Package: rfsrs (via r-universe)

May 20, 2026

Title R Bindings for FSRS Spaced Repetition Algorithm

Version 0.3.2

Description R bindings for the fsrs-rs Rust library implementing the Free Spaced Repetition Scheduler (FSRS) algorithm. FSRS is a modern spaced repetition algorithm based on the DSR (Difficulty, Stability, Retrievability) model of memory. Includes parameter optimization to train custom parameters from your review history, achieving better scheduling accuracy than default parameters or traditional algorithms like SM-2.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/open-spaced-repetition/r-fsrs, https://chrislongros.r-universe.dev/rfsrs

BugReports https://github.com/open-spaced-repetition/r-fsrs/issues

SystemRequirements Cargo (Rust's package manager), rustc

Config/rextendr/version 0.4.2

Imports R6, jsonlite

Depends R (>= 4.2)

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs libclang-dev

Repository https://chrislongros.r-universe.dev

Date/Publication 2026-04-20 15:48:02 UTC

RemoteUrl https://github.com/open-spaced-repetition/r-fsrs

RemoteRef master

RemoteSha 3e656fb1be7ce85152c3bc8cd87d8d675469d9b8

Contents

`Card` *FSRS Card*

Description

R6 class representing a flashcard

Methods

Public methods:

- `Card$new()`
- `Card$get_retrievability()`
- `Card$to_json()`
- `Card$deep_clone()`
- `Card$print()`
- `Card$clone()`

Method `new()`**:** Create a new Card

Usage:

```
Card$new(due = Sys.time())
```

Arguments:

due Initial due date (default: now)

Method `get_retrievability()`**:** Get current retrievability

Usage:

```
Card$get_retrievability(now = Sys.time())
```

Arguments:

now Reference time (default: Sys.time())

Method `to_json()`**:** Serialize card to JSON

Usage:

```
Card$to_json()
```

Method `deep_clone()`**:** Clone the card

Usage:

```
Card$deep_clone()
```

Method `print()`**:** Print card details

Usage:

```
Card$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Card$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`Card_from_json` *Create Card from JSON*

Description

Create Card from JSON

Usage

```
Card_from_json(json)
```

Arguments

json	JSON string

Value

Card object

fsrs_anki_to_reviews *Convert Anki Review Log to FSRS Format*

Description

Converts an Anki review log (from ankiR or similar) to the format required by `fsrs_optimize`.

Usage

```
fsrs_anki_to_reviews(revlog, min_reviews = 2)
```

Arguments

`revlog`	A data.frame with Anki review data. Supports column names from ankiR (`cid`, `ease`, `id`) or standard names (`card_id`, `rating`, `time`).
`min_reviews`	Minimum number of reviews required per card (default 2).

Value

A data.frame with columns: card_id, rating, delta_t

Examples

```
## Not run:
library(ankiR)
revlog <- anki_revlog()
reviews <- fsrs_anki_to_reviews(revlog, min_reviews = 3)
result <- fsrs_optimize(reviews)

## End(Not run)
```

fsrs_evaluate *Evaluate FSRS Parameters*

Description

Evaluates how well FSRS parameters predict actual recall outcomes.

Usage

```
fsrs_evaluate(reviews, params = NULL)
```

Arguments

`reviews`	A data.frame with columns: card_id, rating, delta_t (same format as `fsrs_optimize`).
`params`	Optional vector of 21 FSRS parameters. Uses defaults if NULL.

Value

List with:

log_loss Log loss metric (may be NaN for some data)

rmse_bins Root mean square error of binned predictions (lower is better)

success Logical indicating if evaluation succeeded

Examples

```
## Not run:
# Compare default vs custom parameters
default_metrics <- fsrs_evaluate(reviews, NULL)
custom_metrics <- fsrs_evaluate(reviews, my_params)
cat("Default RMSE:", default_metrics$rmse_bins, "\n")
cat("Custom RMSE:", custom_metrics$rmse_bins, "\n")

## End(Not run)
```

---

fsrs_interval *Next review interval*

Description

Next review interval

Usage

```
fsrs_interval(stability, desired_retention = 0.9, params = NULL)
```

Arguments

`stability`	Memory stability in days (positive numeric).
`desired_retention`	Target recall probability, e.g. 0.9.
`params`	Optional vector of 21 FSRS parameters

Value

Recommended interval in days.

fsrs_memory_state_from_history *Memory state replayed from a rating history*

Description

Replays a sequence of ratings and intervals to produce the final FSRS memory state. When `initial_stability` and `initial_difficulty` are both NULL, the first rating bootstraps the state as for a new card.

Usage

```
fsrs_memory_state_from_history(
  ratings,
  delta_ts,
  initial_stability = NULL,
  initial_difficulty = NULL,
  params = NULL
)
```

Arguments

`ratings`	Integer vector of ratings (1=Again, 2=Hard, 3=Good, 4=Easy).
`delta_ts`	Integer vector of days elapsed before each rating, same length as `ratings`. The first element is typically 0.
`initial_stability`	Optional positive numeric scalar; starting stability before the first rating. Must be supplied together with `initial_difficulty`.
`initial_difficulty`	Optional numeric scalar between 1 and 10; starting difficulty. Must be supplied together with `initial_stability`.
`params`	Optional numeric vector of length 21.

Value

Named list with `stability` and `difficulty`.

fsrs_migrate_sm2 *Migrate an SM-2 card to FSRS*

Description

Migrate an SM-2 card to FSRS

Usage

```
fsrs_migrate_sm2(ease_factor, interval, sm2_retention = 0.9, params = NULL)
```

Arguments

ease_factor	SM-2 ease factor.
interval	Current SM-2 interval in days.
sm2_retention	Retention target used in SM-2 (default 0.9).
params	Optional vector of 21 FSRS parameters

Value

Named list with stability and difficulty.

---

fsrs_new_card_state *Initial memory state for a new card*

Description

Initial memory state for a new card

Usage

```
fsrs_new_card_state(rating, params = NULL)
```

Arguments

rating	Review rating: 1=Again, 2=Hard, 3=Good, 4=Easy.
params	Optional vector of 21 FSRS parameters

Value

Named list with stability and difficulty.

fsrs_next_memory_state *Memory state after a review*

Description

Memory state after a review

Usage

```
fsrs_next_memory_state(
  stability,
  difficulty,
  elapsed_days,
  rating,
  desired_retention = 0.9,
  params = NULL
)
```

Arguments

stability	Positive numeric. Current stability.
difficulty	Current difficulty (1-10).
elapsed_days	Days since last review.
rating	Review rating (1-4).
desired_retention	Target recall probability (default 0.9).
params	Optional vector of 21 FSRS parameters

Value

Named list with `stability` and `difficulty`.

fsrs_optimize *Optimize FSRS Parameters*

Description

Trains custom FSRS parameters from review history using machine learning. This typically improves prediction accuracy by 10-30% compared to defaults.

Usage

```
fsrs_optimize(reviews, enable_short_term = TRUE, verbose = TRUE)
```

Arguments

reviews	A data.frame with columns: **card_id** Unique identifier for each card **rating** Review rating (1=Again, 2=Hard, 3=Good, 4=Easy) **delta_t** Days since previous review (0 for first review)
enable_short_term	Whether to enable short-term memory modeling (default TRUE).
verbose	Print progress messages (default TRUE).

Value

List with:

success Logical indicating if optimization succeeded

parameters Numeric vector of 21 optimized parameters

error Error message if failed, NULL otherwise

n_cards Number of cards used

n_reviews Number of reviews used

Examples

```
## Not run:
reviews <- data.frame(
  card_id = c(1, 1, 1, 2, 2, 2),
  rating = c(3, 3, 4, 2, 3, 3),
  delta_t = c(0, 1, 3, 0, 1, 5)
)
result <- fsrs_optimize(reviews)
if (result$success) {
  print(result$parameters)
}

## End(Not run)
```

---

fsrs_parameters *Default FSRS parameters*

Description

Returns the 21 default FSRS model weights.

Usage

```
fsrs_parameters()
```

Value

Numeric vector of length 21

`fsrs_recall_probability` *Retrievability*

Description

Retrievability

Usage

```
fsrs_recall_probability(stability, elapsed_days)
```

Arguments

`stability`	Memory stability in days (positive numeric).
`elapsed_days`	Days since last review.

Value

Recall probability between 0 and 1.

`fsrs_recall_probability_vec` *Vectorized retrievability*

Description

Vectorized retrievability

Usage

```
fsrs_recall_probability_vec(stability, elapsed_days)
```

Arguments

`stability`	Numeric vector of stability values.
`elapsed_days`	Numeric vector of elapsed days.

Value

Numeric vector of recall probabilities.

fsrs_simulate *Simulate Learning a Card*

Description

Simulate reviewing a card multiple times with given ratings

Usage

```
fsrs_simulate(ratings, params = NULL, desired_retention = 0.9)
```

Arguments

ratings	Vector of ratings for each review
params	Optional custom parameters
desired_retention	Target retention (default 0.9)

Value

data.frame with review history

fsrs_version *Get the FSRS algorithm version used by this package*

Description

Get the FSRS algorithm version used by this package

Usage

```
fsrs_version()
```

Value

Character string, e.g. "FSRS-6"

Rating *FSRS Rating*

Description

Rating values for card reviews

Usage

```
Rating
```

Format

An object of class `list` of length 4.

ReviewLog *FSRS Review Log*

Description

R6 class representing a review log entry

Methods

Public methods:

- `ReviewLog$new()`
- `ReviewLog$to_json()`
- `ReviewLog$print()`
- `ReviewLog$clone()`

Method `new()`**:**

Usage:

```
ReviewLog$new(rating, scheduled_days, elapsed_days, review_datetime, state)
```

Method `to_json()`**:**

Usage:

```
ReviewLog$to_json()
```

Method `print()`**:**

Usage:

```
ReviewLog$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReviewLog$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Scheduler *FSRS Scheduler*

Description

R6 class for scheduling card reviews using FSRS algorithm

Methods

Public methods:

- Scheduler$new()
- Scheduler$preview_card()
- Scheduler$review_card()
- Scheduler$get_card_retrievability()
- Scheduler$to_json()
- Scheduler$print()
- Scheduler$clone()

Method `new()`**:** Create a new Scheduler

Usage:

```
Scheduler$new(
  parameters = NULL,
  desired_retention = 0.9,
  maximum_interval = 36500L,
  enable_fuzzing = FALSE
)
```

Arguments:

`parameters` Optional vector of 21 FSRS parameters. Uses defaults if NULL.

`desired_retention` Target retention rate (default 0.9)

`maximum_interval` Maximum interval in days (default 36500 = 100 years)

`enable_fuzzing` Whether to add random fuzz to intervals (default FALSE)

Method `preview_card()`**:** Preview all four rating outcomes without modifying the card

Usage:

```
Scheduler$preview_card(card, review_datetime = Sys.time())
```

Arguments:

`card` Card object to preview

`review_datetime` Time of review (default: now)

Returns: List with $again, $hard, $good, $easy outcomes

Method `review_card()`**:** Review a card and update its state

Usage:

```
Scheduler$review_card(card, rating, review_datetime = Sys.time())
```

Arguments:

`card` Card object to review (modified in place)

`rating` Rating: 1=Again, 2=Hard, 3=Good, 4=Easy (or use Rating$Good etc.)

`review_datetime` Time of review (default: now)

Returns: List with $card (updated) and $review_log

Method `get_card_retrievability()`**:** Get current retrievability of a card

Usage:

```
Scheduler$get_card_retrievability(card, now = Sys.time())
```

Arguments:

`card` Card object

`now` Reference time (default: now)

Method `to_json()`**:** Serialize scheduler to JSON

Usage:

```
Scheduler$to_json()
```

Method `print()`**:** Print scheduler details

Usage:

```
Scheduler$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Scheduler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Scheduler_from_json` *Create Scheduler from JSON*

Description

Create Scheduler from JSON

Usage

```
Scheduler_from_json(json)
```

Arguments

`json`	JSON string

Value

Scheduler object

`State` *FSRS State*

Description

Card states in the FSRS system

Usage

```
State
```

Format

An object of class `list` of length 4.

Index