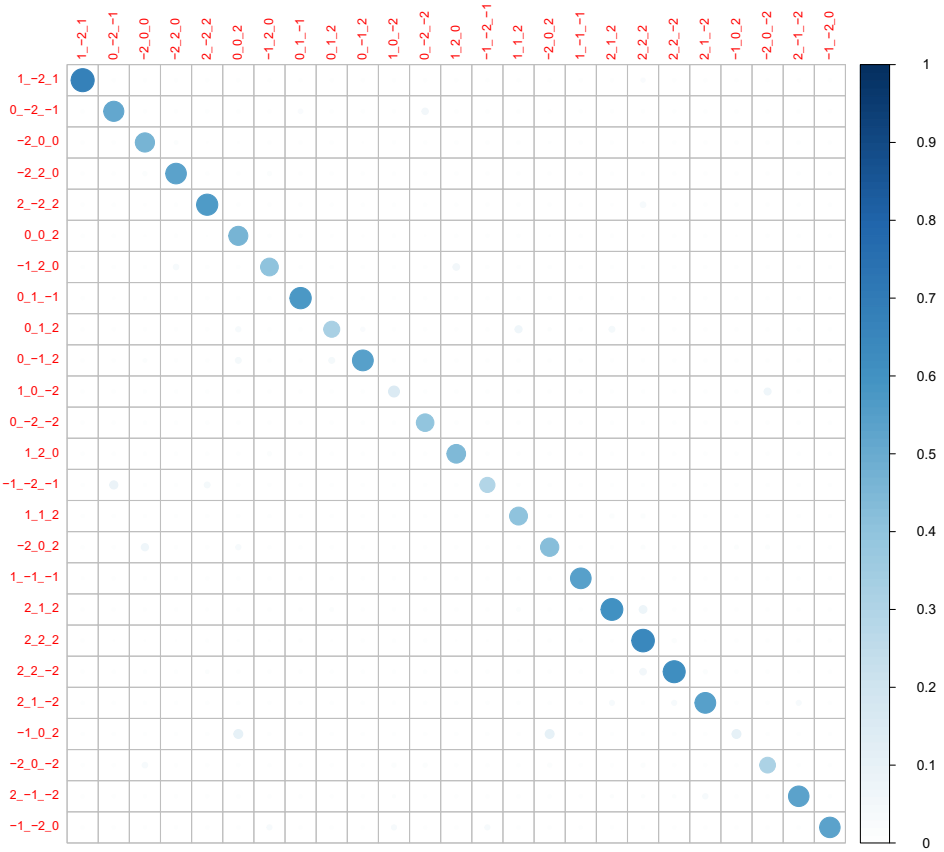

FIGURE 10.

Action transition probability matrix for 25 randomly selected actions in the climate control item.

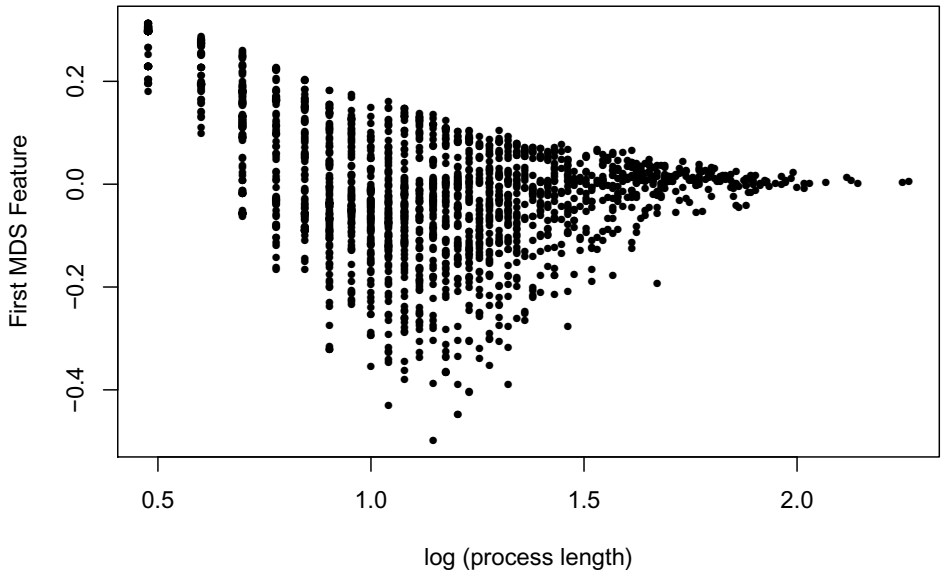

FIGURE 11.

Plot of the first MDS feature against the logarithm of process length in the climate control item example.

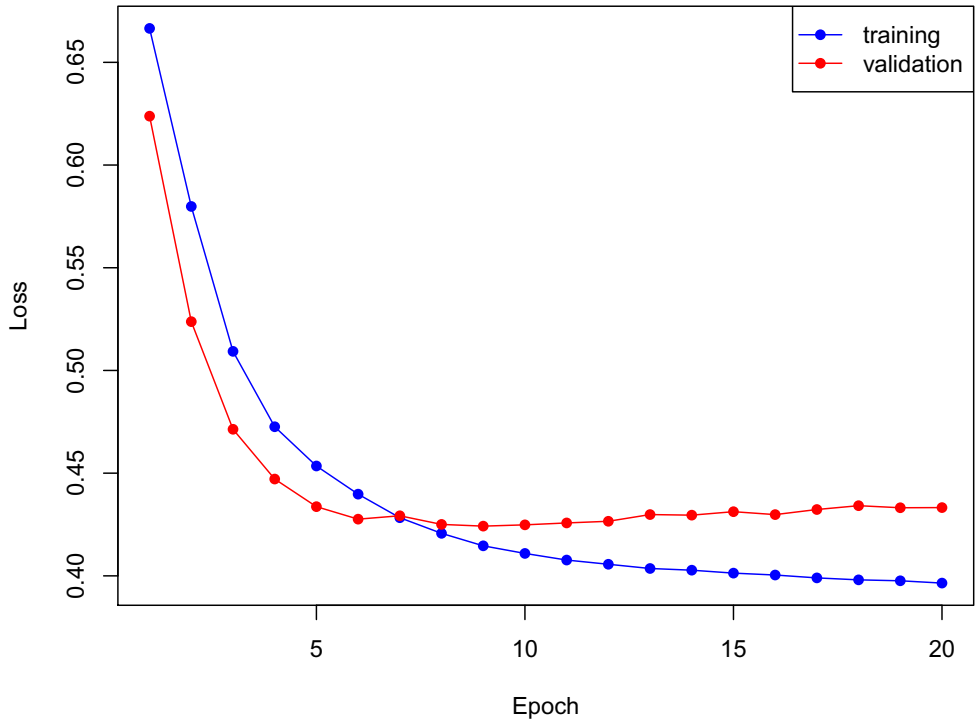

FIGURE 12.

The value of the sequence model objective function on the training and the validation sets at the end of each epoch in the climate control item example.

Tables

TABLE 1.

Summary of `ProcData` features. The functions marked by † depend on the `keras` package.

Topic	Objects
Data structure	`proc`, `print.proc`, `summary.proc`
Data input and output	`read.seqs`, `write.seqs`, `cc_data`
Process manipulation	`remove_repeat`, `remove_action`, `replace_action`, `combine_actions`
Feature extraction	`seq2feature_mds`, `chooseK_mds`, `seq2feature_seq2seq`†, `chooseK_seq2seq`†
Sequence model	`seqm`†, `predict.seqm`†

TABLE 2.

Functions for sequence manipulation.

Function	Description
`sub_seqs()`	subsetting a set of response processes
`remove_repeat()`	removing consecutive repeated actions and their timestamps
`replace_action()`	renaming an action
`remove_action()`	removing a set of actions and their timestamps
`combine_actions()`	combining a given pattern of consecutive actions into a single action