

RFscorer

Recency-Frequency based

Product Recommendation Scoring — a Python package

```
pip install rfscorer
```

GitHub: github.com/jiro-iwanaga/rfscorer PyPI: pypi.org/project/rfscorer

rfscorer (v0.5.1): MIT License / This deck (v0.1.1): CC BY 4.0

© 2026 Erdos Inc.

Contents

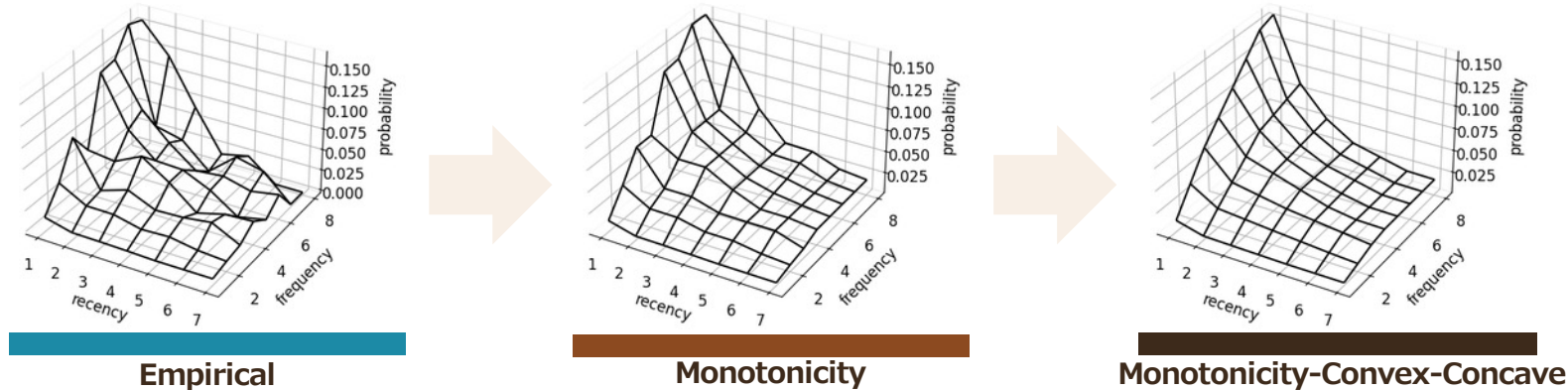
- RF-scoring for product recommendation
- Issues in recommendation systems
- Recommendation system architecture using RF-scoring
- Target users
- Installation & usage
- Input & output data
- Features
- Citation
- Summary
- Appendix: math model, API, tutorials, author

RF-scoring for product recommendation

rfscorer is a Python package that estimates a recommendation score (product-choice probability) for items a user has previously interacted with, based on **recency** and **frequency**.

- **Recency**: more recently viewed items tend to attract more interest
- **Frequency**: more frequently viewed items tend to attract more interest

By estimating product-choice probabilities that satisfy recency–frequency monotonicity via mathematical optimization, it gives an intuitive recommendation order over previously interacted items.



 Based on the paper:

Jiro Iwanaga, Naoki Nishimura, Noriyoshi Sukegawa, and Yuichi Takano, “Estimating product-choice probabilities from recency and frequency of page views,” *Knowledge-Based Systems*, Vol. 99, 2016, pp. 157–167.

Issues in recommendation systems

A recommendation system can be framed as deciding, from past behavior history, which items to prioritize for future recommendation.

Past behavior history

Item	Jul 5	Jul 6
A		1 view
B		2 views
C	1 view	
D	2 views	

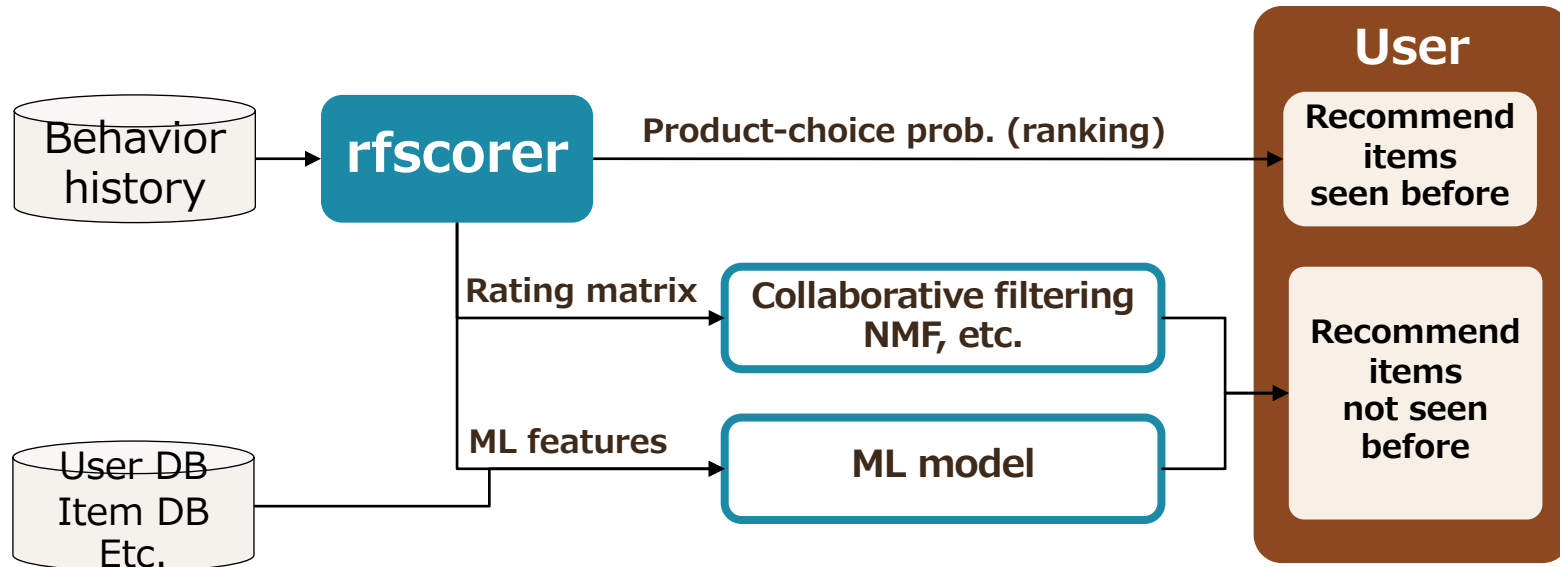
Choosing items to recommend (Jul 7)

- **[By frequency]** A viewed once vs B viewed twice ► B, viewed **more often**
- **[By recency]** A viewed 1 day ago vs C viewed 2 days ago ► A, viewed **more recently**
- **[Trade-off]** A: once, 1 day ago vs D: twice, 2 days ago ► **hard to judge by intuition**

With **rfscorer**, even when recency and frequency trade off, it accounts for their **interaction** while satisfying **recency–frequency monotonicity**, yielding natural product recommendations.

Recommendation system architecture using RF-scoring

The product-choice probabilities from **rfscorer** work not only as a standalone recommendation ranking, but also as input to downstream models (a rating matrix for collaborative filtering, or features for ML models). These features capture the recency–frequency interaction, enabling stronger models.



Based on the paper:

Jiro Iwanaga, Naoki Nishimura, Noriyoshi Sukegawa, and Yuichi Takano, "Improving collaborative filtering recommendations by estimating user preferences from clickstream data," *Electronic Commerce Research and Applications*, Volume 37, Article 100877, 2019.

Target users

Practitioners

- Data scientists & ML engineers
- Marketers & analysts
- Recommender owners on e-commerce / content platforms

Researchers

- Recommender systems & information retrieval
- Marketing science & consumer behavior
- Operations research & mathematical optimization
- Cognitive psychology
(memory, mere-exposure effect)

Installation & usage

pip install

```
$ pip install rfscorer
```

Usage (compute scores in 3 steps: fit → optimize → transform)

```
from rfscorer import RecencyFrequencyScorer, split_by_date

# Split behavior history (cols:user,item,datetime) at a target date into 7-day obs/1-day ground truth
df_obs, df_gt = split_by_date(df, "2026-07-07", 7, 1)

scorer = RecencyFrequencyScorer()
scorer.fit(df_obs, df_gt)          # fit the model
scorer.optimize(kind="mono")      # optimize under RF monotonicity

# score the test data
df_scores = scorer.transform(df_test_obs, "2026-07-07", kind="mono")
```

Input & output data

Input: behavior history with 3 columns

user	item	datetime
u011	i144	2026-07-05
u042	i091	2026-07-06
u044	i077	2026-07-07
u044	i072	2026-07-07

Output: a table adding a score and rank to each user × item

user	item	recency	frequency	probability	order
u011	i144	3	4	0.0767	1
u042	i091	2	3	0.0789	1
u044	i077	1	2	0.0621	1
u044	i072	1	1	0.0248	2

Recommend items to each user in descending probability

Features

- **01 [scikit-learn style]** a simple `fit()` / `transform()` API
- **02 [minimal data]** just three columns: user / item / datetime
- **03 [explainable]** RF monotonicity via optimization explains each recommendation
- **04 [stable probabilities]** estimated directly from recency & frequency
- **05 [downstream use]** as input to collaborative filtering / ML
- **06 [diagnostics & visualization]** statistics & plots for practice and research

scikit-learn style

Provides the familiar `fit()` / `optimize()` / `transform()` interface, fitting naturally into existing ML workflows

HIGHLIGHTS

- Complete in three steps: `fit()` → `optimize()` → `transform()`
- Save and reuse models with `save()` / `load()`

```
scorer = RecencyFrequencyScorer()
scorer.fit(df_obs, df_gt)
scorer.optimize(kind="mono")
df_scores = scorer.transform(df_test_obs, "2026-07-07", kind="mono")

scorer.save("rfscorer.pkl")
scorer_loaded = RecencyFrequencyScorer.load("rfscorer.pkl")
```

Minimal data

All you need is a behavior history with three columns: user, item, datetime
no special preprocessing or extra features required

HIGHLIGHTS

- Most services already log user, item, datetime behavior history
- No additional feature engineering

user	item	datetime
u011	i144	2026-07-07
u042	i091	2026-07-06
u044	i077	2026-07-07
u044	i072	2026-07-07

I Explainable

Estimates scores satisfying RF monotonicity via optimization
not a black box — every recommendation is explainable

HIGHLIGHTS

- “Items viewed more recently or more often are more likely to be chosen” matches intuition
 - Recency: recently viewed items tend to attract more interest
 - Frequency: frequently viewed items tend to attract more interest
- Each score is explained purely by recency and frequency

I Stable probability estimation

Estimates product-choice probabilities directly from recency & frequency, avoiding the instability of rescaling ML outputs into probabilities

HIGHLIGHTS

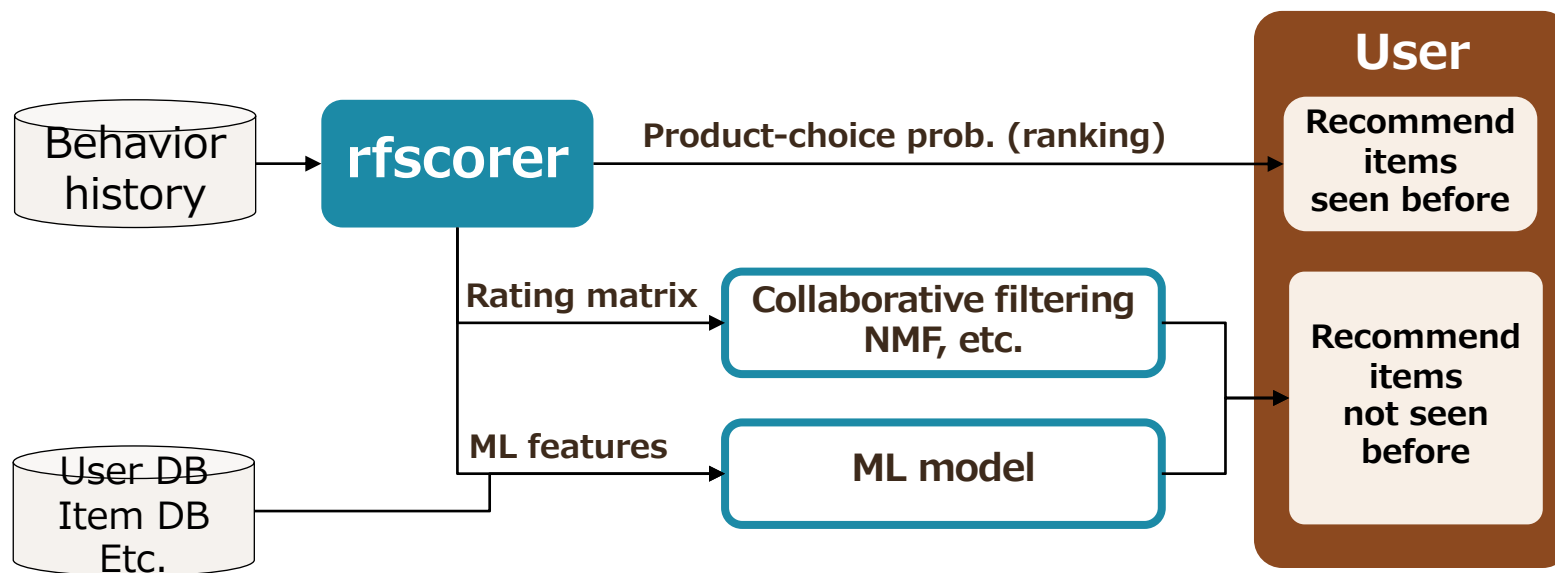
- No fragile probability rescaling, so the values are easy to use — ready for expected-value calculations such as expected revenue
- `fit_rolling()` pools multiple reference dates to stabilize empirical probabilities

Downstream use

Beyond standalone use, the scores work as a rating matrix for collaborative filtering or as features for ML models

HIGHLIGHTS

- Build models on useful features that capture the recency–frequency interaction

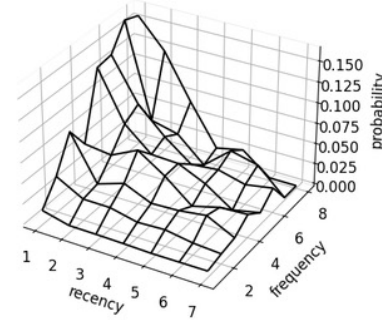


Rich diagnostics & visualization

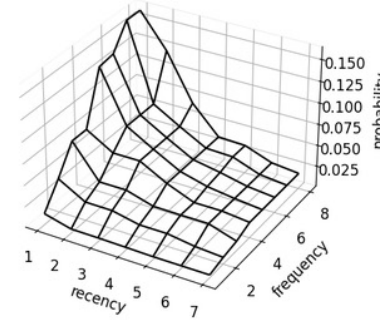
Rich statistics (e.g., correlations) and visualizations easy to explain in practice and to report in research

HIGHLIGHTS

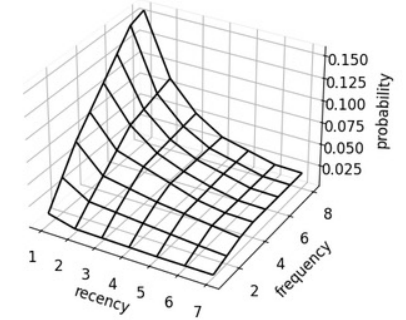
- Output statistics such as correlation coefficients
- Grasp recency \times frequency \times probability intuitively as a 3D surface



Empirical



Monotonicity



Monotonicity-Convex-Concave

Citation

If you use rfscorer in academic work, cite it in your text as follows.

We used rfscorer (Iwanaga et al., 2016), a Python library for Recency-Frequency-based recommendation scoring.¹

¹ <https://github.com/jiro-iwanaga/rfscorer>

References

- Iwanaga, Nishimura, Sukegawa, Takano (2016) “Estimating product-choice probabilities from recency and frequency of page views,” Knowledge-Based Systems, Vol.99, pp.157–167.
- Iwanaga, Nishimura, Sukegawa, Takano (2019) “Improving collaborative filtering recommendations by estimating user preferences from clickstream data,” Electronic Commerce Research and Applications, Vol.37, 100877.

Summary

- ✓ A Python package that estimates product-choice probabilities from recency and frequency
- ✓ Mathematical optimization satisfies RF monotonicity for an interpretable, natural order
- ✓ Works with just user / item / datetime, in a scikit-learn-like API
- ✓ Use it standalone or as input to collaborative filtering / ML
- ✓ Rich diagnostics & visualization support explanation and reporting

`pip install rfscorer`

GitHub: github.com/jiro-iwanaga/rfscorer PyPI: pypi.org/project/rfscorer MIT License

Mathematical optimization model

- Set
 - $R = \{1, 2, 3, \dots, R_{max}\}$
 - $F = \{1, 2, 3, \dots, F_{max}\}$
- Variable
 - $x_{r,f} \in [0, 1] \quad (r \in R, f \in F)$
- Constant
 - $p_{r,f} \in [0, 1] \quad (r \in R, f \in F)$
 - $N_{r,f} \in \mathbb{Z}_{\geq 0} \quad (r \in R, f \in F)$
 - $\epsilon \geq 0$
- Constraint
 - Recency Monotonicity
 - $x_{r,f} \geq x_{r+1,f} + \epsilon \quad (f \in F, r, r+1 \in R)$
 - Frequency Monotonicity
 - $x_{r,f} + \epsilon \leq x_{r,f+1} \quad (r \in R, f, f+1 \in F)$
 - Recency Convex
 - $x_{r,f} - x_{r+1,f} \geq x_{r+1,f} - x_{r+2,f} \quad (f \in F, r, r+1, r+2 \in R)$
 - Frequency Concave
 - $x_{r,f+1} - x_{r,f} \geq x_{r,f+2} - x_{r,f+1} \quad (r \in R, f, f+1, f+2 \in F)$
- Objective (Minimize)
 - $\sum_{r \in R, f \in F} N_{r,f} \cdot (p_{r,f} - x_{r,f})^2$

Optimization models

- **mono**: Monotonicity
- **mrc** : Monotonicity & Recency Convex
- **mfc** : Monotonicity & Frequency Concave
- **mcc**: Monotonicity & Convex & Concave

Optimization packages

- **Modeling**: CVXPY
- **Solver**: Clarabel

API

API	Description
<code>split_by_date(df, date, obs, gt)</code>	Split behavior history into observation / ground-truth by time
<code>RecencyFrequencyScorer()</code>	Create a scorer
<code>.fit(df_obs, df_gt)</code>	Aggregate empirical probabilities by recency \times frequency
<code>.fit_rolling(...)</code>	Rolling aggregation over reference dates to stabilize empirical probabilities
<code>.optimize(kind="mono")</code>	Estimate scores satisfying monotonicity etc. via optimization
<code>.transform(df_obs, date, kind)</code>	Compute scores (probability / order)
<code>.predict(r, f, kind)</code>	Return the probability for a given (recency, frequency)
<code>.evaluate(df_rec, df_gt)</code>	Evaluate recommendation accuracy
<code>.plot_probability_surface(kind)</code>	Visualize the probability surface in 3D
<code>.save(path) / .load(path)</code>	Save / load the model

Tutorials

Beginner

- [tutorial_beginner_en.ipynb](#)

The fastest way to grasp fit → optimize → transform: load data, fit() empirical probabilities, optimize() under RF constraints, visualize the surface, and transform() to scores.

Practical

- [tutorial_practical_en.ipynb](#)

A production-style workflow: split data chronologically, build all 9 models (empirical + optimized), compare accuracy with evaluate(), and save/load the trained model.

Advanced

- [tutorial_advanced_fit_rolling_en.ipynb](#)

Uses fit_rolling() to pool multiple reference dates and stabilize empirical probabilities.

- [tutorial_advanced_int_time_col_en.ipynb](#)

Handling time-series data without calendar dates (integer time column).

- [tutorial_advanced_day_freq_en.ipynb](#)

Caps frequency by counting distinct days an item was viewed instead of total views.

Author

IWANAGA Jiro

- CEO, Erdos Inc.
- Specially Appointed Professor, The University of Electro-Communications

Expertise

- Mathematical optimization, recommender systems, machine learning, NLP

Selected papers

- Iwanaga et al. (2016), Knowledge-Based Systems
Estimating product-choice probabilities from recency and frequency of page views
- Iwanaga et al. (2019), Electronic Commerce Research and Applications
Improving collaborative filtering recommendations by estimating user preferences from clickstream data

Book

- Iwanaga, Ishihara, Nishimura, Tanaka (2021), Ohmsha, Pythonではじめる数理最適化 (in Japanese)

Contact

- GitHub: <https://github.com/jiro-iwanaga>
- Mail: iwanaga@erdos-the-book.com

