ASReview Software Documentation

Release 0.1.dev50+g05b68eb63

ASReview LAB developers, Utrecht University

Jun 03, 2026

CONTENTS

Join the movement towards fast, open, and transparent systematic reviews.

Fast screening

Powerful and customizable AI to screen large volumes of text, achieving up to 95% time savings.

https://doi.org/10.1038/s42256-020-00287-7 Transparant and reproducible

Engage in a fully open and transparent interaction with the AI where you are the oracle.

https://osf.io/preprints/psyarxiv/g93zf Customizable interface

Customize your review environment, offering light and dark modes, adding tags and AI-aided highlighting.

Simulate and validate

Utilize the robust simulation tool to predict AI performance and validate your, another reviewer's or LLM findings

lab/simulation_overview.html CLI and API

Access extensive command-line options and API integration for advanced customization.

technical/index.html Community

Join a community where open discussions and shared insights contribute to the continuous improvement.

https://github.com/asreview/asreview/discussions

CHAPTER

ONE

ASREVIEW LAB EXPLAINED IN 2 MINUTES

https://youtu.be/k-a2SCq-LtA

CHAPTER

TWO

USER GUIDE

2.1 ASReview LAB guide

ASReview LAB is a user-friendly, web-based application designed to make the systematic review process more efficient. Built on the powerful ASReview Python package, it leverages state-of-the-art active learning algorithms to make systematic reviews faster, more efficient, and transparent. Whether you are conducting a systematic review for academic research, policy-making, or other purposes, ASReview LAB provides an intuitive interface and robust tools to support your work.

The application is highly versatile, enabling users to screen large datasets, simulate review processes, and manage projects with ease. By integrating machine learning, ASReview LAB reduces the workload of reviewers while maintaining high accuracy and reproducibility.

See also

Questions can be asked on GitHub Discussions. For bug reports and feature requests, please submit an issue on GitHub.

2.1.1 Introduction

ASReview LAB is an open-source machine learning tool designed to streamline the systematic screening and labeling of large textual datasets. It is widely used for tasks such as title and abstract screening in systematic reviews or meta-analyses, but its applications extend to any scenario requiring systematic text screening.

With ASReview LAB, you can:

Feature	Description
Review	Interactively screen textual data with an active learning model, where the user acts as the 'oracle' to make labeling decisions. You can also validate labels provided by other screeners or AI models.
Simulate	Assess the performance of active learning models using fully labeled datasets.

ASReview LAB is a flagship product of Utrecht University's AI Lab "AI-aided Knowledge Discovery". It has fostered a vibrant global community of researchers, users, and developers.

https://youtu.be/k-a2SCq-LtA

What is active learning?

Artificial Intelligence (AI) and machine learning have allowed the development of AI-aided pipelines that assist in finding relevant texts for search tasks. A well-established approach to increasing the efficiency of screening large amounts of textual data is screening prioritization through Active Learning: a constant interaction between a human who labels records and a machine learning model which selects the most likely relevant record based on a minimum training dataset. The active learning cycle is repeated until the annotator is sufficiently confident they have seen all relevant records. Thus, the machine learning model is responsible for ranking the records, and the human provides the labels. This is called Researcher-In-The-Loop (RITL).

It allows the screening of large amounts of text in an intelligent and time-efficient manner. ASReview LAB, published in Nature Machine Intelligence, has shown the benefits of active learning, reducing up to 95% of the required screening time.

Products

ASReview offers the following tools and resources:

1. **ASReview LAB**: An open-source, browser-based software for AI-aided systematic screening of textual data, such as systematic reviews or meta-analyses. It supports various feature extractors and classifiers. Learn more in the Nature Machine Intelligence publication.
2. **ASReview LAB Server**: A self-hosted solution extending ASReview LAB with features like authentication and AI-aided screening with multiple reviewers.
3. **Datasets**: Access a collection of datasets for research purposes, including the Synergy dataset available on the SYNERGY repository.
4. **Extensions**: Extend ASReview LAB with new models, subcommands, and datasets. Officially supported extensions include:
   - ASReview-dory: Advanced models and components for systematic review screening.
   - ASReview-insights: Advanced insights and performance metrics for simulations.
   - ASReview-makita: Workflow generator for simulation studies.

   For community-maintained extensions, see the List of extensions. To develop your own extension, refer to *Developing Extensions*.

General workflow with ASReview

Start and finish a systematic labeling process with ASReview LAB by following these steps:

1. *Prepare your data*. Your dataset includes relevant records that you aim to identify systematically.
2. *Start ASReview LAB*
3. *Start a review*
4. *Select Prior Knowledge* if available.
5. Start *Screening*
6. Specify a stopping criterion. The dashboard can be used for this.
7. At any time, you can export the resulting *dataset* with the labeling decisions or the entire *project*.

ASReview LAB terminology

When you do text screening for a systematic review in ASReview LAB, it can be useful to know some basic concepts about systematic reviewing and machine learning. The following overview describes some terms you might encounter as you use ASReview LAB.

Active learning model
: An active learning model is a machine learning model that is used to prioritize the records (*record*) in the *dataset*. The model interactively learns from the labels provided by the *user* and improves its performance over time.

CLI
: The CLI is the Command Line Interface that is used to start ASReview LAB and perform various other tasks.

Dataset
: A dataset is the collection of records (*record*) that the *user* reviews.

ELAS
: ELAS stands for "Electronic Learning Assistant". It is the name of ASReview's mascot. ELAS is used for storytelling and enhancing explainability.

Extension
: An extension is an additional element to the ASReview LAB, such as the ASReview Dory extension.

Labeling tags
: Labeling tags allow the *user* to categorize records (*record*) based on specific criteria, such as reasons for inclusion and exclusion, study characteristics, or quality assessment.

Note
: A note allows the *user* to add custom comments or observations about a *record* during the *review* process. This note is stored in the *project* and can be used to clarify decisions or track reasoning. It can be edited on the Collection page.

Prior knowledge
: Prior knowledge refers to labeled records (*record*) that are used to train the machine learning model in ASReview. This knowledge can be provided at the beginning of the screening process or added at any point during the *review*. It helps guide the model in ranking the most likely relevant records, thereby improving the efficiency and focus of the screening process.

Project
: A project is created in ASReview LAB and can be a "review" or a "simulation". It contains the *dataset*, *Active learning model*, and the *user* labels. A project can be exported to an ASReview file with extension `.asreview`. The project can be imported back into ASReview LAB.

Status
: The project status is the stage that a *project* is at in ASReview LAB. Projects can be in one of the following statuses:

  - In review: The project is in the process of being labeled by the user.
  - Finished: The project has been completed by the user or the simulation has been completed.

Simulation
: A simulation is a project that is used to test the performance of the *Active learning model* on a fully labeled *dataset*. The simulation allows the *user* to evaluate the performance of the model and compare it to other models.

Record
: A record is the unit of text that requires labeling in ASReview LAB. It typically includes a title and an abstract, and may also contain keywords. In academic contexts, a record usually refers to the title and abstract of a paper. In other domains, it can represent any text snippet that needs to be labeled.

Review
: Reviewing is the decision-making process on the relevance of *record* ("relevant", "irrelevant"). The term reviewing is interchangeable with Labeling, Screening, and Classifying.

User
: The user is the human annotator or screener who labels records (*record*).

Key principles

ASReview LAB is built on a foundation of core principles that ensure its effectiveness, transparency, and usability. These principles guide the design and functionality of the tool, empowering users to conduct systematic reviews with confidence and efficiency. The five fundamental principles are:

1. Humans are the oracle;
2. Code is open and results are transparent;
3. Decisions are unbiased;
4. The interface clearly communicates the presence of AI;
5. Users are responsible for importing high-quality data.

Privacy

The ASReview LAB software doesn't collect any information about its usage or its user. Great, isn't it!

2.1.2 Open Science

The open source ASReview LAB software is one of the products of Utrecht University's AI Lab "AI-aided Knowledge Discovery". The lab focuses on developing efficient, open, and trustworthy human-AI interaction for processing large amounts of text data. Emphasizing open-source and community-driven efforts, our collaborative research aims to reduce the time and resources needed to process text data while increasing transparency, reproducibility, and explainability.

> **Note**
>
> The ASReview project is developed by researchers for researchers, and anyone is welcome to join the community!

There is still lots of research that can be published using the ASReview products. We welcome researchers worldwide to discover and conduct new research, such as applying the existing models to new types of datasets (different scientific fields, other languages, multilingual data, text data outside academia, large datasets, etc.), adding new models and testing their performance on the available benchmark datasets, introducing and evaluating new stopping rules or performance metrics, and more!

Scientific principles

The research team works according to the Open Science principles and invests in an inclusive community contributing to the project. In short, research is conducted according to the following fundamental principles:

- Research output should be FAIR (Findable Accessible Interoperable and Reusable).
- Research should be conducted with integrity, and we commit ourselves to the Netherlands Code of Conduct for Research Integrity.
- Output should be rewarded according to the Declaration on Research Assessment (DORA).

Utrecht University has established specific regulations governing conduct for its employees. These are based on the key principles of professional and quality academic conduct and ethically-responsible research. Members of the team

employed by Utrecht University, commit themselves to these regulations in all their conduct, including all work related to ASReview. Adherence to similar key principles is expected of all researchers involved in all facets of the ASReview project.

Cite

For scientific use, we encourage users to cite:

> van de Schoot, R., de Bruin, J., Schram, R. et al. An open source machine learning framework for efficient and transparent systematic reviews. Nat Mach Intell 3, 125–133 (2021). https://doi.org/10.1038/s42256-020-00287-7

For citing the software itself, we recommend to use the Cite button on the project overview page of ASReview LAB. This will included the version of the software you used.

For a detailed description of the the data model and it's reproducibility features, we refer to the

> Lombaers, P., de Bruin, J., & van de Schoot, R. (2024). Reproducibility and Data Storage for Active Learning-Aided Systematic Reviews. Applied Sciences, 14(9), 3842. https://doi.org/10.3390/app14093842

More studies related to the project can be found on asreview.ai/research.

Collaborate

Are you interested in (scientific) collaboration? You can contact Prof. Dr. Rens van de Schoot or send an email to asreview@uu.nl.

Contribute

We warmly welcome contributions from everyone, whether you're a seasoned developer, a researcher, or just someone passionate about open science! There are many ways to get involved, and it might be easier than you think.

- **Ask Questions**: If you're curious about ASReview or have questions, feel free to ask on GitHub Discussions. Your questions might help others too!
- **Report Issues**: Found a bug or have a feature request? Submit an issue on GitHub. Clear and detailed reports are incredibly valuable.
- **Share Your Experiences**: Write a blog post, tweet about your experience, or share your use case on the Discussion platform. Your story could inspire others to use ASReview in innovative ways.
- **Contribute Code**: If you're a developer, check out our CONTRIBUTING.md for instructions on how to get started. We also have detailed guidelines for developers.
- **Improve Documentation**: Help us make our documentation better! If you spot errors or think something could be explained more clearly, feel free to suggest changes.
- **Test New Features**: Try out new features and provide feedback. Your input helps us improve the software.
- **Spread the Word**: Share ASReview with your colleagues, friends, or on social media. The more people know about it, the bigger our community grows!

In a blog post, we list some simple examples for first-time contributors. Even small contributions, like answering user questions, are greatly appreciated.

> **Note**
>
> All contributions, whether small or large, are greatly appreciated! Together, we can make ASReview even better.

Donate

ASReview is Free and Open Source Software (FOSS). To support its development, you can make a donation on the ASReview donation page. Even small contributions are highly appreciated!

2.1.3 Installation

Install ASReview LAB

ASReview LAB requires Python 3.10 or later. Detailed step-by-step instructions for installing Python and ASReview LAB are available for Windows and macOS/Linux users.

To install ASReview LAB using Pip, run the following command in *CMD.exe* (Windows) or *Terminal* (macOS/Linux):

```
pip install asreview
```

To start the application, use this command in *CMD.exe* or *Terminal*:

```
asreview lab
```

The ASReview LAB application will open in your browser. For more options to start ASReview LAB, see *Start ASReview LAB*.

Note

Refer to *Troubleshooting* for solutions to common installation issues.

Upgrade ASReview LAB

To upgrade ASReview LAB, run:

```
pip install --upgrade asreview
```

Uninstall ASReview LAB

To uninstall ASReview LAB, run:

```
pip uninstall asreview
```

When prompted, enter `y` to confirm.

Warning

Uninstalling ASReview LAB does not delete your project files. These files are stored in the *.asreview* folder in your home directory.

Server Installation

You can run ASReview LAB on a server or custom domain. Use the *ip* and *port* flags for configuration. ASReview LAB should only be used in closed networks.

```
asreview lab --port 5555 --ip xxx.x.x.xx
```

> **Warning**
>
> For use in production, we recommend to follow the *Installation* instructions of ASReview LAB Server.

Install with Docker

ASReview LAB is also available as a Docker container. Ensure Docker is installed on your machine.

To install and start ASReview LAB at http://localhost:5000, run:

```
docker run -p 5000:5000 ghcr.io/asreview/asreview:latest lab
```

You can pass advanced command-line options as follows:

```
docker run -p 9000:9000 ghcr.io/asreview/asreview lab --port 9000
```

> **Tip**
>
> ASReview LAB is now installed. Open `http://localhost:5000` in your web browser to get started.

Mount Local Volume

To mount the container to your local project folder (or any other folder), use the *-v* flag. Replace *path-to-your-folder* with the path to your local folder. When a project folder is specified, ASReview LAB will store and load all projects from this folder. Multiple containers can access the same folder.

```
docker run -p 5000:5000 -v path-to-your-folder:/project_folder
ghcr.io/asreview/asreview lab
```

Named Container

To simplify usage, create a named container:

```
docker create --name asreview-lab -p 5000:5000 -v
path-to-your-folder:/project_folder ghcr.io/asreview/asreview lab
```

To start ASReview LAB, run:

```
docker start asreview
```

To stop it, replace *start* with *stop*. You can check running containers with *docker ps*.

Customize the Image

To add extensions or build the Docker image yourself, modify the Dockerfile. After making changes, build and run the image with:

```
docker build -t asreview/asreview:custom . docker run -p 5000:5000
ghcr.io/asreview/asreview:custom lab
```

2.1.4 Start ASReview LAB

After you install ASReview LAB, start the program via the command line to start using it.

```
asreview lab
```

When you are using Windows, open *CMD.exe* and run the command. When you use MacOS or Linux, you can open *Terminal* and run the command.

Read the following sections for advanced options to start or configure ASReview LAB users.

Command line arguments for starting ASReview LAB

ASReview LAB provides a powerful command line interface for running ASReview LAB with other options or even run tasks like simulations. For a list of available commands in ASReview LAB, type `asreview lab --help`.

asreview lab launches the ASReview LAB software (the frontend).

```
asreview lab [options]
```

-h, --help
Show help message and exit.

--host HOST
The host/IP address the server will listen on.

--port PORT
The port the server will listen on.

--enable-auth ENABLE_AUTH
Enable authentication. Deprecated.

--secret-key SECRET_KEY
Secret key for authentication. Deprecated.

--salt SALT
When using authentication, a salt code is needed for hasing passwords.

--config-path CONFIG_PATH
Path to a TOML file containing ASReview parameters.

--no-browser NO_BROWSER
Do not open ASReview LAB in a browser after startup.

--port-retries NUMBER_RETRIES
The number of additional ports to try if the specified port is not available.

--certfile CERTFILE_FULL_PATH
The full path to an SSL/TLS certificate file.

--keyfile KEYFILE_FULL_PATH
The full path to a private key file for usage with SSL/TLS.

--skip-update-check
Skip checking for updates.

Set environment variables

The following environment variables are available.

ASREVIEW_PATH

The path to the folder with project. Default *~/.asreview*.

How you set environment variables depends on the operating system and the environment in which you deploy ASReview LAB.

In MacOS or Linux operating systems, you can set environment variables from the command line. For example:

```
export ASREVIEW_PATH=~/.asreview
```

On Windows, you can use the following syntax:

```
set ASREVIEW_PATH=~/.asreview
```

To check if you set an environment variable successfully, run the following on *nix operating systems:

```
echo $ASREVIEW_PATH
```

Or the following on Windows operating systems:

```
echo %ASREVIEW_PATH%
```

Run ASReview LAB on localhost with a different port

By default, ASReview LAB runs on port 5000. If that port is already in use or if you want to specify a different port, start ASReview LAB with the following command:

```
asreview lab --port <port>
```

For example, start ASReview LAB on port 5001:

```
asreview lab --port 5001
```

Local server with authentication

Note

For production use, it is recommended to use the Docker setup. See the *ASReview LAB Server* section for more information.

The most basic configuration of the ASReview LAB application with authentication is to run the application from the CLI with the `--enable-auth` flag. The application will start with authentication enabled and will create a SQLite database if it does not exist. The database will be stored in the ASReview projects folder. The database contains the user accounts and links them to projects.

Start the application with authentication enabled:

```
asreview lab --enable-auth --secret-key=<secret key> --salt=<salt>
```

where `--enable-auth` forces the application to run in an authenticated mode, `<secret key>` is a string that is used for encrypting cookies and `<salt>` is a string that is used to hash passwords. The `--secret-key` and `--salt` parameters are mandatory if authentication is required.

To create user accounts, one can use the `add-users` command of the `auth-tool` sub command of the ASReview application:

```
asreview auth-tool add-users
```

For more information about auth-tool and creating users, see the section Create user accounts below.

2.1.5 Troubleshooting

ASReview LAB is advanced machine learning software. In some situations, you might encounter unexpected behavior. See below for solutions to common problems.

Unknown Command "pip"

The command line returns one of the following messages:

```
-bash: pip: No such file or directory
```

```
'pip' is not recognized as an internal or external command, operable program or batch␣
↪file.
```

First, check if Python is installed by using the following command:

```
python --version
```

If this doesn't return a version number, then Python is either not installed or not correctly installed.

Most likely, the environment variables aren't configured correctly. Follow the step-by-step installation instructions on the ASReview website (Windows and MacOS).

However, there is a simple way to resolve incorrect environment variables by adding *python -m* in front of the command. For example:

```
python -m pip install asreview
```

Unknown command "asreview"

In some situations, the entry point "asreview" can not be found after installation. First check whether the package is correctly installed. Do this with the command *python -m asreview -h*. If this shows a decryption of the program, use *python -m* in front of all your commands. For example:

```
python -m asreview lab
```

Build dependencies error

The command line returns the following message:

```
"Installing build dependencies ... error"
```

This error typically happens when the version of your Python installation has been released very recently. Because of this, the dependencies of ASReview are not compatible with your Python installation yet. It is advised to install the second most recent version of Python instead. Detailed step-by-step instructions to install Python (and ASReview LAB) are available for Windows and MacOS users.

2.1.6 Prepare your data

ASReview LAB requires a dataset containing a set of textual records (e.g., titles and abstracts of scientific papers, newspaper articles, or policy reports) obtained via a systematic search. The goal is to review all records systematically using predetermined inclusion and exclusion criteria. Also, it should be expected that only a fraction of the records in the dataset is relevant.

Datasets can be unlabeled as well as *Partially labeled data* and *Fully labeled data*. See lab/project_create:Project modes for more information.

The easiest way to obtain a dataset is via a search engine or with the help of a reference manager. See *Compatibility* for reference managers export formats supported by ASReview. For more information about the format of the dataset, see *Data format*.

High-quality data

The algorithms of ASReview LAB work best with high-quality datasets. A high-quality dataset is a dataset with duplicate records removed, and the data is complete. Complete data implies that titles and abstracts are available for all (or most) records. See the ASReview blog Importance of Abstracts for more ideas on composing a high-quality dataset.

Compatibility

Citation Managers

The following table provides an overview of export files from citation managers which are accepted by ASReview.

	.ris	.csv	.xlsx
EndNote		N/A	N/A
Excel	N/A		
Mendeley		N/A	N/A
Refworks		N/A	N/A
Zotero			N/A

- = The data can be exported from the citation manager and imported in ASReview.
- N/A = This format does not exist.

RIS files used for screening in ASReview LAB can be imported back into the reference software and the decision labels can be found in the notes field. For more information see this instruction video.

Note: the RIS-pipeline is extensively tested for reference managers Zotero and EndNote. However, it might also work for other reference managers but is currently not supported.

Note

When using EndNote use the following steps to export a RIS file (.ris):

- In EndNote, click on the style selection dropdown menu from the main EndNote toolbar.
- Click “Select Another Style”.
- Browse to RefMan (RIS) Export and click “Choose”.
- Click on the file menu and select “Export”.
- Pick a name and location for the text file.
- Choose the output format RefMan (RIS) Export and click “Save”.

Search Engines

When using search engines, it is often possible to store the articles of interest in a list or folder within the search engine itself. Thereafter, you can choose from different ways to export the list/folder. When you have the option to select parts of the citation to be exported, choose the option which will provide the most information.

The export files of the following search engines have been tested for their acceptance in ASReview:

	.ris	.tsv	.csv	.xlsx
CINAHL (EBSCO)		N/A	X	N/A
Cochrane		N/A		N/A
Embase		N/A		
Eric (Ovid)	*	N/A	N/A	N/A
OpenAlex		N/A		N/A
Psychinfo (Ovid)	*	N/A	N/A	N/A
Pubmed	X	N/A	X	N/A
Scopus		N/A		N/A
Web of Science		N/A	N/A	N/A

- = The data can be exported from the search engine and imported in ASReview.
- N/A = This format does not exist.
- X = Not supported, (see *Data format* for other options).

* Make sure to uncheck all inclusion options (e.g., "URL") when exporting from Ovid.

Tip

If the export of your search engine is not accepted in ASReview, you can also try the following: import the search engine file first into one of the citation managers mentioned in the previous part, and export it again into a format that is accepted by ASReview.

Systematic Review Software

There are several software packages available for systematic reviewing, see https://www.nature.com/articles/s42256-020-00287-7. Some of them use machine learning, while other focus on screening and management. The overview below shows an overview of alternative software programs and the compatibility with ASReview.

	.ris	.tsv	.csv	.xlsx
Abstrackr		N/A		N/A
Covidence*		N/A		N/A
Distiller	X	N/A	**	**
EPPI-reviewer		N/A	N/A	X
Rayyan		N/A		N/A
Robotreviewer	N/A	N/A	N/A	N/A

- = The data can be exported from the third-party review software and imported in ASReview.
- N/A = This format does not exist.
- X = Not supported.

* When using Covidence it is possible to export articles in `.ris` format for different citation managers, such as EndNote, Mendeley, Refworks and Zotero. All of these are compatible with ASReview.

** When exporting from Distiller and if the following error occurs `Unable to parse string "Yes (include)" at position 0` set the `sort references by` to `Authors`. Then the data can be imported in ASReview.

2.1.7 Data format

To carry out a systematic review with ASReview on your own dataset, your data file needs to adhere to a certain format. ASReview accepts the following formats:

Tabular file format

Tabular datasets with extensions `.csv`, `.tab`, `.tsv`, or `.xlsx` can be used in ASReview LAB. CSV and TAB files are preferably comma, semicolon, or tab-delimited. The preferred file encoding is *UTF-8* or *latin1*.

For tabular data files, the software accepts a set of predetermined column names:

Table 1: Table with column name definitions

Name	Column names	Mandatory
Title	title, primary_title	yes*
Abstract	abstract, abstract note	yes*
Keywords	keywords	no
Authors	authors, author names, first_authors	no
DOI	doi	no
URL	url	no
Included	final_included, label, label_included, included_label, included_final, included, included_flag, include	no

* Only a title or an abstract is mandatory.

Title, Abstract Each record (i.e., entry in the dataset) should hold metadata on a paper. Mandatory metadata are only `title` or `abstract`. If both title and abstract are available, the text is combined and used for training the model. If the column `title` is empty, the software will search for the next column `primary_title` and the same holds for `abstract` and `abstract_note`.

Keywords, Authors If `keywords` and/or `author` (or if the column is empty: `author names` or `first_authors`) are available it can be used for searching prior knowledge. Note the information is not shown during the screening phase and is also not used for training the model, but the information is available via the API.

DOI and URL If a Digital Object Identifier (`DOI`) is available it will be displayed during the screening phase as a clickable hyperlink to the full text document. Similary, if a URL is provided, this is also displayed as a clickable link. Note by using ASReview you do *not* automatically have access to full-text and if you do not have access you might want to read this blog post.

Included A binary variable indicating the existing labeling decisions with `0` = irrelevant/excluded, or `1` = relevant/included. If no label is present, we assume the record is not seen by the reviewer. Different column names are allowed, see the table. The behavior of the labels is different for each mode, see *Fully, partially, and unlabeled data*.

RIS file format

RIS file formats (with extensions `.ris` or `.txt`) are used by digital libraries, like IEEE Xplore, Scopus and ScienceDirect. Citation managers Mendeley, RefWorks, Zotero, and EndNote support the RIS file format as well. See (wikipedia) for detailed information about the format.

For parsing RIS file format, ASReview LAB uses a Python RIS files parser and reader (rispy). Successful import/export depends on a proper data set structure. The complete list of accepted fields and default mapping can be found on the rispy GitHub page.

The labels `ASReview_relevant`, `ASReview_irrelevant`, and `ASReview_not_seen` are stored with the N1 (Notes) tag, and can be re-imported into ASReview LAB. The behavior of the labels is different for each mode, see *Fully, partially, and unlabeled data*.

Tip

The labels `ASReview_relevant`, `ASReview_irrelevant`, and `ASReview_not_seen` are stored with the N1 (Notes) tag. In citation managers Zotero and Endnote the labels can be used for making selections; see the screenshots or watch the instruction video.

Note

When re-importing a partly labeled dataset in the RIS file format, the labels stored in the N1 field are used as prior knowledge. When a completely labeled dataset is re-imported it can be used in the Exploration and Simulation mode.

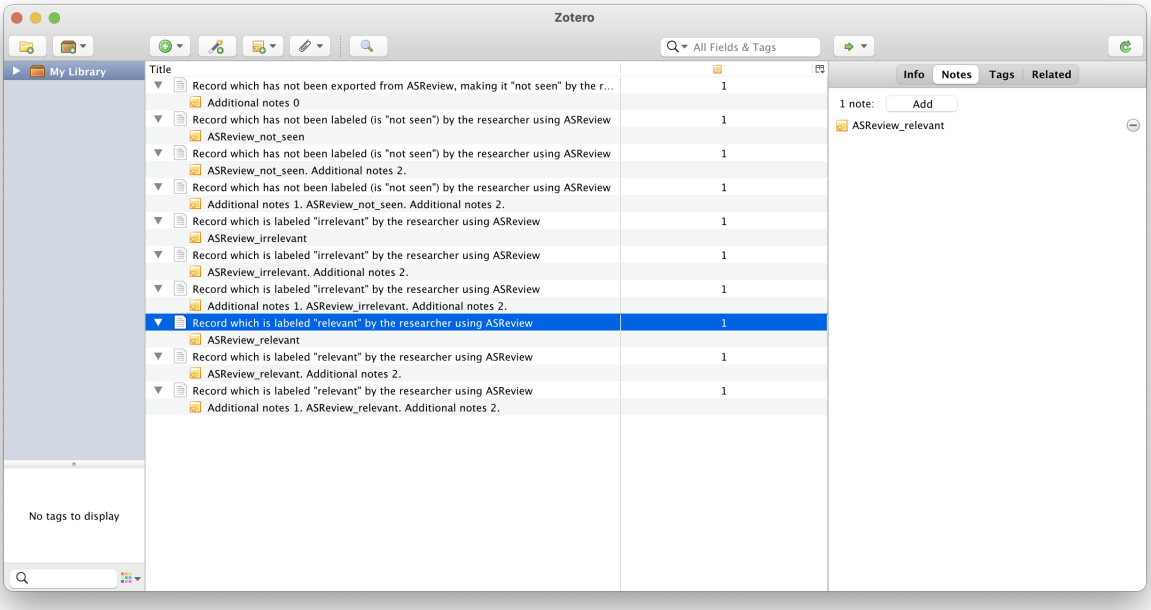


Fig. 1: Example record with a labeling decision imported to Zotero

2.1.8 Fully, partially, and unlabeled data

Fully and partially labeled datasets serve a special role in the ASReview context. These datasets have review decisions for a subset of the records or for all records in the dataset.

Label format

For tabular datasets (*e.g., CSV, XLSX*), the dataset should contain a column called "included" or "label" (See *Data format* for all naming conventions), which is filled with `1`'s or `0`'s for the records that are already screened. The value is left empty for the records that you haven't screened yet, or which are added to the dataset in case of updating a review.

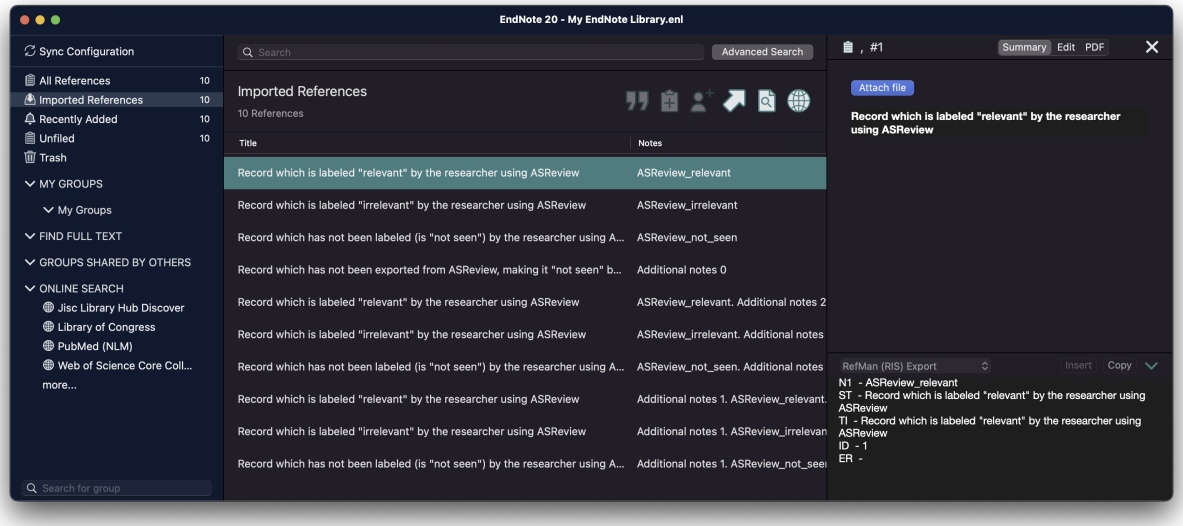


Fig. 2: Example record with a labeling decision imported to Endnote

For the RIS file format, the labels `ASReview_relevant`, `ASReview_irrelevant`, and `ASReview_not_seen`) can be stored with the N1(Notes) tag.

Exported files containing labeling decisions can be re-imported into ASReview LAB whereafter all labels are recognized and its behavior is different for each mode:

- In **Oracle mode** existing labels are used for prior knowledge.
- In **Validation mode** records are presented along with an indication of their previous labeling status: relevant, irrelevant, or not seen. This status is displayed via a color-coded bar above each record.
- In **Simulation** the column containing the labels is used to simulate a systematic review.

Unlabeled data

Unlabeled datasets do not contain any labels and can be used in the **Oracle mode** to start a review from scratch. Prior knowledge has to be selected in the *Prior Knowledge* step of the project set-up.

Partially labeled data

Partially labeled datasets are datasets with a labeling decision for a subset of the records in the dataset and no decision for another subset.

In **Oracle mode**, if labels are available for a part of the dataset, the labels will be automatically detected and used for *Prior Knowledge*. The first iteration of the model will then be based on these decisions and used to predict relevance scores for the unlabeled part of the data. It is usefull when a large number of records is needed for training, or when updating a systematic review, or to continue the screening process with model switching.

In **Validation mode**, the labels available are presented in the review screen along with an indication of their previous labeling status: relevant, irrelevant, or not seen. This status is displayed via a color-coded bar above each record, and you have the opportunity to refine the dataset by correcting any potential misclassifications, useful for the quality evaluation(see, for example, the SAFE procedure).

> **Note**
>
> Merging labeled with unlabeled data should be done outside ASReview LAB, for example, with the compose function of ASReview Datatools, or via *Citation Managers*.

Fully labeled data

Fully labeled datasets are datasets with a labeling decision for all records in the dataset.

In **Simulation mode**, the labels are used for mimicking the review proces for a *Simulation study*. Only records containing labels are used for the simulation, unlabeled records are ignored.

In **Validation mode**, the labels available in a fully labeled dataset are presented in the review screen along with an indication of their previous labeling status: relevant or irrelevant. It is usefull to validate labels as a human when the labels are predicted by a large language model (LLM), like by ChatGPT. Also, one can use this mode for teaching purporses.

Benchmark datasets

The ASReview research project collects fully labeled datasets published open access. The labeled datasets are PRISMA-based systematic reviews or meta-analyses on various research topics. They can be useful for teaching purposes or for testing the performance of (new) active learning models. The datasets and their metadata are available via the SYNERGY Dataset repository. In ASReview LAB, these datasets are found under "Benchmark Datasets"; only available for the Validation and Simulation modi.

The Benchmark Datasets are directly available in the software. During the *Add Dataset* step of the project setup, there is a panel with all the datasets. The datasets can be selected and used directly. Benchmark datasets are also available via the *Simulation via command line*. Use the prefix `synergy:` followed by the identifier of the dataset (see Synergy Dataset repository). For example, to use the Van de Schoot et al. (2018) dataset, use `synergy:van_de_schoot_2018`.

2.1.9 AI Models

AI models in ASReview LAB are the driving force behind efficient and accurate systematic reviews. By learning from your decisions, these models prioritize the most relevant records, significantly reducing the time and effort required for your review process. Whether you're working with single-language datasets, multilingual data, or need advanced semantic understanding, ASReview offers a range of models tailored to your needs.

Each model is built from a combination of components—feature extractors, classifiers, queriers, and balancers—that work together to optimize the review process. You can choose from pre-configured models for simplicity or customize your own for greater flexibility. This guide will help you understand the available models and how to select the best one for your use case.

> **Tip**
>
> Not sure where to start? The ELAS u4 model is a great choice for most users. It's fast, efficient, and performs well across a variety of datasets. It's available by default in ASReview LAB.

ELAS Models

The ELAS models in ASReview LAB are pre-configured AI models designed to cater to a variety of systematic review needs. Whether you need a fast and efficient model, one that handles multilingual datasets, or a model with advanced semantic understanding, the ELAS series has you covered. Each model is built from a combination of components—feature extractors, classifiers, queriers, and balancers—that work together to optimize the review process.

All ELAS models are active learning models. This means they iteratively learn from your labeling decisions and dynamically adjust their predictions to prioritize the most relevant records. Active learning ensures that the review process becomes more efficient over time, focusing on the records that are most likely to be relevant.

Table 2: Model Overview

Model	Short name	Description	Requires
ELAS Ultra	u-series	Rapid and excellent-performing model for most use cases.	•
ELAS Multilingual	l-series	Designed for multilingual datasets.	*ASReview Dory*, *Hardware Requirements*
ELAS Heavy	h-series	Focuses on semantic understanding of text.	*ASReview Dory*, *Hardware Requirements*

For most users, the pre-configured ELAS models are sufficient. However, if you want more control, you can create custom models by mixing and matching components. This flexibility allows you to tailor the AI model to your specific dataset and research goals. Custom models can combine components from both ASReview and the *ASReview Dory* extension, offering advanced options for those with more technical expertise.

ELAS Ultra

The ELAS Ultra AI model in ASReview LAB is the default and most widely used model. It is designed for speed and efficiency, making it ideal for most systematic review tasks. The model leverages "classic" machine learning techniques, which are lightweight and reliable. These techniques are implemented using components from the SciKit-learn library, ensuring robust performance.

Key features of ELAS Ultra:

- **Speed**: Processes data quickly, making it suitable for large datasets.
- **Efficiency**: Balances performance and resource usage, ensuring smooth operation on most systems.
- **Versatility**: Performs well across a wide range of datasets and use cases.

The following table outlines the components of the ELAS Ultra model for its various versions:

Table 3: ELAS Ultra versions

Model	Feature Extractor	Classifier	Querier	Balancer
ELAS u4	TF-IDF (with bigrams)	SVM	Maximum	Balanced
ELAS u3	TF-IDF	Naive Bayes	Maximum	Balanced

Note

While the components of ELAS Ultra models may appear similar across versions, differences in their underlying parameters can significantly impact their performance and behavior. Use the latest version (e.g., ELAS u4) for the best results.

Use ELAS Ultra if you are looking for a reliable, fast, and easy-to-use model that works well for most systematic review scenarios.

ELAS Multilingual

The ELAS Multilingual models are specifically designed for datasets containing multiple languages. These models leverage advanced multilingual feature extractors. They are ideal for systematic reviews involving multilingual datasets, where other ELAS models may struggle with language-specific texts and nuances.

Key features of ELAS Multilingual:

- **Multilingual Support**: Handles datasets with multiple languages seamlessly, supporting over 100 languages.
- **Advanced Feature Extraction**: Uses state-of-the-art multilingual feature extractors for better understanding of text.
- **Flexibility**: Suitable for a wide range of multilingual systematic review tasks.

Requirements for ELAS Multilingual:

- **Dory extension**: The ELAS Multilingual models require the *ASReview Dory* extension for feature extraction. Install the extension using the following command: `pip install asreview-dory`.
- **Hardware**: These models are computationally intensive and may require significant CPU or GPU power to perform efficiently, especially with large datasets. See the section on *Hardware Requirements* for more details.

The following table outlines the components of the ELAS Multilingual model for its various versions:

Table 4: ELAS Multilingual versions

Model	Feature Extractor	Classifier	Querier	Balancer
ELAS l2	multilingual-e5-large	SVM	Maximum	Balanced

For more information about the *multilingual-e5-large* feature extractor, including its support for over 100 languages, visit the official documentation on Hugging Face: https://huggingface.co/intfloat/multilingual-e5-large.

ELAS Heavy

The ELAS Heavy models are designed for tasks requiring advanced semantic understanding of text. These models utilize powerful feature extractors that focus on the underlying meaning of the text, making them ideal for systematic reviews where semantic context is crucial.

Key features of ELAS Heavy:

- **Semantic Understanding**: Focuses on the meaning of text rather than just word occurrences.
- **Advanced Feature Extraction**: Uses state-of-the-art feature extractors for deeper text analysis.
- **Ideal for Complex Reviews**: Suitable for datasets where semantic nuances play a significant role.

Requirements for ELAS Heavy:

- **Dory extension**: The ELAS Heavy models require the *ASReview Dory* extension for feature extraction. Install the extension using the following command: `pip install asreview-dory`.
- **Hardware**: These models are computationally intensive and may require significant CPU or GPU power to perform efficiently, especially with large datasets. See the section on *Hardware Requirements* for more details.

The following table outlines the components of the ELAS Heavy model for its various versions:

Table 5: ELAS Heavy versions

Model	Feature Extractor	Classifier	Querier	Balancer
ELAS h3	mxbai-embed-large-v1	SVM	Maximum	Balanced

For more information about the *mxbai-embed-large-v1* feature extractor and its capabilities, refer to the official documentation provided in the ASReview Dory extension.

Custom ELAS Models

Custom ELAS models allow you to tailor the AI model to your specific needs by combining different components. Each AI model in ASReview LAB is composed of four key components that work together to rank your remaining documents:

- **Querier**: Determines which records to show you next. For example, it can prioritize potentially relevant records, mix in random records, or use uncertainty-based strategies.
- **Feature Extractor**: Converts text into numerical features that the classifier can interpret.
- **Classifier**: Predicts the relevance of records based on your decisions using the numerical features created by the feature extractor.
- **Balancer**: Handles imbalanced data to improve learning accuracy and ensure robust performance.

The following components are available out of the box for creating custom models:

- **Feature Extractors**: *OneHot*, *TF-IDF*
- **Classifiers**: *Naive Bayes*, *Support Vector Machine*, *Random Forest*, *Logistic Regression*
- **Queriers**: *Maximum*, *Mixed (95% Maximum and 5% Random)*, *Mixed (95% Maximum and 5% Uncertainty)*, *Random*, *Top-down*, *Uncertainty*
- **Balancers**: *Balanced*

For advanced users, you can also integrate components from the *ASReview Dory* extension, which provides access to more powerful feature extractors and classifiers:

- **ASReview Dory Feature Extractors**: *doc2vec*, *gtr-t5-large*, *labse*, *multilingual-e5-large*, *mxbai-embed-large-v1*, *sbert*
- **ASReview Dory Classifiers**: *AdaBoost*, *Neural Network - 2-Layer*, *Neural Network - Dynamic*, *Neural Network - Warm Start*, *XGBoost*

Tips for customization:

- Combining components from ASReview and Dory allows for highly flexible and powerful models. However, some feature extractors may not work with certain classifiers. For example, some Dory feature extractors cannot be combined with the ASReview Naive Bayes classifier.
- Experiment with different combinations to find the best fit for your dataset and research goals. You can use the simulation mode in ASReview LAB to evaluate the performance of different models before applying them to your actual dataset.
- Creating custom models requires some knowledge of how the components work. Start with simpler combinations and gradually explore more complex setups as you gain experience.

Hardware Requirements

The hardware requirements for running AI models in ASReview LAB vary depending on the complexity of the model. The ELAS Ultra models are lightweight and can run efficiently on most modern systems, including laptops and desktops, without requiring specialized hardware. In contrast, the ELAS Multilingual and ELAS Heavy models utilize advanced machine learning techniques and feature extractors, making them computationally intensive. These models often require significant CPU or GPU power to perform efficiently, especially when working with large datasets.

For optimal performance, ELAS Multilingual and ELAS Heavy models are better suited for server installations or systems equipped with dedicated GPUs. If you plan to use these models, ensure that your hardware includes a multi-core processor with high clock speed and at least 16 GB of RAM. Some operating systems will also benefit from a

modern GPU for faster processing. Running these models on underpowered hardware may result in slower performance, longer training times, and inefficient screening.

Model Numbering

The ELAS models are numbered with a letter and a number. The letter indicates the type of model, and the number indicates the version. The latest version of each model type is always the one with the highest number. For example, the latest version of the Ultra model is denoted as ELAS uX, where X represents the highest available version number. Not all historical versions are available in ASReview LAB, but you can always use the latest version of the model.

Changing Models

You can change the AI model used in your systematic review at any time. When you switch models, the new model will start training in the background. This process might take some time, depending on the size of your dataset and the complexity of the model. However, you can continue screening records without interruption while the new model is being trained.

To change the model, follow these steps:

1. Go to the **Customize** section in ASReview LAB.
2. Navigate to the **AI** card.
3. Select the desired model from the list of available options.

Once the new model is trained, it will automatically take over and start prioritizing records based on its predictions. In the meantime, you can keep screening records as usual.

> **Note**
>
> Switching to a more complex model, such as those requiring the ASReview Dory extension, may take longer to train.

2.1.10 ASReview Dory

ASReview Dory is an official extension of ASReview LAB that introduces advanced models and components for systematic review screening. Dory is a collection of state-of-the-art tools designed to handle complex datasets and provide more accurate results. These models are optimized for scalability and performance, making them ideal for researchers working with large or multilingual datasets.

Installation

Installing ASReview Dory works like any other Python package. Simply install it using the following command:

```
pip install asreview-dory
```

After installation, *ELAS Heavy* and *ELAS Multilingual* are available for use. Additionally, under the "Custom" option, you can mix and match various feature extractors and classifiers to suit your needs.

> **Tip**
>
> Since these models are generally more computationally intensive, it may take some time for the feature extractor and classifier to process your data.

2.1.11 Start a review

To start reviewing a dataset with ASReview LAB, you create a project containing a dataset with records to screen. The project will contain your dataset, settings, labeling decisions, and machine learning models.

To start a review project, you need to:

1. *Start ASReview LAB*.
2. Go to the *Reviews* if you are not already there (http://localhost:5000/reviews)
3. Upload, select, or choose a dataset to screen.
4. Verify the dataset with the charts. Ensure that the dataset completeness is sufficient.

Add Dataset

The first step in creating a project is to select a dataset. You can upload a dataset from your computer, select a dataset from Discovery, or use a dataset from a URL or DOI. When uploading a dataset from your computer, URL, or DOI, ensure that the dataset is in a supported format. See *Prepare your data* for extensive information about the supported formats and metadata.

> **Tip**
>
> You will benefit most from what active learning has to offer with *High-quality data*.

From File

Drag and drop your file or select your file.

From URL (or DOI)

Provide a URL or a DOI to a dataset. Many data repositories are supported via Datahugger. If the DOI points to multiple files, you can select the specific file you want to use (e.g., 10.17605/OSF.IO/WDZH5).

Click on *Download* to download and add the dataset to the project.

From Discovery

Under Discovery, you can select existing datasets from the SYNERGY dataset or installed dataset extensions. The SYNERGY dataset is a collection of fully labeled datasets that can be used, but not exclusively, to benchmark the performance of active learning models.

More options

Under the dataset card, you find *Show options*. Clicking on *show options* will open extra options for the review.

Add Tags

You can add tags to your records to review. Tags are useful for organizing your records afterwards or for data extraction. You can add tags and tag groups to your review by clicking on the *Add tags* button. You can add multiple tags to a tag group, and you can add multiple tags to a dataset. In the current version, you can't delete tags, so be careful with the tags you add.

Tags are presented to you in the *Reviewer* interface. Tags are presented as checkboxes, and you can select multiple tags for a record. You can find the selected tags in the collection and during the export of the dataset.

Change AI Model

By default, ASReview LAB uses the ELAS ultra model. This is a fast and efficient model that is trained on the SYNERGY dataset. You can change the model to a different model by clicking on the dropdown button. You can select from the following models:

- ELAS ultra
- ELAS multilingual
- ELAS heavy
- Custom

Most users will benefit from the ELAS ultra model and don't need to change the model. The ELAS multilingual model is useful for datasets that are multilingual or contain non-English records.

For more information about the models and the required *ASReview Dory* extension, see the lab/models page.

Prior Knowledge

Prior knowledge refers to records in your dataset that you already know are relevant or irrelevant. Providing prior knowledge helps train the model during the initial and subsequent iterations of the active learning cycle. The model uses this information to generate an initial ranking of records in your dataset.

> **Note**
>
> If your dataset includes *Partially labeled data*, ASReview LAB will automatically use the labeled records as prior knowledge.

To add prior knowledge:

1. Click on *Search* to search your dataset by authors, keywords, titles, or a combination of these.
2. Enter your search terms and press *Enter*. Only the first 10 results will be displayed, so ensure your search terms are precise.
3. Review the record you were searching for and select the relevant or irrelevant label. You can also add tags to the record. Avoid labeling all items; select only those you intend to use as training data.
4. Close the search window or click on *Return* to return to the previous screen.

Providing accurate prior knowledge improves the model's performance and can accelerate the review process.

Screen

Once you have selected a dataset and optionally added tags, changed the model, or searched for prior knowledge, you can click on *Screen* to start the review. For more tips on how to screen records, see *Screening*.

2.1.12 Screening

Once your project is set up, you can immediately begin screening records. To get started, click on *Reviewer* in the left menu if you are not already on the review page. ASReview LAB will present you with a title and abstract to evaluate and label.

Your task is to decide whether the record is relevant or irrelevant. Simply click on your choice, and the next record will be presented to you. While you review, ASReview LAB works in the background, training a model based on your decisions and continuously improving its understanding of your preferences. This process updates the ranking of the remaining records, ensuring that the most relevant records are prioritized for review.

Each labeling decision by the user starts the training of a new model, provided no model is being trained at that time. When this new model is trained, the unseen records' rank order is updated. Training and labeling occur asynchronously. With fast models, a new ranking will likely be available before the user finishes reading the text. With slower models, training continues until a new model is trained, and the user can continue screening the next record in line (2nd, 3rd, etc.).

As you keep reviewing documents and providing labels, you will probably see fewer relevant records. When to stop screening is left to you. See *Progress and results* for more information on progress monitoring and information on when to stop.

> **Tip**
>
> If you are in doubt about your decision, take your time as you are the oracle. Based on your input, a new model will be trained, and you do not want to confuse the prediction model. For the model, it may be better to consult others and read the full text (in the case of reviewing abstracts of scientific papers).

Autosave

Your decisions (and notes) are saved automatically into your ASReview project file. There is no need to press any buttons to save your work anywhere in ASReview LAB (in fact, there is not even a *save* button).

Change decisions

In some cases, you might want to change your previous decision. An overview of your decisions made during screening can be found on the **Collection** page. You can change decisions on this page.

1. *Start ASReview LAB*.
2. Open or *Start a review*.

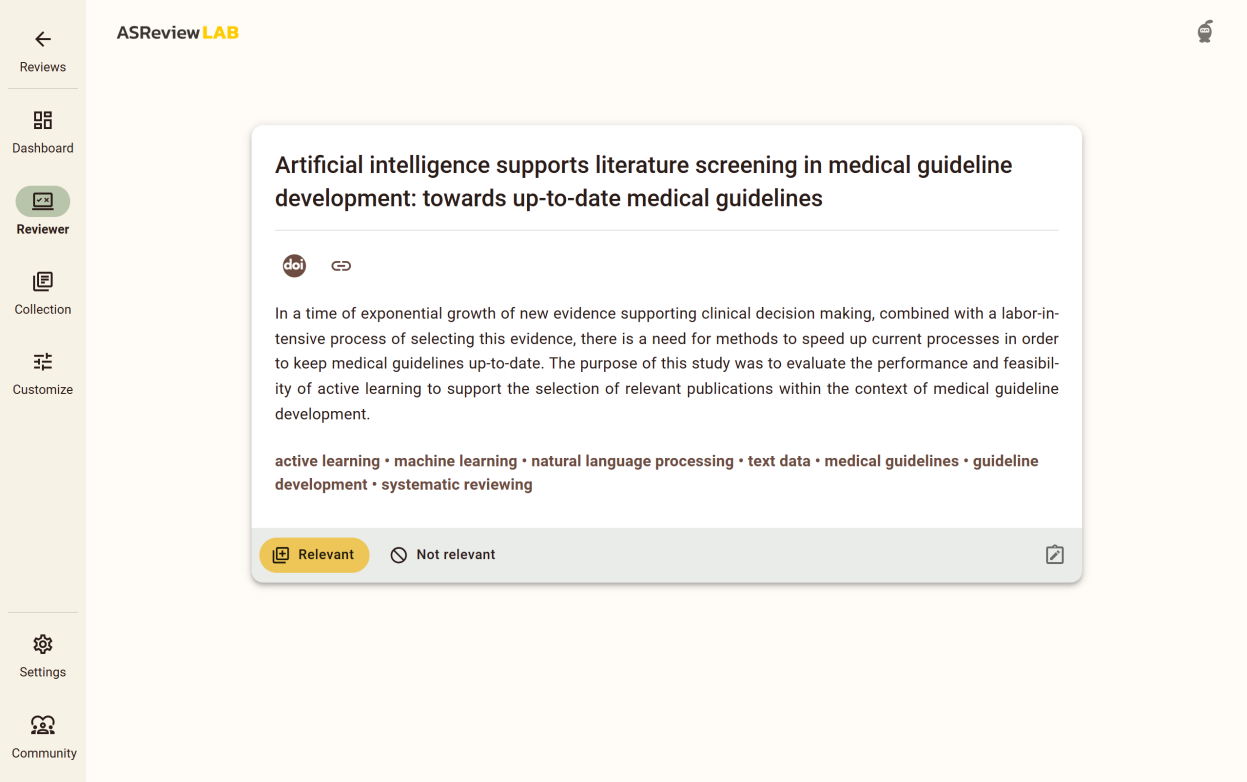


Fig. 3: The review screen of ASReview LAB with the title and abstract of a record to review. Click on the relevant or irrelevant button to label the record and continue to the next record. You can also add a note to the record or check self-defined tags.

3. Click on **Collection** in the menu on the left.
4. To change the label of a record, click on the three dots on the right of the record.
5. Click on *Change label to*.

Full Text

If Digital Object Identifiers (DOI) or URLs are available in the metadata of the records in your dataset, ASReview LAB will display the DOI and URL during screening. Most of the time, DOIs point to the full text of a publication. See *datasets* for more information on including DOI and URL values in your datasets.

Keyboard shortcuts

ASReview LAB supports the use of keyboard shortcuts during screening. The table below lists the available keyboard shortcuts.

Table 6: Keyboard Shortcuts

Action	Shortcut
Label record as relevant	**r** or **Shift + r**
Label record as irrelevant	**i** or **Shift + i**
Add note to record	**n** or **Shift + n**

Show model information

ASReview LAB allows you to view the model used to present you the record you currently see. This is especially useful if you are interested in the underlying model and how it works, or if you want to switch to a different model.

1. *Start ASReview LAB*.
2. Click on *Settings* (bottom left).
3. Click on *Show model information* to enable or disable the model view.
4. Go back to the reviewer screen if you are not there yet.

Note

The model information is only available when the record presented to you is the recommendation of a model. If you haven't trained a model yet, you will not see that the record is presented to you at random.

Warning

Switching models during screening can be hard to understand for new users or non-technical users. As the model is trained in the background after each decision, records might be presented with the "old" model for a while. Once the new model is trained, a record is presented to you based on the new model. This can be confusing for new users.

Dark mode

ASReview LAB offers the option to customize the screening appearance and functionality.

1. *Start ASReview LAB*.
2. Click on *Settings* (bottom left).

3. Click on Dark mode to enable or disable dark mode. You can also use the system preference.

> **Note**
>
> Your preference is saved in the browser.

Font size

ASReview LAB allows you to adjust the font size of the text displayed during screening.

1. *Start ASReview LAB*.
2. Click on *Settings* (bottom left).
3. Click on *Font size*.
4. Slide the slider to the desired font size.

ELAS Game

If you want a break from screening, click on the ELAS mascot on the top right in the reviewer screen *;)*. Let us know about your high score!

2.1.13 Progress and results

During screening, you might want to keep track of your progress and to obtain information for your stopping criteria. This section provides documentation on useful tools for these purposes.

Dashboard

ASReview LAB offers insightful statistics and charts to help you monitor your progress.

To open the dashboard:

1. *Start ASReview LAB*.
2. Click on *Dashboard* in the left menu.

For all of the statistics and charts, you can find information on how they work by clicking on the lamp icon in the top right corner of a statistic or chart. Feel free to ask questions about the statistics and charts on the Discussion platform https://github.com/asreview/asreview/discussions.

Stop screening

On the dashboard, you can set a stopping criteria for your screening. The stopping criterium is a threshold for the number of *not relevant* records since the last *relevant* record. If you reach this threshold, you will be asked to stop screening. You can decide to stop screening or to continue. As you will most likely see more not relevant records the longer you screen, the stopping criteria is a useful tool to help you decide when to stop screening.

The stopping treshold is not set by default. You can set the stopping criteria by clicking on the *Set threshold* button in the top right corner of the dashboard. The circle will turn dark when the threshold is reached. You can set the stopping criteria to any number you like. Choose the treshold that works best for you. Base the threshold on your own experience or simulation studies on simular topics. The ideal threshold is still actively researched.

In a popular discussion on the ASReview Discussion platform, How to stop screening?, several stopping strategies are discussed.

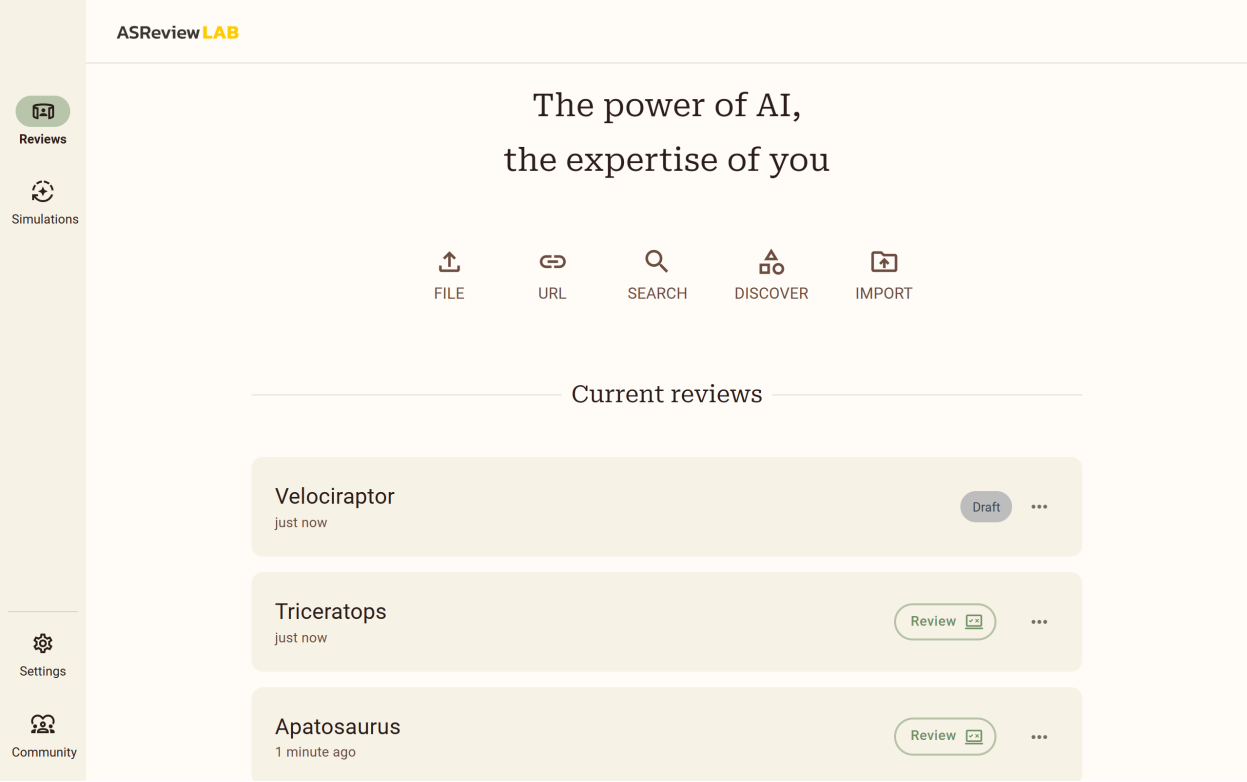


Fig. 4: The Dashboard of a review that almost hit the stopping criteria.

Tip

The wave plot shows the "waves" of irrelevant records. The larger the waves, the more irrelevant records you have seen and the more likely you are ready to stop. The wave plot displays the threshold you set.

Mark project as finished

When you decide to stop screening, you can mark the project as finished. You can undo this at any time. To mark your project as finished:

1. *Start ASReview LAB*.
2. Open the project you want to mark as finished.
3. Click on *Dashboard* if you are not there yet.
4. Click on *In review* in the top right corner of the dashboard.
5. Click on *Mark as finished*.

Continuing screening is now disabled. This can be undone by clicking again on *Finished* and resume the review.

Tip

You can find some interesting estimates of the time saved on the dashboard after marking the project as finished.

Export dataset

At any moment during the screening process, you can export your dataset. This includes the labels you provided, the data you imported, and some additional variables. You can export your dataset to a RIS, CSV, TSV, or Excel file.

To download your dataset follow these steps:

1. *Start ASReview LAB*.
2. Open a project.
3. Click on *Collection* in the menu on the left.
4. Click on the *Export* button in the top right corner of the screen.
5. Select the records you want to export.
6. Click on *Export*.

> **Note**
>
> A RIS file can only be exported if a RIS file is imported.

Variables in the exported dataset

The exported dataset contains the labels you provided during screening, the data you imported, and some additional variables. The following table lists the additional variables that are included in the exported dataset:

Table 7: Variables in the exported dataset

Variable	Description
asreview_label	Contains the labels provided by the user: `1` for relevant, `0` for not relevant, and missing if the record was not seen during screening.
asreview_time	Contains the datatime of the screening decision.
asreview_note	Contains any notes made by the user during screening.
asreview_user_name	Contains the name of the user who made the screening decision (if applicable).
asreview_user_email	Contains the email of the user who made the screening decision (if applicable).

For RIS files, the variables are stored in the N1 (Notes) field. The **asreview_label** variable is stored with the *ASReview_relevant* and *ASReview_irrelevant* tags to find them easily via search option in a reference manager.

Order of the records in the exported dataset

The file is ordered as follows:

1. All relevant records you have seen in the order they were shown during the screening process.
2. All records not seen during the screening and ordered from most to least relevant according to the last iteration of the model.
3. All non-relevant records are presented in the order these are shown during the screening process.

2.1.14 Validation reviews

Validation reviews are tailored for scenarios where it's necessary to validate existing labels or engage in a review process without being an oracle. This mode is especially beneficial for validating labels made by a first screener or an AI (i.e., Large Language Models). It is also used for educational purposes.

In a validation review, records are presented along with an indication of their existing label: relevant or irrelevant. This status is displayed via an information box above the labeling buttons in the record. If a record was labeled by another screener or an AI model, you have the opportunity to validate or challenge these labels, helping to refine the dataset by correcting any potential misclassifications. This is useful for quality evaluation.

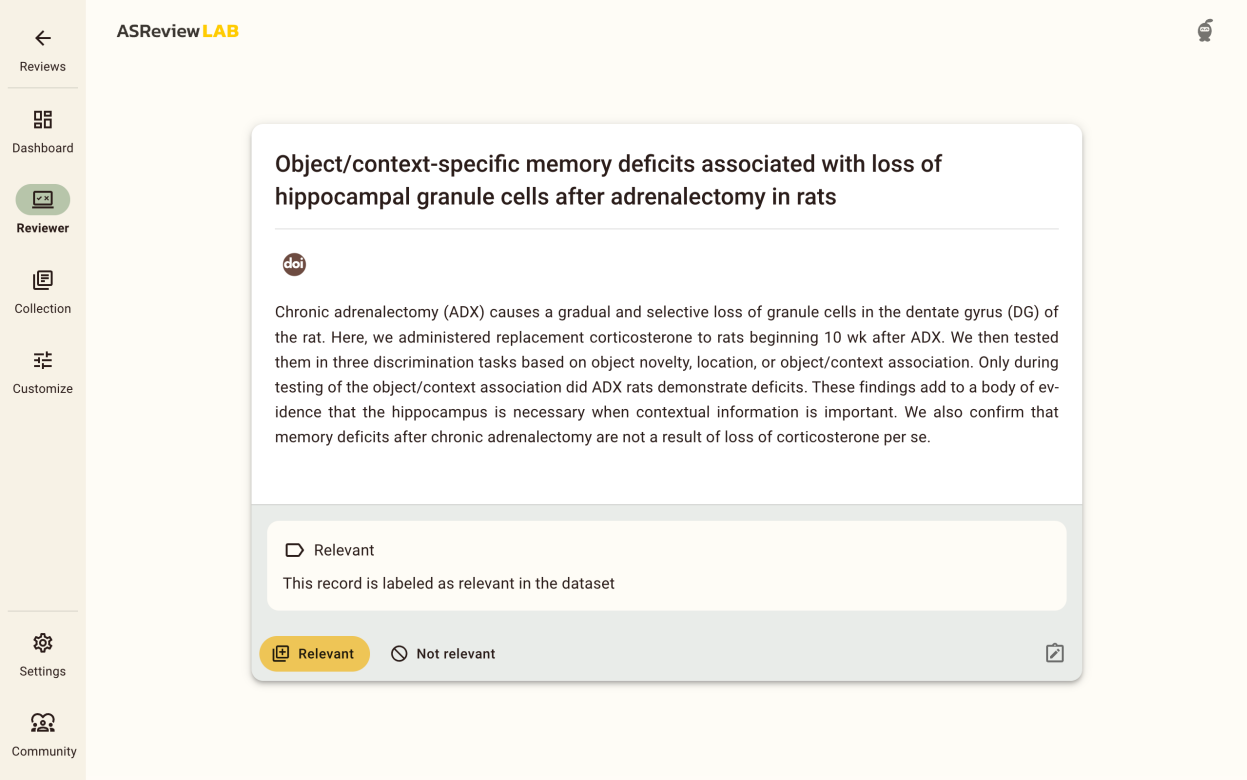


Fig. 5: The review screen of ASReview LAB with the title and abstract of a record to review. The box above the labeling buttons indicates the label of the record according to the dataset. This label can be validated or challenged.

Validation reviews for education

The validation review is useful for educational purposes. Instructors and trainers can utilize this feature to mimic the screening process without being the expert decision-maker. This setup is particularly advantageous in workshop settings, where participants can engage with the screening process using the labeled SYNERGY datasets. This hands-on experience offers valuable insights into the software's functionality and the systematic review process without the need to be a content expert.

> **Tip**
>
> For comprehensive online teaching materials and tutorials on using ASReview LAB effectively, please visit the ASReview Academy.

2.1.15 Manage projects

ASReview LAB allows you to manage your projects. You can create, open, and delete projects. You can also import and export projects. This is useful for sharing results, archiving projects, and for backup purposes. The projects dashboard shows all your projects. You can open a project by clicking on the project name.

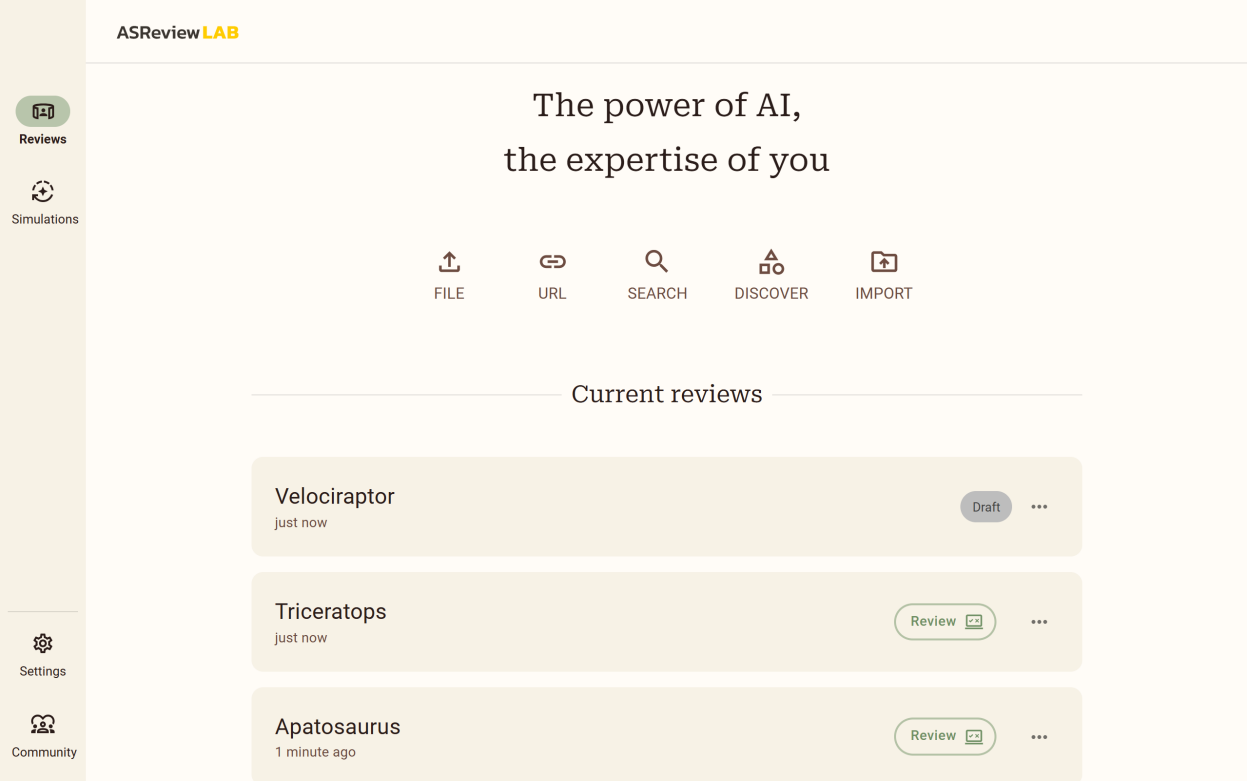


Fig. 6: On the Reviews or Simulations page, you can see all your projects. You can open a project dashboard by clicking on the project name. You can also start screening directly by clicking on the *Review* button. The three dots on the right allow you to export or delete the project.

Import Project

To import a project:

1. *Start ASReview LAB*.
2. Go to *Reviews* (http://localhost:5000/reviews) or *Simulations* (http://localhost:5000/simulations)
3. Click on the *Import* icon on the right.
4. Drag and drop your project file or select a file. The project should be an ASReview project file (extension `.asreview`).
5. Open the project from the *Reviews* or *Simulations* page.

Export Project

The ASReview project file (extension `.asreview`) can be exported from ASReview LAB. The file contains the dataset, review history, notes, and model configuration. It can be imported into ASReview LAB on a different device, which allows other users to replicate the project, or continue the systematic review.

To export your project:

1. *Start ASReview LAB*.
2. Go to *Reviews* (http://localhost:5000/reviews) or *Simulations* (http://localhost:5000/simulations)
3. Click the three dots on the right of the project you want to export.
4. Click on *Export Project*

> **Tip**
>
> Use this options also for backup purposes. The ASReview project file contains all your data, including the dataset, review history, notes, and model configuration. You can use this file to restore your project in case of data loss or corruption.

Delete Project

To permanently delete a project, including ALL files:

1. *Start ASReview LAB*.
2. Go to *Reviews* (http://localhost:5000/reviews) or *Simulations* (http://localhost:5000/simulations)
3. Click the three dots on the right of the project you want to export.
4. Click on *Export Project*
5. Type the project name in the confirmation box.
6. Click on *Delete forever*.

2.1.16 Simulate a review

Simulations in ASReview LAB provide a controlled environment to test hypotheses, refine strategies, and gain insights into model performance. By leveraging fully labeled datasets, the software mimics how a human would label records in interaction with the Active Learning model. If you're unsure which model to use for a new (unlabeled) dataset, simulations can help identify the best-performing combination of model components.

ASReview LAB offers three versatile methods to run simulations:

- *Simulate with ASReview LAB*
- *Command line interface*
- *Python API*

Simulating with ASReview LAB allows you to evaluate model performance using various metrics and estimate workload reduction achieved through active learning compared to manual screening.

Additionally, simulation mode enables benchmarking your custom models against existing ones across diverse datasets. ASReview LAB supports extending its capabilities by adding new models via a template.

Datasets for simulation

Simulations require *fully labeled datasets* (labels: `0` = irrelevant, `1` = relevant). Such a dataset can be the result of an earlier study. ASReview also provides fully labeled datasets via the SYNERGY dataset. These datasets are available via the user interface in the *Data* step of the setup and in the command line with the prefix *synergy:* (e.g., *synergy:van_de_schoot_2018*).

> **Tip**
>
> When importing your data, ensure duplicates are removed and as many abstracts as possible are retrieved (See Importance-of-abstracts blog for help). Clean data allows you to fully benefit from what *active learning* has to offer.

Simulate with ASReview LAB

To run a simulation in the ASReview LAB, go to Simulations, create a project in the same way as described in *Start a review*. Most of the steps of the setup are identical or straightforward. Make sure you import a *fully labeled dataset* or use one of the benchmark datasets.

Selecting prior knowledge is straightforward. In case you know relevant records to start with, use the search function. In case you don't, the simulation will start with random screening.

Click on Simulate to start the simulation. The simulation will run in the background. You can follow the progress on the projects overview page. Once the simulation is finished, the project can be opened to analyze or the results.

Insights into simulation results

After a simulation, the results can be exported to an ASReview project file (extension *.asreview*). This file contains a wealth of variables and logs related to the simulation. The data can be extracted from the project file via the API or with one of the available extensions. See *these examples on the Project API* for more information about accessing the project file.

One readily available extension for analyzing simulation results is ASReview Insights. This extension provides tools for plotting recall and extracting statistical results for several performance metrics, such as Loss, Work Saved over Sampling (WSS), the proportion of Relevant Records Found (RRF), Extra Relevant Records Found (ERF), and Average Time to Discover (ATD).

Install ASReview Insights directly from PyPi:

```
pip install asreview-insights
```

Detailed documentation on the extension can be found on the ASReview Insights project page.

2.1.17 Simulation via command line

ASReview LAB comes with a command line interface for simulating the performance of ASReview algorithm.

Getting started

The simulation command line tool can be accessed directly like:

```
asreview simulate MY_DATASET.csv -o MY_SIMULATION.asreview
```

This performs a simulation with the default active learning model, where `MY_DATASET.csv` is the path to the *Fully labeled data* you want to simulate. The result of the simulation is stored, after a successful simulation, at `MY_SIMULATION.asreview` where `MY_SIMULATION` is the filename you prefer and the extension is `.asreview` (ASReview project file extension).

Simulation progress

The progress of the simulation is given with two progress bars. The top one is used to count the number of relevant records found. The bottom one monitors the number of records labeled. By default (see `--n-stop`), the simulation stops once the the top progress bar reaches 100%.

```
Relevant records found: 100%|| 38/38 [00:04<00:00,  7.83it/s]
Records labeled       :   7%|                                                    | 322/
↪4544 [00:04<01:03, 66.37it/s]

Loss: 0.021
NDCG: 0.530
```

Command line arguments for simulating

The command `asreview simulate --help` provides an overview of available arguments for the simulation. Each of the sections below describe the available arguments. The example below shows how you can set the command line arguments.

```
asreview simulate MY_DATASET.csv -o MY_SIMULATION.asreview -q max_random
```

Dataset

dataset

Required. File path or URL to the dataset or one of the SYNERGY datasets.

You can also use one of the *SYNERGY dataset*. Use the following command and replace `DATASET_ID` by the dataset ID.

```
asreview simulate synergy:DATASET_ID
```

For example:

```
asreview simulate synergy:van_de_schoot_2018 -o myreview.asreview
```

Active learning

--ai AI

The AI to simulate with. Default is `elas_u4`.

-c, --classifier CLASSIFIER

The classifier for active learning. Default is Naive Bayes (`nb`).

-q, --querier QUERIER

The querier for active learning. Default is Maximum (`max`).

-b, --balancer BALANCER

Data rebalancing strategy mainly for RNN methods. Helps against imbalanced datasets with few inclusions and many exclusions. Default is `balanced`.

-e, --feature-extractor FEATURE_EXTRACTOR

Feature extraction algorithm. Some combinations of feature extractors and classifiers are not supported or feasible. Default is TF-IDF (`tfidf`).

--seed SEED

Seed for the model (classifiers, balance strategies, feature extraction techniques, and query strategies).

--prior-seed PRIOR_SEED

Seed for selecting prior records if the --prior-idx option is not used. If the option --prior-idx is used with one or more indices, this option is ignored.

--embedding EMBEDDING_FP

File path of embedding matrix. Required for LSTM models.

Prior knowledge

By default, the model initializes with no prior included or excluded records. You can set the number of priors by --n-prior-included and --n-prior-excluded. Alternatively, you can initialize your model with a specific set of starting papers using --prior-idx or --prior-record-id to select the indices or record IDs of the papers you want to start the simulation with.

The following options can be used to label prior knowledge:

--n-prior-included N_PRIOR_INCLUDED

Sample n prior included records. Only used when --prior-idx is not given. Default 0.

--n-prior-excluded N_PRIOR_EXCLUDED

Sample n prior excluded records. Only used when --prior-idx is not given. Default 0.

--prior-idx [PRIOR_IDX [PRIOR_IDX ...]]

Prior indices by row number (row numbers start at 0).

--prior-record-id [PRIOR_RECORD_ID [PRIOR_RECORD_ID ...]]

Prior indices by record ID.

Simulation setup

--n-query N_QUERY

Number of records queried each query. Default 1.

--n-stop N_STOP

The number of label actions to simulate. If not set, simulation stops after the last relevant record is found. Use -1 to simulate all label actions.

--config-file CONFIG_FILE

Configuration file for the learning cycle.

Results

--output OUTPUT, **-o** OUTPUT

Location to ASReview project file of simulation.

--verbose VERBOSE, **-v** VERBOSE

Verbosity level.

Algorithms

The command line interface provides an easy way to get an overview of all available active learning model elements (classifiers, query strategies, balance strategies, and feature extraction algorithms) and their names for command line usage in ASReview LAB. The following command lists the available models:

```
asreview algorithms
```

The command includes models added via *Developing Extensions*. See *Developing Extensions* for more information on developing new models and install them via extensions.

Use `pip install asreview-dory` to get access to all Dory models. The Dory extension contains a collection of New and Exciting MOdels.

2.1.18 Simulate with Python API

The ASReview Python API provides advanced control over the ASReview software, allowing users to customize models, implement different sampling strategies, and more. This example demonstrates how to simulate a systematic review using the ASReview API and save the results in an ASReview project file.

[1]:
```
import asreview as asr
from synergy_dataset import Dataset
```

Here, we use a dataset from the SYNERGY collection, accessed via the `synergy-dataset` package.

[2]:
```
d = Dataset("Hall_2012").to_frame()
d.head()
```

[2]:

openalex_id	doi	title	abstract	label_included
https://openalex.org/W2131536587	https://doi.org/10.1109/indcon.2010.5712716	Computer vision based offset error computation...	The use of computer vision based approach has ...	0
https://openalex.org/W2557025555	https://doi.org/10.1109/induscon.2010.5740045	Design and development of a software for fault...	This paper presents an on-line fault diagnosis...	0
https://openalex.org/W2143148279	https://doi.org/10.1109/tpwrd.2005.848672	Analytical Approach to Internal Fault Simulati...	A new method for simulating faulted transforme...	0
https://openalex.org/W2111816457	https://doi.org/10.1109/icelmach.2008.4799852	Nonlinear equivalent circuit model of a tracti...	The paper presents the development of an equiv...	0
https://openalex.org/W3142547111	https://doi.org/10.1109/ipdps.2006.1639408	Fault tolerance with real-time Java	After having drawn up a state of the art on th...	0

Next, we import the required models for the simulation.

[3]:
```
from asreview.models.balancers import Balanced
from asreview.models.classifiers import SVM
from asreview.models.feature_extractors import Tfidf
from asreview.models.queriers import Max, TopDown
from asreview.models.stoppers import IsFittable
```

We create a simulation workflow that begins with a top-down reading strategy until both a relevant and an irrelevant article are identified. Afterward, the simulation transitions to an active learning phase powered by an SVM classifier.

[4]:
```
learners = [
    asr.ActiveLearningCycle(querier=TopDown(), stopper=IsFittable()),
    asr.ActiveLearningCycle(
        querier=Max(),
        classifier=SVM(C=3),
        balancer=Balanced(ratio=5),
        feature_extractor=Tfidf(),
    ),
]

sim = asr.Simulate(
    d,
    d["label_included"],
    learners,
)
sim.review()
```

```
Relevant records found: 100%|| 104/104 [02:06<00:00,  1.22s/it]
Records labeled       :  65%|   | 5672/8793 [02:06<01:09, 44.75it/s]


Loss: 0.022
NDCG: 0.656
```

Finally, we review the simulation results to analyze the performance and outcomes of the systematic review process.

[5]:
```
sim._results
```

[5]:
```
      record_id  label classifier   querier  balancer feature_extractor  \
0             0      0       None  top_down      None              None
1             1      0       None  top_down      None              None
2             2      0       None  top_down      None              None
3             3      0       None  top_down      None              None
4             4      0       None  top_down      None              None
...         ...    ...        ...       ...       ...               ...
5667       8389      0        svm       max  balanced             tfidf
5668       1739      0        svm       max  balanced             tfidf
5669       4807      0        svm       max  balanced             tfidf
5670       5160      0        svm       max  balanced             tfidf
5671       5647      1        svm       max  balanced             tfidf

      training_set          time  note  tags user_id
0                0  1.745846e+09  None  None    None
1                1  1.745846e+09  None  None    None
```

(continues on next page)

(continued from previous page)

```
2                2  1.745846e+09  None  None    None
3                3  1.745846e+09  None  None    None
4                4  1.745846e+09  None  None    None
...            ...           ...   ...   ...     ...
5667          5667  1.745846e+09  None  None    None
5668          5668  1.745846e+09  None  None    None
5669          5669  1.745846e+09  None  None    None
5670          5670  1.745846e+09  None  None    None
5671          5671  1.745846e+09  None  None    None

[5672 rows x 11 columns]
```

CHAPTER

THREE

SERVER GUIDE

All documentation for the ASReview LAB Server

3.1 Server guide

3.1.1 ASReview LAB Server

ASReview LAB Server is a self-hosted, secure version of ASReview LAB that enables users to screen together in the same project. It is designed with organizations and multi-user situations in mind but can also be used by individuals who want to use ASReview LAB without installing it on their own computer.

The self-hosted ASReview LAB Server is especially useful for:

- **Organizations**: Organizations can use ASReview LAB Server to give access to ASReview LAB to their employees. This can be useful for organizations that make extensive use of ASReview LAB or other applications of screening prioritization.
- **Multi-user screening**: ASReview LAB Server is designed for multi-user situations. This means that multiple users can screen the same set of records at the same time. All users interact with the same AI model and the same set of records.
- **Data privacy**: ASReview LAB Server is self-hosted, which means that organizations can deploy it on their own hardware or in the cloud. This ensures that data is kept private and secure, and that organizations can comply with institutional policies.
- **Heavy models**: ASReview LAB Server is designed to separate the AI engine (Task server) from the front end and back end. This implies that the AI engine can be run on a separate server, which can be more powerful than the front-end and back-end servers. This is especially useful for organizations that have heavy models that require a lot of processing power or even GPUs.
- **Mobile screening**: ASReview LAB Server enables users to screen on mobile devices. This can lead to increased productivity and efficiency, as users can screen records on the go.

It facilitates users who want to use ASReview LAB without the need to install it on their own computer. The web application can be accessed from any device with a web browser and can be used on desktops, laptops, tablets, and mobile devices. ASReview LAB Server enables users to create an account or connect via their GitHub, ORCID, or Google accounts.

To streamline self-hosting and enterprise-level deployments, the ASReview Server Stack provides a production-ready Docker Compose setup. Follow the installation instructions in the *Installation* section to get started. See the *Server configuration* details for more information on how to configure your ASReview LAB on your server.

3.1.2 Installation

> **Note**
>
> Migrating from ASReview version 1 to version 2? Follow the *Migration* guide.

ASReview LAB Server is a web application that allows you to run ASReview LAB on your own server. It is designed to be easy to install and configure, and provides the same functionality as the ASReview LAB application with extra features like user authentication and crowd screening.

The software can be installed in a similar way as ASReview LAB. However, for production environments, we recommend using ASReview Server Stack. This is a Docker Compose setup that packages the main components of ASReview—such as the AI engine (task server), the React front end, reverse proxy server, and a database layer—into separate, containerized services.

Prerequisites

- Docker and Docker Compose installed on your server. See the Docker installation instructions.
- A server with sufficient resources to run the ASReview LAB application and database. We recommend at least 8 GB of RAM and 4 CPU cores for a production environment.
- A domain name or IP address for your server. This is required for the reverse proxy configuration.

Optional

- Git installed on your server so that you can clone the ASReview Server Stack repository from GitHub. If you don't have Git installed, you can download the repository as a ZIP file and extract it to your server.
- A valid SSL certificate for your domain name. This is required for secure communication between the server and the client. You can use Let's Encrypt to obtain a free SSL certificate.
- A PostgreSQL database. You can use the PostgreSQL database provided by the ASReview Server Stack, or you can use your own PostgreSQL database. If you use your own database, make sure to configure the connection settings in the *Server configuration* file.

Deploy to production

To deploy ASReview LAB Server to production, follow these steps:

1. Clone the ASReview Server Stack repository from GitHub:

   ```
   git clone git@github.com:asreview/asreview-server-stack.git
   ```

2. Change to the ASReview Server Stack directory:

   ```
   cd asreview-server-stack
   ```

3. Configure the environment file if needed. In the *.env* file, you find parameters of the Docker Compose file only:

   ```
   nano .env
   ```

4. Edit the *asreview_config.toml* file to configure the ASReview LAB application. You can use the example configuration file provided in the repository as a starting point.

5. Change the secrets in the *asreview_config.toml* file. The secrets are used to encrypt sensitive data, such as password salt and the session key. You can use the *openssl* command to generate a random secret key:

```
openssl rand -hex 32
```

SSL Configuration

For secure communication, it is recommended to use SSL (Secure Sockets Layer) for your ASReview LAB Server. This ensures that all data transmitted between the server and the client is encrypted and secure. SSL is especially important if you are handling sensitive data, such as user credentials or personal information.

> **Note**
>
> This documentation is not fully ready yet. For an example of SSL configuration, please refer to the ASReview Server Stack repository.

PostgreSQL database

You can replace the SQLite database used for authentication with a PostgreSQL database. This requires an extra step during installation and an extra step in the configuration file:

1. Install the psycopg2 package. At the time of this writing, two versions of this package exist: `psycopg2` and `psycopg2-binary`. According to the documentation, the binary version works on most operating systems. You can skip this step if you use the Docker image provided by the ASReview Server Stack.
2. Then add the `SQLALCHEMY_DATABASE_URI` key to the config file:

```
SQLALCHEMY_DATABASE_URI = "postgresql+psycopg2://username:password@host:port/database_
↪name"
```

3.1.3 Authentication

OAuth

ASReview LAB Server provides an easy way to log in with your GitHub, ORCID, or Google account. This allows users to authenticate themselves without the need to create a separate account. More OAuth providers are supported but not documented or extensively tested.

See the *Server configuration* details for more information on how to configure your ASReview on your server to enable OAuth.

Basic authentication

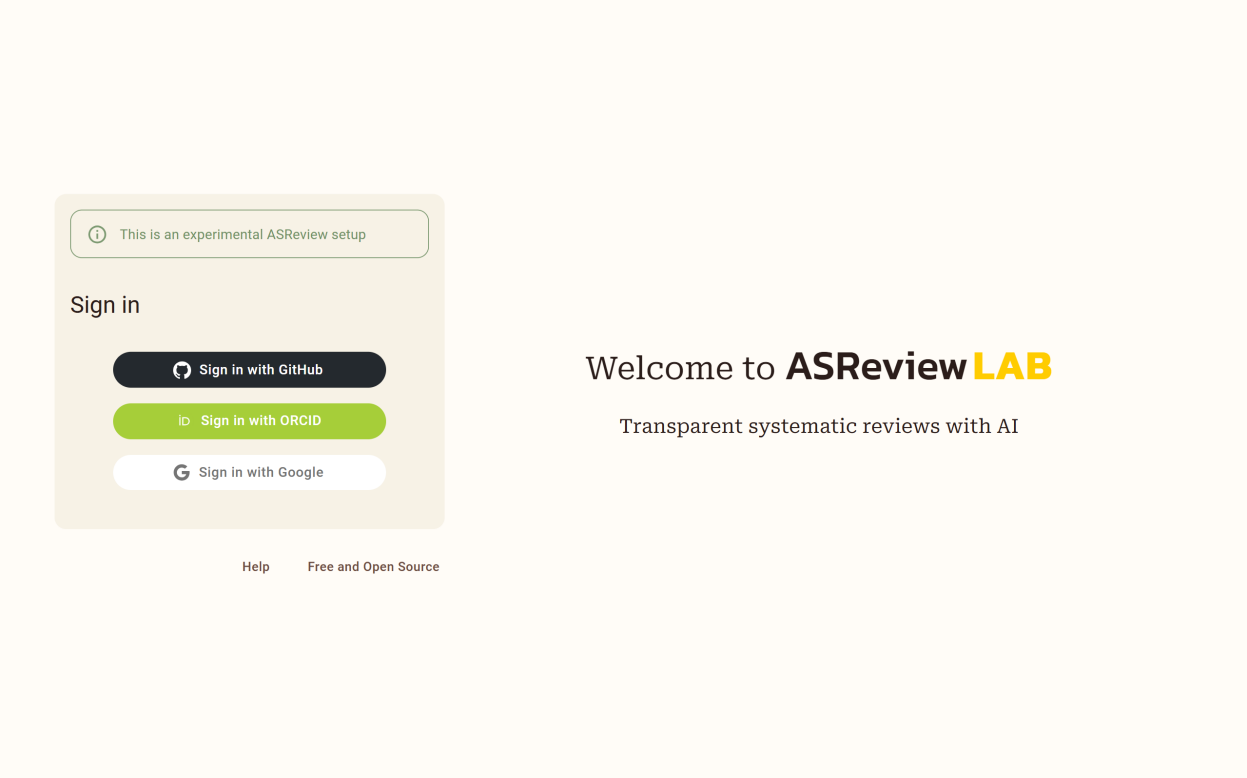
ASReview LAB Server provides an easy way to create an account with your email. This is eanbled by default when authentication is enabled. You can let users make an account themselves or create accounts for them with *Auth-Tool*.

Authentication with remote user

ASReview LAB Server provides the option to outsource authentication completely to a web server or middleware application that is placed in front of ASReview. This is a common pattern in web hosting: we use a web server like Nginx to implement authentication (for example, using its built-in modules for things like LDAP authentication), and let it *reverse proxy* to the web application we want to serve (ASReview). The web server then only needs to pass on the information about the user (such as username, full name, email address) to ASReview in the HTTP headers.

Although this is a powerful feature that allows one to leverage a myriad of authentication options, **it should be used with caution**. If the web server is not properly configured, ASReview will be improperly secured.

Note that if the user specified by the remote user header does not yet exist, it will be created *regardless of the value of the ALLOW_ACCOUNT_CREATION option*.



Use the *REMOTE_USER* option to enable this form of authentication handling. This is a *dict* in which you can configure which headers ASReview will attempt to read user information from.

The default is simply:

```
[REMOTE_USER]
USER_IDENTIFIER_HEADER = 'REMOTE_USER' # The primary header identifying the user. Can be␣
↪use a username or email.
```

However, you can set some additional options. **It is imperative that any of the headers you configure here are set by your middleware.** Otherwise, any user will be able to pass arbitrary values.

Example with optional values:

```
[REMOTE_USER]
USER_EMAIL_HEADER = 'REMOTE_USER_EMAIL' # Header containing user's email. If not set,␣
↪will default to 1. USER_IDENTIFIER_HEADER (if it is an email) 2. <username>@<DEFAULT_
↪EMAIL_DOMAIN>.
USER_NAME_HEADER = 'REMOTE_USER_FULLNAME' # Header containing user's full name. If not␣
↪set, user's name will be set to the username inferred from the identifier.
USER_AFFILIATION_HEADER = 'REMOTE_USER_AFFILIATION' # Header containing user's␣
↪affiliation.
DEFAULT_AFFILIATION = '' # Default affiliation if no header is set.
DEFAULT_EMAIL_DOMAIN = 'localhost' # If no email header is set and USER_IDENTIFIER_
↪HEADER is not an email, use this as a default domain. The user's email will be set to:
↪<username>@<default_email_domain>
REMOTE_AUTH_SECRET = 'secret' # If set, authentication will fail unless the request␣
↪contains a 'REMOTE_AUTH_SECRET' header with the same value as this. This adds some␣
```

(continues on next page)

(continued from previous page)

```
↪additional security, so that users with direct access to the webapp (on localhost,␣
↪say) cannot easily authenticate without this secret.
```

3.1.4 Auth-Tool

The *auth-tool* is a command-line utility provided by ASReview to manage user authentication and project linking in an authenticated environment. It allows administrators to create databases, manage user accounts, list projects, and migrate projects from unauthenticated to authenticated modes.

Features of Auth-Tool

- Create and manage authentication databases.
- Add user accounts interactively or in bulk.
- List users and projects in the database.
- Migrate projects to authenticated environments.

Creating the Authentication Database

To create a database for authentication, use the *create-db* command:

```
asreview auth-tool create-db --db-uri=sqlite:////path/example.sqlite
```

If the *–db-uri* option is not provided, the tool will use the default database URI configured in the ASReview folder.

Adding User Accounts

You can add user accounts interactively or in bulk using a JSON string.

Interactive Mode:

Run the following command:

```
asreview auth-tool add-users --db-uri=sqlite:////path/example.sqlite
```

The tool will prompt you to add user details such as email, name, affiliation (optional), and password.

Bulk Mode:

Use the *–json* option to add multiple users at once:

```
asreview auth-tool add-users \
        --db-uri=sqlite:////path/example.sqlite \ -j "[{\"email\":
        \"name@email.org\", \"name\": \"Name of User\", \"affiliation\":
        \"Some Place\", \"password\": \"1234@ABcd\"}]"
```

The JSON string should contain a list of dictionaries with the following keys: *email*, *name*, *affiliation*, and *password*.

Listing Projects

To list all projects in the database, use the *list-projects* command:

```
asreview auth-tool list-projects
```

To get the output in JSON format, use the *–json* flag:

```
asreview auth-tool list-projects --json
```

The JSON output can be used to bulk insert or link projects to users.

Listing Users

To list all users in the database, use the *list-users* command:

```
asreview auth-tool list-users
```

Migrating Projects to Authenticated Mode

If you are switching from unauthenticated to authenticated mode, you need to migrate existing projects.

Interactive Migration:

Run the following command:

```
asreview auth-tool link-projects --db-uri=sqlite:////path/example.sqlite
```

The tool will prompt you to assign a user ID to each project.

Bulk Migration:

Use the JSON output from the *list-projects* command, add user IDs to each project, and run:

```
asreview auth-tool link-projects \
        --db-uri=sqlite:////path/example.sqlite \ --json "[{\"folder\":
        \"project-id\", \"version\": \"1.3\", \"project_id\":
        \"project-id\", \"name\": \"project 1\", \"authors\":
        \"Authors\", \"created\": \"2023-04-12 21:23:28.625859\",
        \"owner_id\": 15}]"
```

3.1.5 Server configuration

ASReview LAB offers a number of options to run the application on a server. It is possible to run the application on a server with or without authentication. The latter is the default option. This page describes how to configure the ASReview LAB application to run on a server with authentication enabled. With authentication enabled, users can run their projects in their own separate workspaces. Authentication requires the storage of user accounts and links these accounts to projects. Currently, we are using a SQLite database (asreview.development.sqlite or asreview.production.sqlite) in the ASReview projects folder to store that information.

Getting started

To configure the authentication in more detail, we need to create a TOML file that contains all relevant authentication parameters. The parameters in that TOML file will override parameters that were passed in the CLI. Below is an example of a TOML file (extension *.toml*) that enables authentication and OAuth with GitHub, ORCID, and Google. It also enables email verification and allows users to create their own accounts. The email server is configured to confirm new accounts and to allow users to retrieve a new password if they forget it. The TOML file also contains the necessary parameters to run the application in a secure way (HTTPS).

```
DISABLE_LOGIN = false LOGIN_DURATION = 31 SECRET_KEY = "<secret key>"
SECURITY_PASSWORD_SALT = "<salt>" SESSION_COOKIE_SECURE = true
REMEMBER_COOKIE_SECURE = true SESSION_COOKIE_SAMESITE = "Lax"
SQLALCHEMY_TRACK_MODIFICATIONS = true ALLOW_ACCOUNT_CREATION = true
```

(continues on next page)

(continued from previous page)

```
EMAIL_VERIFICATION = false

MAIL_SERVER = "<smtp-server>" MAIL_PORT = 465 MAIL_USERNAME =
"<smtp-server-username>" MAIL_PASSWORD = "<smtp-server-password>"
MAIL_USE_TLS = false MAIL_USE_SSL = true MAIL_DEFAULT_SENDER = "<preferred
reply email address>"

[OAUTH]
        [OAUTH.GitHub] AUTHORIZATION_URL =
        "https://github.com/login/oauth/authorize" TOKEN_URL =
        "https://github.com/login/oauth/access_token" CLIENT_ID = "<GitHub
        client ID>" CLIENT_SECRET = "<GitHub client secret>" SCOPE = ""

        [OAUTH.Orcid] AUTHORIZATION_URL =
        "https://sandbox.orcid.org/oauth/authorize" TOKEN_URL =
        "https://sandbox.orcid.org/oauth/token" CLIENT_ID = "<Orcid client
        ID>" CLIENT_SECRET = "<Orcid client secret>"

        [OAUTH.Google] AUTHORIZATION_URL =
        "https://accounts.google.com/o/oauth2/auth" TOKEN_URL =
        "https://oauth2.googleapis.com/token" CLIENT_ID = "<Google client
        ID>" CLIENT_SECRET = "<Google client secret>" SCOPE = "profile
        email"
```

Not that the SCOPE parameter is missing for Orcid: to retrieve user data from Orcid, multiple calls have to be made with different scopes. Therefor the scopes are hard-coded.

Store the TOML file on the server and start the ASReview application from the CLI with the `--config-path` parameter:

```
asreview lab --config-path=<path-to-TOML-config-file>
```

A number of the keys in the TOML file are standard Flask parameters. The keys that are specific for authenticating ASReview are summarized below:

- DISABLE_LOGIN: if set to `false` the application will start with authentication. If the SQLite database does not exist, one will be created during startup.
- LOGIN_DURATION: number of days that a user should remain logged in. Default: 31.
- SECRET_KEY: the secret key is a string that is used to encrypt cookies and is mandatory if authentication is required.
- SECURITY_PASSWORD_SALT: another string used to hash passwords, also mandatory if authentication is required.
- SESSION_COOKIE_SAMESITE: Restrict how cookies are sent with requests from external sites. In the example the value is set to "Lax" which is the recommended option. If backend and frontend are served on different domains set to the string "None".
- ALLOW_ACCOUNT_CREATION: enables account creation by users, either by front- or backend.
- EMAIL_VERIFICATION: used in conjunction with ALLOW_ACCOUNT_CREATION. If set to `true` the system sends a verification email after account creation. Only relevant if the account is __not__ created by OAuth. This parameter can be omitted if you don't want verification.
- MAIL_<PAR>: configuration parameters to setup the SMTP email server that is used for email verification.

It also allows users to retrieve a new password after forgetting it. Don't forget to enter the reply address (MAIL_DEFAULT_SENDER) of your system emails. Remove these parameters if system emails for verification and password retrieval are unwanted.

- OAUTH: an authenticated ASReview application may integrate with the OAuth functionality of Github, Orcid and Google. Provide the necessary OAuth login credentails (for Github, Orcid en Google). Please note that the *AUTHORIZATION_URL* and *TOKEN_URL* of the Orcid entry are sandbox-urls, and thus not to be used in production. Omit this parameter if OAuth is unwanted.

The `SQLALCHEMY_DATABASE_URI` key is not included in the TOML file. This key is used to configure the database connection. The default value is `sqlite:///asreview.production.sqlite`. This means that the application will use the SQLite database in the ASReview projects folder. If you would like to use a different database, you can add the `SQLALCHEMY_DATABASE_URI` key to the TOML file.

Full configuration

All ASReview LAB settings are prefixed with *ASREVIEW_LAB_*. They include all settings from Flask configuration.

ASReview LAB settings

ASREVIEW_LAB_CONFIG_PATH

Path to ASReview LAB config TOML file with ASReview LAB configuration.

ASREVIEW_LAB_SECRET_KEY

Secret key for ASReview LAB.

Login configuration

ASREVIEW_LAB_AUTHENTICATION

If false, login is disabled and no password is required to use ASReview LAB.

ASREVIEW_LAB_SQLALCHEMY_DATABASE_URI

Database URI for ASReview LAB.

Account creation configuration

ASREVIEW_LAB_ALLOW_ACCOUNT_CREATION

If true, account creation is enabled.

ASREVIEW_LAB_SECURITY_PASSWORD_SALT

Salt for password hashing.

ASREVIEW_LAB_RE_CAPTCHA_V3

If true, reCAPTCHA v3 is enabled for account creation.

OAuth configuration

ASREVIEW_LAB_OAUTH

OAuth configuration for ASReview LAB. It is a dictionary with the following keys: *GitHub*, *Orcid*, and *Google*. Each of these keys is a dictionary with the following keys: *AUTHORIZATION_URL*, *TOKEN_URL*, *CLIENT_ID*, *CLIENT_SECRET*, and *SCOPE*.

Remote user configuration

ASREVIEW_LAB_REMOTE_USER

Remote user configuration for ASReview LAB. It is a dictionary with the following keys: *USER_IDENTIFIER_HEADER*, *USER_NAME_HEADER*, *USER_EMAIL_HEADER*, *USER_AFFILIATION_HEADER*, *DEFAULT_EMAIL*, *DEFAULT_AFFILIATION*, *REMOTE_AUTH_SECRET*.

Cookie configuration

ASREVIEW_LAB_REMEMBER_COOKIE_*

Login-related cookie settings. Refer to Flask-Login documentation for details.

Mail configuration

ASREVIEW_LAB_EMAIL_VERIFICATION

If true, email verification is required for new accounts.

ASREVIEW_LAB_MAIL_*

Mail-related configuration. Refer to Flask-Mail documentation for details.

CORS configuration

ASREVIEW_LAB_CORS_*

CORS configuration. Refer to Flask-CORS documentation for details, except *ASREVIEW_LAB_CORS_SUPPORTS_CREDENTIALS*, which is always true. *ASREVIEW_LAB_CORS_ORIGINS* is used to link backend to frontend on different host and port.

ASREVIEW_LAB_POST_LOGOUT_URL

URL to which to redirec the user after they have signed out. Default is the standard login page */signin*. It can be useful to set this to a custom path when using external users, with remote user auth or OAuth. In that case, you might want to redirect the user to an organizational page after logout. Note that this should be set to a valid URI, so */custom_logout_page* and not *custom_logout_page*.

3.1.6 Migration

This guide explains how to use the *asreview migrate* command to ensure your database or projects are compatible with the latest stable version of ASReview, specifically for migrating from version 1 (v1) to version 2 (v2).

Migration Steps Version 1 to Version 2

Follow these two steps to complete the migration process:

Step 1: Backing Up Before Migration

Before starting the migration, it is recommended to back up both the database and the projects folder. This precaution prevents data loss and provides a recovery point in case of errors.

Backing up the default SQLite3 database can be done by copying the database file to a different location:

```
cp /home/username/.asreview/asreview.production.sqlite /home/username/new_folder
```

The project files are located in the dedicated ASReview folder. To create a copy of all files recursively:

```
cp -r /home/username/.asreview /home/username/new_folder
```

For server administrators using the ASReview Lab Docker stack, backing up the projects and database requires executing commands within the running Docker containers.

To start, identify the active containers:

```
docker ps --format "{{.Names}}"
```

Example output:

```
asreview-server-stack-server-1
asreview-server-stack-asreview-1
asreview-server-stack-database-1
```

Back up the PostgreSQL database using the following command:

```
docker exec -u postgres asreview-server-stack-database-1 pg_dump asreview_db > /home/
↪username/new_folder/asreview.backup.sql
```

To back up all project files, copy the contents of the `project_folder` volume from the container to the host system:

```
docker cp asreview-server-stack-asreview-1:/project_folder /home/username/new_folder/
↪asreview-projects-backup
```

Step 2: Migrate the Database

The first step is to migrate the database, which includes user profiles and other related data.

Run the following command to migrate the database:

```
asreview migrate --db
```

If you are using a custom database URI, specify the URI of the database.

```
asreview migrate --db --db-uri sqlite:///custom_database.sqlite
```

By default, the value is taken from the environment variable `ASREVIEW_LAB_SQLALCHEMY_DATABASE_URI`. If not set, the default is `asreview.production.sqlite` in the ASReview folder.

Server administrators using the ASReview Lab Docker stack should run the migration command inside the Docker container. For example:

```
docker exec -it asreview-server-stack-asreview-1 asreview migrate --db
```

Step 3: Migrate the Projects

After migrating the database, proceed to migrate the projects format.

Run the following command:

```
asreview migrate --projects
```

This ensures that all your projects are updated to the latest compatible format.

To perform the project migration within the ASReview Lab Docker stack, run the following command inside the container:

```
docker exec -it asreview-server-stack-asreview-1 asreview migrate --projects
```

CHAPTER

FOUR

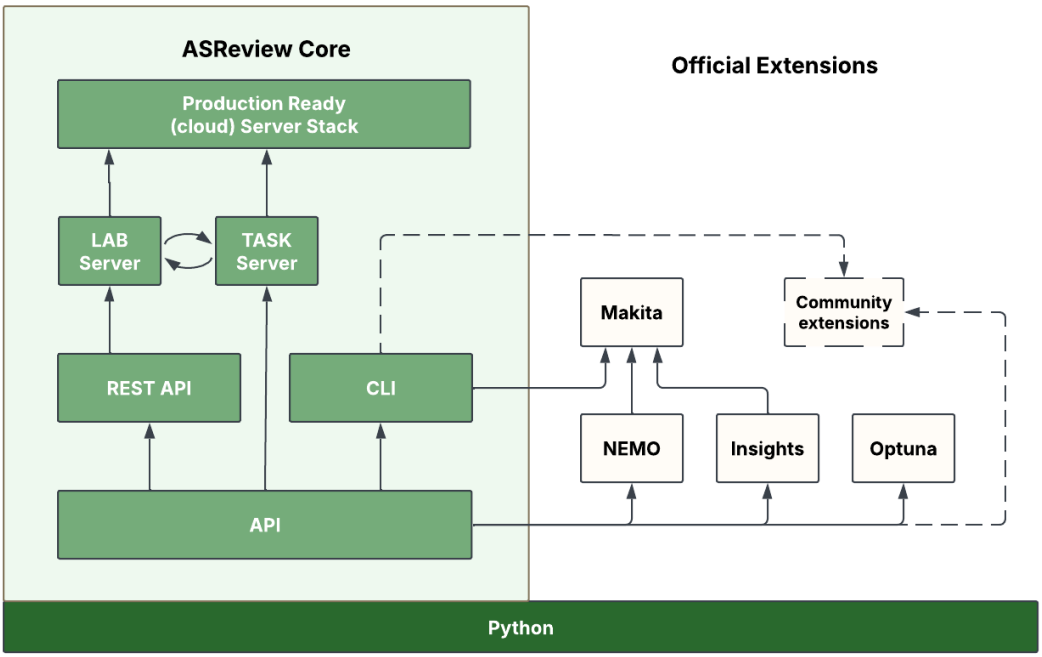
TECHNICAL GUIDE

All technical documentation and reources

4.1 Technical guide

4.1.1 Architecture

ASReview provides developers and researchers (with a more technical skill set) with several interfaces to interact directly with the underlying ASReview machinery. This enables the development of custom algorithms, the integration of ASReview into larger projects, and the creation of custom workflows. The following overview shows the available interfaces for interacting with the ASReview software:



Core Interfaces

API

ASReview LAB ships with a documented Application Programming Interface (API) that provides models, data, and project management functionality. The rich set of functions, classes, and modules allows researchers and developers to develop custom workflows, integrate new algorithms, or embed ASReview functionality in larger projects. It is also the foundation for the higher-level interfaces of ASReview LAB. For detailed documentation, refer to the *API reference*.

REST API

A stateless REST API written in Flask provides an interface for web applications built on ASReview. While integral to ASReview LAB, this REST API is still under active development and is not yet fully documented.

CLI

The *Command Line Interface (CLI)* of ASReview provides an interface for users of computer terminals to start ASReview LAB, run simulations, list algorithms, and more. The command *asreview lab* will start the user-friendly web app interface. It can also be extended with subcommands provided by both official and community-built extensions (see *Developing Extensions*).

Servers

Task Server

ASReview LAB v2 introduces a task server for handling asynchronous tasks like training agents and running simulations. The task server comes with a network socket interface and makes use of the Transmission Control Protocol (TCP) for communication. Via environment variables or *Server configuration*, you can set the port, the host, and the number of workers. The variables are:

ASREVIEW_LAB_TASK_MANAGER_PORT

The port for the task server.

ASREVIEW_LAB_TASK_MANAGER_HOST

The host for the task server.

ASREVIEW_LAB_TASK_MANAGER_WORKERS

The number of workers for the task server.

ASReview LAB Server

The LAB server runs on Flask and serves the RESTful API and the web application. It is responsible for handling incoming requests and serving the ASReview LAB web application. The LAB server is started with the command *asreview lab*.

Extensions

ASReview LAB is designed to be extensible, allowing users to add new models, subcommands, and datasets. The extension system is built on top of the core ASReview API and Python's entry point system. More information on developing extensions can be found under *Developing Extensions*.

4.1.2 Command Line

ASReview provides a powerful command-line interface for running tasks *Start ASReview LAB* and *Simulation via command line*. Also, *Developing Extensions* often make use of the command-line interface by extending it with subcommands.

The structure of the command line is given by:

```
asreview [-h] [-V] [subcommand]
```

A list of available and installed subcommands is given by `asreview -h`. Each subcommand is listed with its name, the package it comes from, and the version of the package. For example, the default subcommand `lab` (to start ASReview LAB) is listed as `lab [asreview-2.0]`. A subcommand installed via an extension, e.g., `plot`, is listed as `plot [asreview-insights-1.3]`, where `asreview-insights` is the name of the extension that installed this subcommand, and 1.3 is the version of this package.

4.1.3 Developing Extensions

ASReview extensions enable you to integrate your programs with the ASReview framework seamlessly by using the Python API. These extensions fall into three different categories and interact with the API in different ways.

1. *Model extensions*
2. *Subcommand extensions*
3. *Dataset extensions*

The extensibility of the framework is provided by the entry points of setuptools. You will need to create a package and install it (for example, with pip).

Did you develop a useful extension to ASReview and want to list it on the Discussion platform? Leave a message there, and we will add it to the list of extensions.

For more information on the ASReview API for creating an extension, a technical reference for development is found under the *asreview*. This technical reference contains functions for use in your extension and an overview of all classes to extend.

Model Extensions

Model extensions extend the ASReview software with new classifiers, query strategies, balance strategies, or feature extraction techniques. Model extensions are Python packages that can be installed in the ASReview environment. Model extensions typically inherit from the `sklearn.base.BaseEstimator` class in Scikit-learn or have a similar interface. The model extensions can be used in the ASReview LAB and via the Command Line Interface (CLI).

The easiest way to extend ASReview with a model is by using the . Create a copy of the template and add the new algorithm to a new model file. It is advised to use the following structure of the package:

```
├── README.md
├── asreviewcontrib
│   └── models
│       ├── classifiers.py
│       ├── feature_extractors.py
│       ├── balancers.py
│       └── queriers.py
└── tests
```

The next step is to add metadata to the `pyproject.toml` file. Edit the `name` of the package and point the `entry-points` to the models.

```
[project]

name = "asreviewcontrib-yourmodel"

[project.entry-points."asreview.models.classifiers"]
```

(continues on next page)

(continued from previous page)

```
example = "asreviewcontrib.models.classifiers.example_model:ExampleClassifier"

[project.entry-points."asreview.models.feature_extractors"]

# define feature_extraction algorithms

[project.entry-points."asreview.models.balancers"]

# define balance_strategy algorithms

[project.entry-points."asreview.models.queriers"]

# define query_strategy algorithms
```

This code registers the model with name `example`.

Subcommand Extensions

Subcommand extensions are programs that create a new entry point for ASReview. From this entry point the Python API can be used in many ways (like `plot` or `simulate`).

Extensions in ASReview are Python packages and can extend the subcommands of asreview (see `asreview -h`). An example of a subcommand extension is ASReview Insights.

The easiest way to create a new subcommand is by defining a function or class with *execute* method that can be used as a new entry point for ASReview.

```
class ExampleEntryPoint:

    def execute(self, argv):
        pass  # Implement your functionality here.
```

The class method `execute` accepts a positional argument (`argv` in this example). The argument `argv` are the command line arguments for your subcommand.

It is advised to place the newly defined entry point in the following package structure: `asreviewcontrib.{extension_name}.{your_modules}`. For example:

```
├── README.md
├── asreviewcontrib
│   └── example
│       ├── __init__.py
│       ├── entrypoint.py
│       └── example_utils.py
├── pyproject.toml
└── tests
```

Create a `pyproject.toml` in the root of the package, and define the entry points under `[project.entry-points."asreview.entry_points"]`, for example:

```
[project] name = "asreviewcontrib-example"

# ...other metadata...
```

(continues on next page)

(continued from previous page)

```
[project.entry-points."asreview.entry_points"]

example = "asreviewcontrib.example.entrypoint:ExampleEntryPoint"
```

After installing this package, ASReview is extended with the `asreview example` subcommand. See `asreview -h` for this option.

Dataset Extensions

An extension of the *asreview.datasets.BaseDataSet* class.

Dataset extensions integrate new datasets for use in ASReview. Adding datasets via extension provides quick access to the dataset via Command Line Interface or in ASReview LAB.

It is advised to place the new dataset `your_dataset` in the following package structure:

```
├── README.md
├── asreviewcontrib
│   └── dataset_name
│       ├── __init__.py
│       └── your_dataset.py
├── data
│   └── your_dataset.csv
├── pyproject.toml
└── tests
```

For minimal functionality, `your_dataset.py` should extend *asreview.datasets.BaseDataSet* and *asreview.datasets.BaseDataGroup*.

A working template to clone and use can be found at Template for extending ASReview with a new dataset.

Further functionality can be extensions of any other class in *asreview.datasets*.

4.1.4 Access data from ASReview file

The API is still under development and can change at any time without warning.

Data generated using ASReview LAB is stored in an ASReview project file. Via the ASReview Python API, there are two ways to access the data in the ASReview (extension `.asreview`) file: Via the asreview.Project API and the asreview.SQLiteState API. The project API is for retrieving general project settings, the imported dataset, the feature matrix, etc. The state API retrieves data related directly to the reviewing process, such as the labels, the time of labeling, and the classifier used.

The ASReview Python API can be used for project files obtained reviews and simulations.

Example Data

To illustrate the ASReview Python API, the benchmark dataset `van_de_Schoot_2017` is used. The project file `example.asreview` can be obtained by running the following command:

[ ]:
```
%%bash

asreview simulate synergy:van_de_Schoot_2018 -o example.asreview
```

Python imports

Import the `asreview` module as `asr`.

[1]:
```
from pathlib import Path

import asreview as asr
```

Project API

The ASReview project file is a zipped folder with the extension `.asreview`. This makes inspection of it's contents straightforward. The `asreview` Python package offers an API to open a project file directly as an asreview.Project.

For this example, we will create a temporary folder to unpack the project to:

[2]:
```
from tempfile import TemporaryDirectory

tmpdir = TemporaryDirectory()
```

Open the project with the `load` classmethod of asreview.Project.

[3]:
```
project = asr.Project.load("example.asreview", tmpdir.name)
```

The following files can be found in the folder

[4]:
```
tmpdir_path = Path(tmpdir.name)
for path in tmpdir_path.rglob("*"):
    print(path.relative_to(tmpdir_path))
```

```
example
example/project.json
example/data_store.db
example/feature_matrices
example/data
example/reviews
example/data/van_de_Schoot_2018.csv
example/reviews/3b7b6c2b3d6f487ba19ffcc4ac8adef5
example/reviews/3b7b6c2b3d6f487ba19ffcc4ac8adef5/results.db
```

To inspect the project details in `project.json`, use the following code:

[5]:
```
project.config
```

[5]:
```
{'version': '2.0b6.dev20+gf283ffd9.d20250505',
 'id': 'example',
 'mode': 'simulate',
 'name': 'van_de_Schoot_2018',
 'created_at_unix': 1746457760,
 'reviews': [{'id': '3b7b6c2b3d6f487ba19ffcc4ac8adef5', 'status': 'finished'}],
 'feature_matrices': [],
 'tags': None,
```

(continues on next page)

(continued from previous page)

```
 'datasets': [{'id': 'van_de_Schoot_2018.csv',
   'name': 'van_de_Schoot_2018.csv'}]}
```

The imported dataset is located at `data/{dataset_filename}`, and can be inspected using the following code:

[6]:
```
datastore_fp = Path(tmpdir.name) / "example" / "data_store.db"

dataset = asr.DataStore(datastore_fp)
print(f"The dataset contains {len(dataset)} records.")
```

```
The dataset contains 4544 records.
```

[7]:
```
dataset.get_df().head()
```

[7]:

	dataset_row	dataset_id	duplicate_of	title	abstract	authors	keywords	year	doi	url	included	record_id
0	0	van_de_Schoot_2018.csv	None	Annual Research Review: Resilience and mental ...	Researchers focused on mental health of confli...	[]	[]	None	https://doi.org/10.1111/jcpp.12053	None	0	0
1	1	van_de_Schoot_2018.csv	None	Profiling the Trauma Related Symptoms of Bosni...	The objective of this study was to profile tra...	[]	[]	None	https://doi.org/10.1097/00005053-200007000-00004	None	0	1
2	2	van_de_Schoot_2018.csv	None	Acute panicogenic, anxiogenic and dissociative...	Increased anxiety and panic to inhalation of c...	[]	[]	None	https://doi.org/10.1016/j.jpsychires.2011.01.009	None	0	2
3	3	van_de_Schoot_2018.csv	None	A Pooled Analysis of Gender and Trauma-Type Ef...	To examine effects of gender and trauma type o...	[]	[]	None	https://doi.org/10.4088/jcp.v69n1002	None	0	3
4	4	van_de_Schoot_2018.csv	None	Twelve-Month Use of Mental Health Services in ...	Dramatic changes have occurred in mental healt...	[]	[]	None	https://doi.org/10.1001/archpsyc.62.6.629	None	0	4

State API

The data stored during the review process can be accessed as a pandas DataFrame using the following code:

[8]:
```
with asr.open_state("example.asreview") as state:
    df_results = state.get_results_table()
    print(f"The state contains {len(df_results)} records.")
```

```
The state contains 322 records.
```

The returned `state` instance is of type asreview.SQLState. Note that the state contains less records than the original dataset. This is because by default the simulation stopped after finding all relevant records.

[9]:
```
df_results.tail(10)
```

[9]:
```
     record_id  label classifier querier  balancer feature_extractor  \
312       1376      0        svm     max  balanced             tfidf
313       1589      0        svm     max  balanced             tfidf
314       2368      0        svm     max  balanced             tfidf
315       1359      0        svm     max  balanced             tfidf
316        952      0        svm     max  balanced             tfidf
317       1754      0        svm     max  balanced             tfidf
318       1985      0        svm     max  balanced             tfidf
319       1196      0        svm     max  balanced             tfidf
320       1008      0        svm     max  balanced             tfidf
321       3442      1        svm     max  balanced             tfidf

     training_set               time  note  tags  user_id
312           312  1746457765.998829  None  None     <NA>
313           313  1746457766.009229  None  None     <NA>
314           314  1746457766.019135  None  None     <NA>
315           315  1746457766.028713  None  None     <NA>
316           316  1746457766.038262  None  None     <NA>
317           317   1746457766.04837  None  None     <NA>
318           318   1746457766.05749  None  None     <NA>
319           319   1746457766.06699  None  None     <NA>
320           320  1746457766.076556  None  None     <NA>
321           321  1746457766.086036  None  None     <NA>
```

You can merge the information from the state file with the original dataset.

[10]:
```
dataset_with_results = df_results.reset_index(names="labeling_order").join(
    dataset.get_df().set_index("record_id")
)
dataset_with_results.head()
```

[10]:
```
   labeling_order  record_id  label classifier   querier balancer  \
0               0          0      0       None  top_down     None
1               1          1      0       None  top_down     None
2               2          2      0       None  top_down     None
3               3          3      0       None  top_down     None
4               4          4      0       None  top_down     None

  feature_extractor  training_set               time  note  ...  \
0              None             0  1746457762.147489  None  ...
1              None             1  1746457762.150226  None  ...
2              None             2   1746457762.15211  None  ...
3              None             3  1746457762.153575  None  ...
4              None             4  1746457762.154858  None  ...

               dataset_id  duplicate_of  \
0  van_de_Schoot_2018.csv          None
1  van_de_Schoot_2018.csv          None
2  van_de_Schoot_2018.csv          None
3  van_de_Schoot_2018.csv          None
4  van_de_Schoot_2018.csv          None
```

(continues on next page)

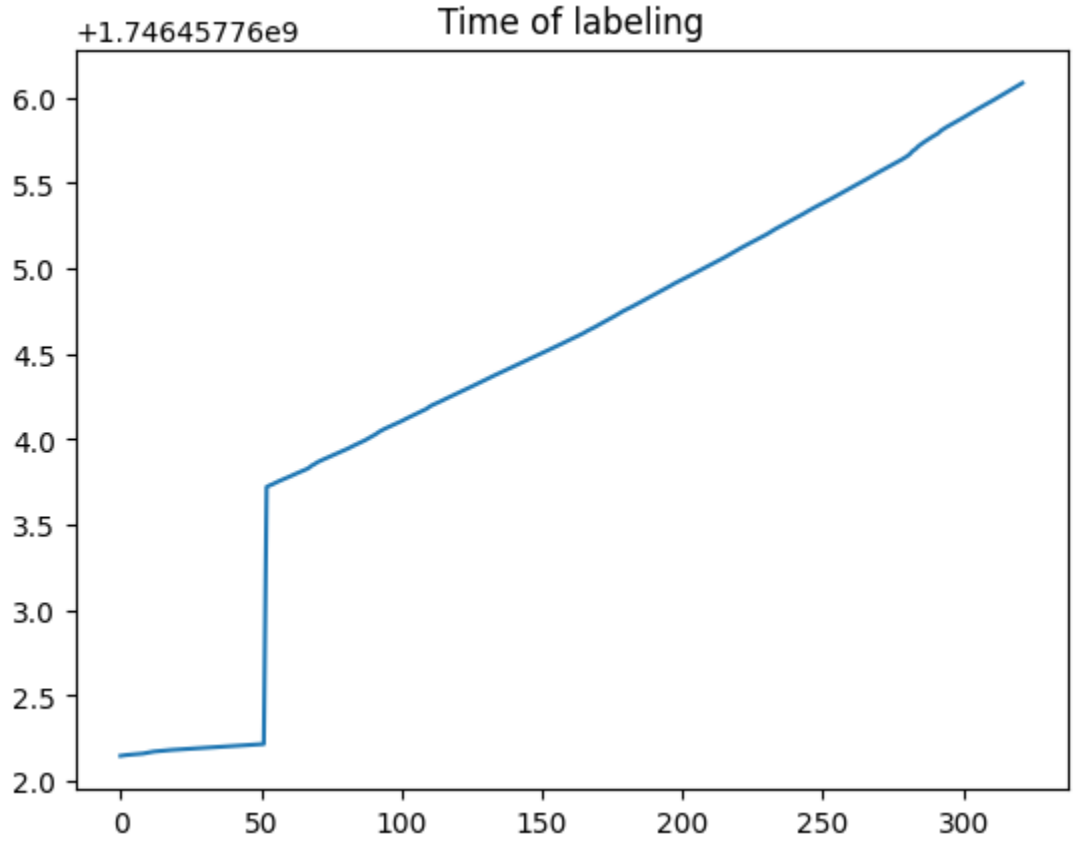
(continued from previous page)

```
                                                title  \
0  Annual Research Review: Resilience and mental ...
1  Profiling the Trauma Related Symptoms of Bosni...
2  Acute panicogenic, anxiogenic and dissociative...
3  A Pooled Analysis of Gender and Trauma-Type Ef...
4  Twelve-Month Use of Mental Health Services in ...

                                             abstract authors keywords  year  \
0  Researchers focused on mental health of confli...      []       []  None
1  The objective of this study was to profile tra...      []       []  None
2  Increased anxiety and panic to inhalation of c...      []       []  None
3  To examine effects of gender and trauma type o...      []       []  None
4  Dramatic changes have occurred in mental healt...      []       []  None

                                               doi   url included
0                https://doi.org/10.1111/jcpp.12053  None        0
1  https://doi.org/10.1097/00005053-200007000-00004  None        0
2  https://doi.org/10.1016/j.jpsychires.2011.01.009  None        0
3              https://doi.org/10.4088/jcp.v69n1002  None        0
4         https://doi.org/10.1001/archpsyc.62.6.629  None        0

[5 rows x 23 columns]
```

There are also multiple functions to obtain one specific variable in the data. For example, to plot the labeling times in a graph, use the following code (keep in mind that this data is from a simulation):

[11]:
```
df_results["time"].plot(title="Time of labeling")
```

[11]:
```
<Axes: title={'center': 'Time of labeling'}>
```



By default, the records that are part of the prior knowledge are included in the results. To obtain the labels use the following code:

[12]:
```
df_results["label"]
```

[12]:
```
0      0
1      0
2      0
3      0
4      0
      ..
317    0
318    0
319    0
320    0
321    1
Name: label, Length: 322, dtype: Int64
```

For normal reviews, the state also contains the ranking of the last iteration of the machine learning model. To get these, use the following code:

[13]:
```
with asr.open_state("example.asreview") as state:
    last_ranking = state.get_last_ranking_table()

last_ranking
```

[13]:
```
Empty DataFrame
Columns: [record_id, ranking, classifier, querier, balancer, feature_extractor, training_
↪set, time]
Index: []
```

asreview	Active learning for Systematic Reviews.

4.1.5 asreview

Active learning for Systematic Reviews.

Active learning for Systematic Reviews (ASReview) is a package for systematic reviews. It is designed to automate the step of screening titles and abstracts in a systematic review. It can be used for screening any type of text data, such as scientific papers, but also, e.g. legal documents or news articles. The package also includes rich simulation functionality, which can be used to evaluate the performance of the active learning model.

Classes

ActiveLearningCycle(querier[, classifier, ...])	Active learner cycle class.
ActiveLearningCycleData(querier, classifier, ...)	
Simulate(X, labels, cycles[, stopper, ...])	ASReview simulation class.
Project(project_path[, project_id])	Project class for ASReview project files.
Database([fp, record_cls, read_only])	Database containing the input data and results.
DataStore([fp, record_cls, read_only, conn_uri])	Data store to hold user input data.
Record(dataset_row, dataset_id, title, ...)	

asreview.ActiveLearningCycle

class asreview.**ActiveLearningCycle**(*querier*, *classifier=None*, *balancer=None*, *feature_extractor=None*, *stopper=None*, *n_query=1*)

Bases: object

Active learner cycle class.

The active learner cycle class is a wrapper around the various learner components.

The classifier is optional, if no classifier is provided, the active learner will only rank the instances based on the query strategy. This can be useful for example if you want random screening.

Parameters

- **querier** (*BaseQueryStrategy*) – The query strategy to use.
- **classifier** (*BaseTrainClassifier*) – The classifier to use. Default is None.
- **balancer** (*BaseTrainClassifier*) – The balance strategy to use. Default is None.
- **feature_extractor** (*BaseFeatureExtraction*) – The feature extraction method to use. Default is None.
- **stopper** (*BaseStopper*) – The stopping criteria. Default is None.
- **n_query** (*int, callable*) – The number of instances to query at once. Default is 1.

Methods

__init__(querier[, classifier, balancer, ...])	
fit(X, y)	Fit the classifier to the data.
from_file(fp[, load])	Load the active learner from a file.
from_meta(cycle_meta_data[, ...])	Load the active learner from a metadata object.
get_n_query(results, labels)	Get the number of records to query at each step in the active learning.
rank(X)	Rank the instances in X.
stop(results, data)	Check if the stopping criteria is met.
to_file(fp)	
to_meta()	
transform(X)	Transform the data.

fit(*X*, *y*)

Fit the classifier to the data.

Parameters

- **X** (*np.array*) – The instances to fit.
- **y** (*np.array*) – The labels of the instances.

classmethod from_file(*fp*, *load=None*)

Load the active learner from a file.

Parameters

- **fp** (*str, Path*) – Review config file.
- **load** (*object*) – Config reader. Default tomllib.load for TOML (.toml) files, otherwise json.load.

classmethod from_meta(*cycle_meta_data*, *skip_feature_extraction=False*)

Load the active learner from a metadata object.

Parameters

cycle_meta_data (*CycleMetaData*) – The metadata object with the active learner settings.

Returns

ActiveLearningCycle – The active learner cycle object.

get_n_query(*results*, *labels*)

Get the number of records to query at each step in the active learning.

n_query can be an integer or a function that takes the results of the simulation as input. If n_query is a function, it should return an integer. n_query can not be larger than the number of records left to label.

Parameters

- **n_query** (*int | callable*) – Number of records to query at each step in the active learning process. Default is 1.
- **results** (*pd.DataFrame*) – The results of the simulation.

Returns

int – Number of records to query at each step in the active learning process.

rank(*X*)

Rank the instances in X.

Parameters
X (*np.array*) – The instances to rank.

Returns
np.array – The ranking of the instances.

stop(*results*, *data*)

Check if the stopping criteria is met.

Parameters

- **results** (*pd.DataFrame*) – The results of the simulation.
- **data** (*pandas.DataFrame*) – The data store object.

Returns
bool – True if the stopping criteria is met, False otherwise.

to_file(*fp*)

to_meta()

transform(*X*)

Transform the data.

Parameters
X (*np.array*) – The instances to transform.

Returns
np.array, scipy.sparse.csr_matrix – The transformed instances.

asreview.ActiveLearningCycleData

class asreview.**ActiveLearningCycleData**(*querier: str*, *classifier: str | None = None*, *balancer: str | None = None*, *feature_extractor: str | None = None*, *stopper: str | None = None*, *querier_param: Optional[dict[str, Any]]=<factory>*, *classifier_param: Optional[dict[str, Any]]=<factory>*, *balancer_param: Optional[dict[str, Any]]=<factory>*, *feature_extractor_param: Optional[dict[str, Any]]=<factory>*, *stopper_param: Optional[dict[str, Any]]=<factory>*, *n_query: int = 1*)

Bases: object

Methods

__init__(querier, classifier, balancer, ...)

Attributes

balancer
classifier
feature_extractor
n_query

continues on next page

Table 5 – continued from previous page

stopper
querier
querier_param
classifier_param
balancer_param
feature_extractor_param
stopper_param

balancer: str | None = None

balancer_param: dict[str, Any] | None

classifier: str | None = None

classifier_param: dict[str, Any] | None

feature_extractor: str | None = None

feature_extractor_param: dict[str, Any] | None

n_query: int = 1

querier: str

querier_param: dict[str, Any] | None

stopper: str | None = None

stopper_param: dict[str, Any] | None

asreview.Simulate

class asreview.**Simulate**(*X*, *labels*, *cycles*, *stopper=None*, *skip_transform=False*, *print_progress=True*, *groups=None*)

Bases: object

ASReview simulation class.

The simulation will stop when all records have been labeled or when the number of steps/queries reaches the stopping.

To seed the simulation, provide the seed to the classifier, query strategy, feature extraction model, and balance strategy or use a global random seed.

Parameters

- **fm** (*numpy.ndarray*) – The feature matrix to use for the simulation.
- **labels** (*numpy.ndarray*, *pandas.Series*, *list*) – The labels to use for the simulation.
- **classifier** (*BaseModel*) – The initialized classifier to use for the simulation.
- **querier** (*BaseQueryModel*) – The initialized query strategy to use for the simulation.
- **balancer** (*BaseBalanceModel*) – The initialized balance strategy to use for the simulation.
- **feature_extractor** (*BaseFeatureModel*) – The initialized feature extraction model to use for the simulation. If None, the name of the feature extraction model is set to None.

- **stopper** (*int, callable*) – The stopping mechanism to use for the simulation. When stopper is None, the simulation stops when all relevant records are found. If an integer, the simulation stops after n queries. A stopper or -1 stops the simulation after all records have been labeled. If class with .stop() method, the simulation stops when the callable returns True. Default is None.
- **skip_transform** (*bool*) – If True, the feature matrix is not computed in the simulation. It is assumed that X is the feature matrix or input to the estimator. Default is False.
- **groups** (*list[tuple[int, int]] | None*) – List of tuples (group_id, record_id). If this is not None, records in the same group will be labeled at the same time in the simulation.

Methods

__init__(X, labels, cycles[, stopper, ...])	
label(record_ids[, cycle])	Label the records with the given record_ids.
review()	Start the review process.
to_sql(fp)	Write the data a sql file.

label(*record_ids*, *cycle=None*)

Label the records with the given record_ids.

Parameters
record_ids (*list*) – The record ids to label.

review()

Start the review process.

to_sql(*fp*)

Write the data a sql file.

Parameters
fp (*str, Path*) – The path to the sqlite file to write the results to. If there is no database yet at the location a new database will be created.

asreview.Project

class asreview.**Project**(*project_path*, *project_id=None*)

Bases: object

Project class for ASReview project files.

This class represents the complete data file for a review project. This data is contained in a single directory with the following files and subdirectories: - *project.json*: A JSON file containing the configuration and metadata of the project. It's structure is described in *schema.py* - *data/*: A directory containing the input data file exactly as provided by the user. When exporting, this input data is merged with the results of the review to get the export file. - *feature_matrices/*: A directory containing all the feature matrices that are generated during the review. - *results.db*: An SQLite database containing all data generated by ASReview: the data parsed from the input file, the labeled records, the last model ranking etc. See *asreview/data* for information on the parsing of the input. See *asreview/state* for information on the model and the labeling decisions.

Methods

__init__(project_path[, project_id])

continues on next page

Table 7 – continued from previous page

add_dataset(fp[, dataset_id, file_writer])	Add a dataset to the project file.
add_feature_matrix(feature_matrix, name)	Add feature matrix to project file.
add_review([cycle, reviewer, status])	Add new review metadata.
close()	Close the project and release all resources.
create(project_path[, project_id, ...])	Initialize the necessary files specific to the web app.
export(export_fp)	
get_feature_matrix(name)	Get the feature matrix from the project file.
get_input_data_reader()	
get_model_config()	Get the current model configuration of the review.
get_review_error()	
label_priors()	Label prior knowledge from a partially labeled dataset.
load(asreview_file, project_path[, ...])	
read_input_data(*args, **kwargs)	
remove_dataset()	Remove dataset from project.
remove_review_error()	
set_review_error(err)	
update_config(**kwargs)	Update project info
update_review([status, model_name, model])	Update review metadata.

Attributes

MODE_SIMULATE
PATH_CONFIG
PATH_CONFIG_LOCK
PATH_DATA_DIR
PATH_DB
PATH_ERROR
PATH_FEATURE_MATRICES
VERSION
config
db
feature_matrices
input_data_fp
review

MODE_SIMULATE = 'simulate'

PATH_CONFIG = 'project.json'

PATH_CONFIG_LOCK = 'project.json.lock'

PATH_DATA_DIR = 'data'

PATH_DB = 'results.db'

PATH_ERROR = 'error.json'

PATH_FEATURE_MATRICES = 'feature_matrices'

VERSION = 3

add_dataset(*fp*, *dataset_id=None*, *file_writer=None*)

Add a dataset to the project file.

Parameters

fp (*str, Path*) – Filepath to the dataset. It will be copied to the correct location in the project file.

add_feature_matrix(*feature_matrix*, *name*)

Add feature matrix to project file.

Parameters

- **feature_matrix** (*numpy.ndarray, scipy.sparse.csr.csr_matrix*) – The feature matrix to add to the project file.
- **name** (*str*) – Name of the feature extractor.

add_review(*cycle=None*, *reviewer=None*, *status='setup'*)

Add new review metadata.

Parameters

- **cycle** – An active learning cycle object to add to the review. This object is used to store the configuration of the active learning cycle to file.
- **reviewer** (*object*) – A reviewer object with to_sql() method.
- **status** (*str*) – The status of the review. One of 'setup', 'running', 'finished'.

close()

Close the project and release all resources.

Closes the database connection if it was opened. Safe to call multiple times.

property config

classmethod create(*project_path*, *project_id=None*, *project_mode='oracle'*, *project_name=None*, *project_tags=None*)

Initialize the necessary files specific to the web app.

property db

export(*export_fp*)

property feature_matrices

get_feature_matrix(*name*)

Get the feature matrix from the project file.

Parameters

name (*str*) – Name of the feature extractor for which to get the cached matrix.

Returns

numpy.ndarray, scipy.sparse – (Sparse) feature matrix.

get_input_data_reader()

get_model_config()

Get the current model configuration of the review.

Returns

dict | None – Dictionary containing the model configuration. Returns None if there is no review yet in the project.

get_review_error()

property input_data_fp

label_priors()

Label prior knowledge from a partially labeled dataset.

If the input dataset is partially labeled (some records have an `included` value of 0 or 1 while others are unlabeled), the labeled records are stored as prior knowledge in the results table.

Fully labeled or fully unlabeled datasets are skipped.

classmethod load(*asreview_file*, *project_path*, *safe_import=False*, *reset_model_if_not_found=False*)

read_input_data(**args*, ***kwargs*)

remove_dataset()

Remove dataset from project.

remove_review_error()

property review

set_review_error(*err*)

update_config(***kwargs*)

Update project info

update_review(*status=None*, *model_name=None*, *model=None*)

Update review metadata.

asreview.Database

class asreview.**Database**(*fp=':memory:'*, *record_cls=<class 'asreview.data.record.Record'>*, *read_only=False*)

Bases: `object`

Database containing the input data and results.

Database contains two parts: the input and the results. For more information on the input, see *asreview.database.store.py*. For more information on the results, see *asreview.database.sqlstate.py*.

Variables

user_version (*str*) – Return the version number of the database.

Methods

__init__([fp, record_cls, read_only])	Initialize the Database.
add_last_ranking(ranked_record_ids, ...[, ...])	Save the ranking of the last iteration of the model.
close()	Close the database and release all resources.
create_tables()	
delete_result(record_id)	
get_decision_changes()	Get the record ids for any decision changes.
get_last_ranking_table()	Get the ranking from the state.
get_pending([user_id])	Get pending records from the results table.
get_pool()	Get the unlabeled, not-pending records in ranking order.

continues on next page

Table 9 – continued from previous page

get_priors()	Get the record ids of the priors.
get_results_record(record_id)	Get the data of a specific query from the results table.
get_results_table([columns, priors, ...])	Get a subset from the results table.
get_unlabeled([groups])	Get the unlabeled record ids in ranking order.
label_record(record_id, label[, tags, user_id])	
query_top_ranked([user_id])	
update_note(record_id[, note])	Change the note of an already labeled or pending record.
update_result(record_id[, label, tags, user_id])	

Attributes

exist_new_labeled_records	Return True if there are new labeled records.
record_table_name	
user_version	Version number of the state.

add_last_ranking(*ranked_record_ids*, *classifier*, *querier*, *balancer*, *feature_extractor*, *training_set=None*)

Save the ranking of the last iteration of the model.

Save the ranking of the last iteration of the model, in the ranking order, so the record on row 0 is ranked first by the model.

Parameters

- **ranked_record_ids** (*list, numpy.ndarray*) – A list of records ids in the order that they were ranked.
- **classifier** (*str*) – Name of the classifier of the model.
- **querier** (*str*) – Name of the query strategy of the model.
- **balancer** (*str*) – Name of the balance strategy of the model.
- **feature_extractor** (*str*) – Name of the feature extraction method of the model.
- **training_set** (*int*) – Number of labeled records available at the time of training.

close()

Close the database and release all resources.

For in-memory databases this will destroy the database. Safe to call multiple times.

create_tables()

delete_result(*record_id*)

property exist_new_labeled_records

Return True if there are new labeled records.

Return True if there are any record labels added since the last time the model ranking was added to the state. Also returns True if no model was trained yet, but priors have been added.

get_decision_changes()

Get the record ids for any decision changes.

Get the record ids of the records whose labels have been changed after the original labeling action.

Returns
pd.DataFrame – Dataframe with columns 'record_id', 'label', 'time', and 'user_id' for each record of which the labeling decision was changed.

`get_last_ranking_table()`

Get the ranking from the state.

Returns
pd.DataFrame – Dataframe with columns 'record_id', 'ranking', 'classifier', 'querier', 'balancer', 'feature_extractor', 'training_set' and 'time'. It has one row for each record in the dataset, and is ordered by ranking.

`get_pending(user_id=None)`

Get pending records from the results table.

Parameters
user_id (*int*) – User id of the user who labeled the records.

Returns
pd.DataFrame – DataFrame with pending results records.

`get_pool()`

Get the unlabeled, not-pending records in ranking order.

Returns
pd.Series – Series containing the record_ids of the unlabeled, not pending records, in the order of the last available ranking. If the state does not yet contain a last ranking, the return value will be an empty dataframe. If multiple records are in the same group, only the base record of the group is returned.

`get_priors()`

Get the record ids of the priors.

Returns
pd.DataFrame – The result records of the priors in the order they were added. If multiple records are in the same group, only the base record of the group is returned.

`get_results_record(record_id)`

Get the data of a specific query from the results table.

Parameters
record_id (*int*) – Record id of which you want the data.

Returns
pd.DataFrame – Dataframe containing the data from the results table with the given record_id and columns.

`get_results_table(columns=None, priors=True, pending=False, groups=False)`

Get a subset from the results table.

Can be used to get any column subset from the results table. Most other get functions use this one, except some that use a direct SQL query for efficiency.

Parameters

- **columns** (*list, str*) – List of columns names of the results table, or a string containing one column name.
- **priors** (*bool*) – Whether to keep the records containing the prior knowledge.
- **pending** (*bool*) – Whether to keep the records which are pending a labeling decision.

- **groups** (*bool*) – Return all the records of a group of records. Be default only returns the base record of each group.

Returns
pd.DataFrame – Dataframe containing the data of the specified columns of the results table.

get_unlabeled(*groups=False*)

Get the unlabeled record ids in ranking order.

Records that have no label or no entry in the results table are considered unlabeled.

Parameters
groups (*bool*) – If True, return all records in each unlabeled group. If False, return only group representatives (record_id == group_id).

Returns
pd.Series – Series of record_ids of unlabeled records ordered by ranking.

label_record(*record_id*, *label*, *tags=None*, *user_id=None*)

query_top_ranked(*user_id=None*)

property record_table_name

update_note(*record_id*, *note=None*)

Change the note of an already labeled or pending record.

Parameters

- **record_id** (*int*) – Id of the record whose label should be changed.
- **note** (*str*) – Note to add to the record.

update_result(*record_id*, *label=None*, *tags=None*, *user_id=None*)

property user_version

Version number of the state.

asreview.DataStore

class asreview.**DataStore**(*fp=':memory:'*, *record_cls=<class 'asreview.data.record.Record'>*, *read_only=False*, *conn_uri=None*)

Bases: object

Data store to hold user input data.

Data input always happens via the record class. This means that if you want to add data to the data store, you will first need to clean it, make sure it has the correct columns and make sure it passes the validations defined in the record class.

Getting data from the store can happen in rows or in columns. If you read rows, you will get record objects as response. If you read columns, you will get pandas objects. If you ask for a single column you get a pandas Series, and if you ask for multiple columns you get a pandas DataFrame.

DataStore uses an SQLite database in the backend and SQLAlchemy ORM to interact with the database.

Methods

__init__([fp, record_cls, read_only, conn_uri])	Initialize the data store.
add_records(records)	Add records to the data store.

continues on next page

Table 11 – continued from previous page

create_tables()	Initialize the tables containing the data.
delete_record(record_id)	Delete a record from the store.
get_df()	Get all data from the data store as a pandas DataFrmae.
get_groups([record_id])	Get the record groups.
get_records([record_id])	Get the records with the given record identifiers.
is_empty()	
set_groups(groups)	Add record group information to the data store.

Attributes

columns	
pandas_dtype_mapping	pandas data type}

add_records(*records*)

Add records to the data store.

Parameters
records (*list[self.record_cls]*) – List of records to add to the store.

Raises
ValueError – If some *record.duplicate_of* points to a non-existing record_id.

property columns

create_tables()

Initialize the tables containing the data.

If you are creating a new data store, you will need to call this method before adding data to the data store.

delete_record(*record_id*)

Delete a record from the store.

WARNING: This method is purely here for completeness, it should not be used in any production setting. Deleting records can lead to undefined behavior because we make assumptions about the record_id in other parts of the code.

get_df()

Get all data from the data store as a pandas DataFrmae.

Returns
pd.DataFrame

get_groups(*record_id=None*)

Get the record groups.

Parameters
record_id (*int | None*) – Get only the group containing the record with this record_id.

Returns
list[tuple[int, int]] – List of tuples (group_id, record_id) ordered by group id. The tuples values are also accessible by the attribute names (so *tuple.group_id* and *tuple.record_id*).

get_records(*record_id=None*)

Get the records with the given record identifiers.

Parameters
record_id (*int | list[int] | None*) – Record identifier or list record identifiers. If None, get all records.

Returns
asreview.data.record.Record | list[asreview.data.record.Record] | None

is_empty()

property pandas_dtype_mapping

pandas data type}

Type
Mapping {column name

set_groups(*groups*)

Add record group information to the data store.

Parameters
groups (*list[tuple[int,int]]*) – List of tuples (group_id, record_id). This data is added to the record as the *duplicate_of* attribute. The data store will normalize these values: One record is chosen as the root, satisfying *root.duplicate_of = None*. All other records in the group will get *record.duplicate_of = root.record_id*.

asreview.Record

class asreview.**Record**(*dataset_row: int, dataset_id: str, title: str = '', abstract: str = '', authors: list = <factory>, keywords: list = <factory>, year: int | None = None, doi: str | None = None, url: str | None = None, included: int | None = None*)

Bases: Base

Methods

__init__(dataset_row, dataset_id[, title, ...])	A simple constructor that allows initialization from kwargs.
get_columns()	Get the list of database column names.
get_pandas_dtype_mapping()	Get the mapping from record column name to pandas data type.
validate_included(key, included)	
validate_list_of_string(key, value)	
validate_optional_integer(key, value)	
validate_string(key, value)	

Attributes

abstract
authors
dataset_id
dataset_row
doi
duplicate_of
group_id
included

continues on next page

Table 14 – continued from previous page

keywords	
metadata	Refers to the `_schema.MetaData` collection that will be used for new `_schema.Table` objects.
record_id	
registry	Refers to the `_orm.registry` in use where new `_orm.Mapper` objects will be associated.
title	
type_annotation_map	
url	
year	

abstract: Mapped[str]

authors: Mapped[list]

dataset_id: Mapped[str]

dataset_row: Mapped[int]

doi: Mapped[str | None]

duplicate_of

classmethod get_columns()

Get the list of database column names.

Returns
list[str]

classmethod get_pandas_dtype_mapping()

Get the mapping from record column name to pandas data type.

By default it uses the mapping from *__sqlalchemy_to_pandas_dtype_mapping__*, but you can overwrite this method if you need different behavior.

Returns
dict[str, str] – Dictionary whose keys are record column names and the values are the corresponding pandas data type that the column should have when reading it into a pandas object.

group_id

included: Mapped[int | None]

keywords: Mapped[list]

metadata: ClassVar[MetaData] = MetaData()

Refers to the `_schema.MetaData` collection that will be used for new `_schema.Table` objects.

See also

orm_declarative_metadata

record_id: Mapped[int]

registry: ClassVar[registry] = <sqlalchemy.orm.decl_api.registry object>

Refers to the _orm.registry in use where new _orm.Mapper objects will be associated.

title: Mapped[str]

type_annotation_map = {<class 'list'>: <class 'sqlalchemy.sql.sqltypes.JSON'>}

url: Mapped[str | None]

validate_included(*key*, *included*)

validate_list_of_string(*key*, *value*)

validate_optional_integer(*key*, *value*)

validate_string(*key*, *value*)

year: Mapped[int | None]

Functions

is_project(project_dir)	Check if the given path is a valid ASReview project.
load_dataset(name[, dataset_id, db, record_cls])	Load dataset from file, URL, or plugin.
open_db(fp[, read_only])	Open a database.
extensions(group)	Get the extension class from an entry point.
get_extension(group, name)	Get the extension class from an entry point.
load_extension(group, name)	Load the extension class from an entry point.

asreview.is_project

asreview.**is_project**(*project_dir*)

Check if the given path is a valid ASReview project.

Parameters

project_dir (*str | Path*) – The path to the project directory.

Returns

bool – True if the path is a valid ASReview project, False otherwise.

asreview.load_dataset

asreview.**load_dataset**(*name*, *dataset_id=None*, *db=None*, *record_cls=<class 'asreview.data.record.Record'>*, ***kwargs*)

Load dataset from file, URL, or plugin.

Parameters

- **name** (*str, pathlib.Path*) – File path, URL, or alias of extension dataset.
- **dataset_id** (*str, optional*) – dataset_id that the records in the dataset should get. If not this will be the string form of the name. By default None.
- **db** (*asreview.database.database.Database, optional*) – Database in which to load the records of the dataset. If None, an in memory database is created. By default None.
- **record_cls** (*Type[asreview.data.record.Base], optional*) – Record type to use for the dataset records, by default Record

- **kwargs** (*dict, optional*) – Keyword arguments passed to *load_records*.

Returns
asreview.Database – Database containing the records of the input file.

asreview.open_db

asreview.**open_db**(*fp*, *read_only=False*)

Open a database.

Parameters

- **fp** (*path-like*) – File path to the database
- **read_only** (*bool, optional*) – Whether to create a new database if one doesn't exist yet and whether the opened database will be in read only mode or not.

Returns
Database – ASReview database.

Raises

- **FileNotFoundError** – If *read_only* and there is no file at *fp*.
- **ValueError** – If *read_only* and there is no valid database at *fp*.

asreview.extensions

asreview.**extensions**(*group*)

Get the extension class from an entry point.

Parameters
group (*str*) – The group of the extension.

Returns
dict – The class corresponding to the extension.

asreview.get_extension

asreview.**get_extension**(*group*, *name*)

Get the extension class from an entry point.

Parameters

- **group** (*str*) – The group of the extension.
- **name** (*str*) – The name of the extension.

Returns
class – The class corresponding to the extension.

asreview.load_extension

asreview.**load_extension**(*group*, *name*)

Load the extension class from an entry point.

Parameters

- **group** (*str*) – The group of the extension.
- **name** (*str*) – The name of the extension.

Returns
class – The class corresponding to the extension.

Exceptions

ProjectError	
ProjectNotFoundError	

asreview.ProjectError

exception asreview.**ProjectError**

asreview.ProjectNotFoundError

exception asreview.**ProjectNotFoundError**

Modules

data	
models	
metrics	Performance metrics for activate learning results.
datasets	

asreview.data

Classes

CSVReader()	CVS file reader.
CSVWriter()	CSV file writer.
ExcelReader()	Excel file reader.
ExcelWriter()	Excel file writer.
RISReader()	RIS file reader.
RISWriter()	RIS file writer.
TSVWriter()	TSV file writer.

asreview.data.CSVReader

class asreview.data.**CSVReader**

Bases: *BaseReader*

CVS file reader.

Methods

__init__()	
clean_data(df)	Clean the raw data.
read_data(fp)	Import dataset.
read_records(fp, dataset_id[, record_cls])	
standardize_column_names(df)	Standardize column names of input data.

continues on next page

Table 19 – continued from previous page

to_records(df[, dataset_id, record_cls])	Turn the cleaned data into records.

Attributes

mime_types
read_format
write_format

classmethod clean_data(*df*)

Clean the raw data.

Parameters
df (*pd.DataFrame*) – Data to clean. This should be of the same type as the output of *read_data*.

Returns
pd.DataFrame – Cleaned data. By default it standardizes the column names, some data types and missing values.

mime_types = {'text/csv': ['.csv'], 'text/plain': ['.csv', '.tsv', '.tab'], 'text/tab-separated-values': ['.tsv', '.tab']}

classmethod read_data(*fp*)

Import dataset.

Parameters
fp (*str, pathlib.Path*) – File path to the CSV file.

Returns
list – List with entries.

read_format = ['.csv', '.tab', '.tsv']

classmethod read_records(*fp*, *dataset_id*, *record_cls=<class 'asreview.data.record.Record'>*, **args*, ***kwargs*)

classmethod standardize_column_names(*df*)

Standardize column names of input data.

The reader can accept multiple names for a specific type of data, for example both 'title' and 'primary_title' could refer to the column containing the title data. This function makes sure the correct columns are used. See also the attribute *__alternative_column_names__* for customizing this behavior.

Parameters
df (*pd.DataFrame*) – Dataframe containing raw data.

Returns
pd.DataFrame – Dataframe with column names lowercased and stripped of white space. In addition, for the columns in *__alternative_column_names__*, the first alternative column name in the data will be used as input for the column values.

classmethod to_records(*df*, *dataset_id=None*, *record_cls=<class 'asreview.data.record.Record'>*)

Turn the cleaned data into records.

Parameters

- **df** (*pd.DataFrame*) – Cleaned data.

- **dataset_id** (*str, optional*) – Identifier of the dataset, by default None
- **record_cls** (*asreview.data.record.Base, optional*) – Record class to use, by default Record

Returns
list[Record] – List of records.

write_format = ['.csv', '.tsv', '.xlsx']

asreview.data.CSVWriter

class asreview.data.**CSVWriter**

Bases: object

CSV file writer.

Methods

__init__()	
write_data(df, fp[, sep])	Export dataset.

Attributes

label
name
write_format

label = 'CSV (UTF-8)'

name = 'csv'

classmethod write_data(*df*, *fp*, *sep=','*)

Export dataset.

Parameters

- **df** (*pandas.Dataframe*) – Dataframe of all available record data.
- **fp** (*str, NoneType*) – Filepath or None for buffer.
- **sep** (*str*) – Seperator of the file.

Returns
CSV file – Dataframe of all available record data.

write_format = '.csv'

asreview.data.ExcelReader

class asreview.data.**ExcelReader**

Bases: *BaseReader*

Excel file reader.

Methods

__init__()	
clean_data(df)	Clean the raw data.
read_data(fp)	Import dataset.
read_records(fp, dataset_id[, record_cls])	
standardize_column_names(df)	Standardize column names of input data.
to_records(df[, dataset_id, record_cls])	Turn the cleaned data into records.

Attributes

mime_types
read_format
write_format

classmethod clean_data(*df*)

Clean the raw data.

Parameters
df (*pd.DataFrame*) – Data to clean. This should be of the same type as the output of *read_data*.

Returns
pd.DataFrame – Cleaned data. By default it standardizes the column names, some data types and missing values.

mime_types = {'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet': ['.xlsx']}

classmethod read_data(*fp*)

Import dataset.

Parameters
fp (*str, pathlib.Path*) – File path to the Excel file (.xlsx).

Returns
list – List with entries.

read_format = ['.xlsx']

classmethod read_records(*fp*, *dataset_id*, *record_cls=<class 'asreview.data.record.Record'>*, **args*, ***kwargs*)

classmethod standardize_column_names(*df*)

Standardize column names of input data.

The reader can accept multiple names for a specific type of data, for example both 'title' and 'primary_title' could refer to the column containing the title data. This function makes sure the correct columns are used. See also the attribute *__alternative_column_names__* for customizing this behavior.

Parameters
df (*pd.DataFrame*) – Dataframe containing raw data.

Returns
pd.DataFrame – Dataframe with column names lowercased and stripped of white space.

In addition, for the columns in *__alternative_column_names__*, the first alternative column name in the data will be used as input for the column values.

classmethod to_records(*df*, *dataset_id=None*, *record_cls=<class 'asreview.data.record.Record'>*)

Turn the cleaned data into records.

Parameters

- **df** (*pd.DataFrame*) – Cleaned data.
- **dataset_id** (*str, optional*) – Identifier of the dataset, by default None
- **record_cls** (*asreview.data.record.Base, optional*) – Record class to use, by default Record

Returns

list[Record] – List of records.

write_format = ['.csv', '.tsv', '.xlsx']

asreview.data.ExcelWriter

class asreview.data.**ExcelWriter**

Bases: object

Excel file writer.

Methods

__init__()	
write_data(df, fp)	Export dataset.

Attributes

label
name
write_format

label = 'Excel'

name = 'xlsx'

classmethod write_data(*df*, *fp*)

Export dataset.

Parameters

- **df** (*pandas.Dataframe*) – Dataframe of all available record data.
- **fp** (*str, NoneType*) – Filepath or None for buffer.

Returns

Excel file – Dataframe of all available record data.

write_format = '.xlsx'

asreview.data.RISReader

class asreview.data.**RISReader**

Bases: *BaseReader*

RIS file reader.

Methods

__init__()	
clean_data(df)	Clean the raw data.
read_data(fp)	Import dataset.
read_records(fp, dataset_id[, record_cls])	
standardize_column_names(df)	Standardize column names of input data.
to_records(df[, dataset_id, record_cls])	Turn the cleaned data into records.

Attributes

mime_types
read_format
write_format

classmethod clean_data(*df*)

Clean the raw data.

Parameters
df (*pd.DataFrame*) – Data to clean. This should be of the same type as the output of *read_data*.

Returns
pd.DataFrame – Cleaned data. By default it standardizes the column names, some data types and missing values.

mime_types = {'application/x-research-info-systems': ['.ris', '.txt'], 'text/plain': ['.txt', '.ris']}

classmethod read_data(*fp*)

Import dataset.

Parameters
fp (*str*, *pathlib.Path*) – File path to the RIS file.

Returns
pd.DataFrame – Dataframe with entries. If the notes field contains a note with the text *ASReview_relevant*, *ASReview_irrelevant* or *ASReview_not_seen*, the data frame will have a column *included* with the value *1*, *0* or *None*.

Raises
ValueError – File with unrecognized encoding is used as input.

read_format = ['.ris', '.txt']

classmethod read_records(*fp*, *dataset_id*, *record_cls=<class 'asreview.data.record.Record'>*, **args*, ***kwargs*)

classmethod standardize_column_names(*df*)

Standardize column names of input data.

The reader can accept multiple names for a specific type of data, for example both 'title' and 'primary_title' could refer to the column containing the title data. This function makes sure the correct columns are used. See also the attribute *__alternative_column_names__* for customizing this behavior.

Parameters
df (*pd.DataFrame*) – Dataframe containing raw data.

Returns
pd.DataFrame – Dataframe with column names lowercased and stripped of white space. In addition, for the columns in *__alternative_column_names__*, the first alternative column name in the data will be used as input for the column values.

classmethod to_records(*df*, *dataset_id=None*, *record_cls=<class 'asreview.data.record.Record'>*)

Turn the cleaned data into records.

Parameters

- **df** (*pd.DataFrame*) – Cleaned data.
- **dataset_id** (*str, optional*) – Identifier of the dataset, by default None
- **record_cls** (*asreview.data.record.Base, optional*) – Record class to use, by default Record

Returns
list[Record] – List of records.

write_format = ['.csv', '.tsv', '.xlsx', '.ris']

asreview.data.RISWriter

class asreview.data.**RISWriter**

Bases: object

RIS file writer.

Methods

__init__()	
write_data(df, fp)	Export dataset.

Attributes

caution
label
name
write_format

caution = 'Available only if you imported a RIS file when creating the project'

label = 'RIS'

name = 'ris'

classmethod write_data(*df*, *fp*)

Export dataset.

Parameters

- **df** (*pd.Dataframe*) – Dataframe of all available record data.
- **fp** (*str, pathlib.Path*) – File path to the RIS file, if exists.

Returns

RIS file – Dataframe of all available record data. Any column from the data frame that starts with *asreview_* is added to the RIS file as note in the notes field of the form: *asreview_{column_name}: json.dumps({column_value})*. If the dataframe contains a column *asreview_label*, also a note is added with the value *ASReview_relevant*, *ASReview_irrelevant* or *ASReview_not_seen* corresponding to the value *1*, *0* or *None* in that column.

write_format = '.ris'

asreview.data.TSVWriter

class asreview.data.**TSVWriter**

Bases: object

TSV file writer.

Methods

__init__()	
write_data(df, fp[, sep])	Export dataset.

Attributes

label
name
write_format

label = 'TSV (UTF-8)'

name = 'tsv'

classmethod write_data(*df*, *fp*, *sep='\t'*)

Export dataset.

Parameters

- **df** (*pandas.Dataframe*) – Dataframe of all available record data.
- **fp** (*str, NoneType*) – Filepath or None for buffer.
- **sep** (*str*) – Seperator of the file.

Returns

TSV file – Dataframe of all available record data.

write_format = '.tsv'

Modules

base

asreview.data.base

Classes

BaseReader()	Base class for data readers.

asreview.data.base.BaseReader

class asreview.data.base.**BaseReader**

Bases: ABC

Base class for data readers.

Reading data from a file happens in three steps: read the raw data, perform data cleaning and turn it into *Record* instances. This happens in *read_data*, *clean_data* and *to_records*. Anyone implementing a *BaseReader* should provide an implementation of *read_data*. There are default implementations of *clean_data* and *to_records*. They assume that *read_data* produces a pandas DataFrame. There are a number of ways to customize the default cleaning behavior, see the comments next to the class attributes.

Methods

__init__()	
clean_data(df)	Clean the raw data.
read_data(fp, *args, **kwargs)	Read the raw data from a file.
read_records(fp, dataset_id[, record_cls])	
standardize_column_names(df)	Standardize column names of input data.
to_records(df[, dataset_id, record_cls])	Turn the cleaned data into records.

classmethod clean_data(*df*)

Clean the raw data.

Parameters
df (*pd.DataFrame*) – Data to clean. This should be of the same type as the output of *read_data*.

Returns
pd.DataFrame – Cleaned data. By default it standardizes the column names, some data types and missing values.

abstractmethod classmethod read_data(*fp*, **args*, ***kwargs*)

Read the raw data from a file.

The data type of the output should be equal to the data type of the input of *clean_data*. Typically this will mean a pandas DataFrame, but anyone creating a custom class can choose a different data type.

This method should not perform any cleaning of the data. That way data writers can add columns to a dataset without changing the original data: Use *reader.read_data* to get the data, then add the column, then write away the data to a file.

Parameters
fp (*Path*) – Filepath of the file to read.

Returns
pd.DataFrame – A dataframe of user input data that has not been cleaned yet.

classmethod read_records(*fp*, *dataset_id*, *record_cls=<class 'asreview.data.record.Record'>*, **args*, ***kwargs*)

classmethod standardize_column_names(*df*)

Standardize column names of input data.

The reader can accept multiple names for a specific type of data, for example both 'title' and 'primary_title' could refer to the column containing the title data. This function makes sure the correct columns are used. See also the attribute *__alternative_column_names__* for customizing this behavior.

Parameters
df (*pd.DataFrame*) – Dataframe containing raw data.

Returns
pd.DataFrame – Dataframe with column names lowercased and stripped of white space. In addition, for the columns in *__alternative_column_names__*, the first alternative column name in the data will be used as input for the column values.

classmethod to_records(*df*, *dataset_id=None*, *record_cls=<class 'asreview.data.record.Record'>*)

Turn the cleaned data into records.

Parameters

- **df** (*pd.DataFrame*) – Cleaned data.
- **dataset_id** (*str, optional*) – Identifier of the dataset, by default None
- **record_cls** (*asreview.data.record.Base, optional*) – Record class to use, by default Record

Returns
list[Record] – List of records.

asreview.models

Functions

get_ai_config([name])	Get the AI configuration.

asreview.models.get_ai_config

asreview.models.**get_ai_config**(*name=None*)

Get the AI configuration.

Parameters
name (*str*) – The name of the AI configuration. If None, the default AI configuration is returned.

Returns
dict – The default AI configuration.

Modules

balancers	
classifiers	
feature_extractors	
queriers	
stoppers	Stopper mechanisms for the review process.

asreview.models.balancers

Classes

Balanced([ratio])	Balanced sample weight

asreview.models.balancers.Balanced

class asreview.models.balancers.**Balanced**(*ratio=1.0*)

Bases: BaseEstimator

Balanced sample weight

Parameters
ratio (*float*) – The ratio of the number of samples in class 0 to the number of samples in class 1.

Methods

__init__([ratio])	
compute_sample_weight(y)	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

compute_sample_weight(*y*)

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*dict*) – Parameter names mapped to their values.

`label = 'Balanced Sample Weight'`

`name = 'balanced'`

`set_params`(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self (*estimator instance*) – Estimator instance.

asreview.models.classifiers

Classes

SVM([penalty, loss, dual, tol, C, ...])	Support vector machine classifier.
RandomForest([n_estimators, criterion, ...])	Random forest classifier.
NaiveBayes(*[, alpha, force_alpha, ...])	Naive Bayes classifier.
Logistic([penalty, C, l1_ratio, dual, tol, ...])	Logistic regression classifier.

asreview.models.classifiers.SVM

class asreview.models.classifiers.**SVM**(*penalty='l2', loss='squared_hinge', * (Keyword-only parameters separator (PEP 3102)), dual='auto', tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*)

Bases: `LinearSVC`

Support vector machine classifier.

Based on the sklearn implementation of the support vector machine sklearn.svm.LinearSVC.

Methods

`__init__`([penalty, loss, dual, tol, C, ...])	
decision_function(X)	Predict confidence scores for samples.
densify()	Convert coefficient matrix to dense array format.
fit(X, y[, sample_weight])	Fit the model according to the given training data.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class labels for samples in X.

continues on next page

Table 42 – continued from previous page

score(X, y[, sample_weight])	Return accuracy on provided data and labels.
set_fit_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `fit` method.
set_params(**params)	Set the parameters of this estimator.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.
sparsify()	Convert coefficient matrix to sparse format.

Attributes

label
name

decision_function(*X*)

Predict confidence scores for samples.

The confidence score for a sample is proportional to the signed distance of that sample to the hyperplane.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The data matrix for which we want to get the confidence scores.

Returns

scores (*ndarray of shape (n_samples,) or (n_samples, n_classes)*) – Confidence scores per *(n_samples, n_classes)* combination. In the binary case, confidence score for *self.classes_[1]* where >0 means this class would be predicted.

densify()

Convert coefficient matrix to dense array format.

Converts the `coef_` member (back) to a numpy.ndarray. This is the default format of `coef_` and is required for fitting, so calling this method is only required on models that have previously been sparsified; otherwise, it is a no-op.

Returns

self – Fitted estimator.

fit(*X*, *y*, *sample_weight=None*)

Fit the model according to the given training data.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training vector, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*array-like of shape (n_samples,)*) – Target vector relative to X.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

  Added in version 0.18.

Returns

self (*object*) – An instance of the estimator.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*dict*) – Parameter names mapped to their values.

label = 'Support vector machine'

name = 'svm'

predict(*X*)

Predict class labels for samples in X.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The data matrix for which we want to get the predictions.

Returns

y_pred (*ndarray of shape (n_samples,)*) – Vector containing the class labels for each sample.

score(*X*, *y*, *sample_weight=None*)

Return accuracy on provided data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns

score (*float*) – Mean accuracy of `self.predict(X)` w.r.t. *y*.

set_fit_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *SVM*

Configure whether metadata should be requested to be passed to the `fit` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.

- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `fit`.

Returns
self (*object*) – The updated object.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

set_score_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *SVM*

Configure whether metadata should be requested to be passed to the `score` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self (*object*) – The updated object.

sparsify()

Convert coefficient matrix to sparse format.

Converts the `coef_` member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation.

The `intercept_` member is not converted.

Warning

This method is not supported for estimators fitted with array API inputs (i.e. when `sklearn.config_context()` is used with `array_api_dispatch=True`). The call may succeed but subsequent calls to *predict()* and other methods involving passing arrays may raise or return unexpected results.

Returns

self – Fitted estimator.

Notes

For non-sparse models, i.e. when there are not many zeros in `coef_`, this may actually *increase* memory usage, so use this method with care. A rule of thumb is that the number of zero elements, which can be computed with `(coef_ == 0).sum()`, must be more than 50% for this to provide significant benefits.

After calling this method, further fitting with the partial_fit method (if any) will not work until you call densify.

asreview.models.classifiers.RandomForest

class asreview.models.classifiers.**RandomForest**(*n_estimators=100*, ***, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='sqrt'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*, *monotonic_cst=None*)

Bases: `RandomForestClassifier`

Random forest classifier.

Based on the sklearn implementation of the random forest sklearn.ensemble.RandomForestClassifier.

Methods

`__init__`([n_estimators, criterion, ...])	
apply(X)	Apply trees in the forest to X, return leaf indices.
decision_path(X)	Return the decision path in the forest.
fit(X, y[, sample_weight])	Build a forest of trees from the training set (X, y).
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.

continues on next page

Table 44 – continued from previous page

predict(X)	Predict class for X.
predict_log_proba(X)	Predict class log-probabilities for X.
predict_proba(X)	Predict class probabilities for X.
score(X, y[, sample_weight])	Return accuracy on provided data and labels.
set_fit_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `fit` method.
set_params(**params)	Set the parameters of this estimator.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.

Attributes

*estimators_samples_*	The subset of drawn samples for each base estimator.
*feature_importances_*	The impurity-based feature importances.
label	
name	

apply(*X*)

Apply trees in the forest to X, return leaf indices.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csr_matrix`.

Returns

X_leaves (*ndarray of shape (n_samples, n_estimators)*) – For each datapoint x in X and for each tree in the forest, return the index of the leaf x ends up in.

decision_path(*X*)

Return the decision path in the forest.

Added in version 0.18.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csr_matrix`.

Returns

- **indicator** (*sparse matrix of shape (n_samples, n_nodes)*) – Return a node indicator matrix where non zero elements indicates that the samples goes through the nodes. The matrix is of CSR format.
- **n_nodes_ptr** (*ndarray of shape (n_estimators + 1,)*) – The columns from indicator[n_nodes_ptr[i]:n_nodes_ptr[i+1]] gives the indicator value for the i-th estimator.

property estimators_samples_

The subset of drawn samples for each base estimator.

Returns a dynamically generated list of indices identifying the samples used for fitting each member of the ensemble, i.e., the in-bag samples.

Note: the list is re-created at each call to the property in order to reduce the object memory footprint by not storing the sampling data. Thus fetching the property may be slower than expected.

property feature_importances_

The impurity-based feature importances.

The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.

Warning: impurity-based feature importances can be misleading for high cardinality features (many unique values). See `sklearn.inspection.permutation_importance()` as an alternative.

Returns

feature_importances_ (*ndarray of shape (n_features,)*) – The values of this array sum to 1, unless all trees are single node trees consisting of only the root node, in which case it will be an array of zeros.

fit(*X*, *y*, *sample_weight=None*)

Build a forest of trees from the training set (X, y).

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The training input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csc_matrix`.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – The target values (class labels in classification, real numbers in regression).
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node. In the case of classification, splits are also ignored if they would result in any single class carrying a negative weight in either child node.

Returns

self (*object*) – Fitted estimator.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*dict*) – Parameter names mapped to their values.

label = 'Random forest'

name = 'rf'

predict(*X*)

Predict class for X.

The predicted class of an input sample is a vote by the trees in the forest, weighted by their probability estimates. That is, the predicted class is the one with highest mean probability estimate across the trees.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csr_matrix`.

Returns

y (*ndarray of shape (n_samples,) or (n_samples, n_outputs)*) – The predicted classes.

`predict_log_proba`(*X*)

Predict class log-probabilities for X.

The predicted class log-probabilities of an input sample is computed as the log of the mean predicted class probabilities of the trees in the forest.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csr_matrix`.

Returns

p (*ndarray of shape (n_samples, n_classes), or a list of such arrays*) – The class probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

`predict_proba`(*X*)

Predict class probabilities for X.

The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the trees in the forest. The class probability of a single tree is the fraction of samples of the same class in a leaf.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, its dtype will be converted to `dtype=np.float32`. If a sparse matrix is provided, it will be converted into a sparse `csr_matrix`.

Returns

p (*ndarray of shape (n_samples, n_classes), or a list of such arrays*) – The class probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

`score`(*X*, *y*, *sample_weight=None*)

Return accuracy on provided data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns

score (*float*) – Mean accuracy of `self.predict(X)` w.r.t. *y*.

`set_fit_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *RandomForest*

Configure whether metadata should be requested to be passed to the `fit` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters

sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `fit`.

Returns

self (*object*) – The updated object.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self (*estimator instance*) – Estimator instance.

set_score_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *RandomForest*

Configure whether metadata should be requested to be passed to the `score` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters

sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns

self (*object*) – The updated object.

asreview.models.classifiers.NaiveBayes

class asreview.models.classifiers.**NaiveBayes**(*, *alpha=1.0*, *force_alpha=True*, *fit_prior=True*, *class_prior=None*)

Bases: `MultinomialNB`

Naive Bayes classifier.

Based on the sklearn implementation of the naive bayes sklearn.naive_bayes.MultinomialNB.

Methods

`__init__`(*[, alpha, force_alpha, fit_prior, ...])	
fit(X, y[, sample_weight])	Fit Naive Bayes classifier according to X, y.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
partial_fit(X, y[, classes, sample_weight])	Incremental fit on a batch of samples.
predict(X)	Perform classification on an array of test vectors X.
predict_joint_log_proba(X)	Return joint log probability estimates for the test vector X.
predict_log_proba(X)	Return log-probability estimates for the test vector X.
predict_proba(X)	Return probability estimates for the test vector X.
score(X, y[, sample_weight])	Return accuracy on provided data and labels.
set_fit_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `fit` method.
set_params(**params)	Set the parameters of this estimator.
set_partial_fit_request(*[, classes, ...])	Configure whether metadata should be requested to be passed to the `partial_fit` method.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.

Attributes

label
name

fit(*X*, *y*, *sample_weight=None*)

Fit Naive Bayes classifier according to X, y.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training vectors, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*array-like of shape (n_samples,)*) – Target values.

- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Weights applied to individual samples (1. for unweighted).

Returns
self (*object*) – Returns the instance itself.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Naive Bayes'

name = 'nb'

partial_fit(*X*, *y*, *classes=None*, *sample_weight=None*)

Incremental fit on a batch of samples.

This method is expected to be called several times consecutively on different chunks of a dataset so as to implement out-of-core or online learning.

This is especially useful when the whole dataset is too big to fit in memory at once.

This method has some performance overhead hence it is better to call partial_fit on chunks of data that are as large as possible (as long as fitting in the memory budget) to hide the overhead.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training vectors, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*array-like of shape (n_samples,)*) – Target values.
- **classes** (*array-like of shape (n_classes,), default=None*) – List of all the classes that can possibly appear in the y vector.

  Must be provided at the first call to partial_fit, can be omitted in subsequent calls.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Weights applied to individual samples (1. for unweighted).

Returns
self (*object*) – Returns the instance itself.

predict(*X*)

Perform classification on an array of test vectors X.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The input samples.

Returns
C (*ndarray of shape (n_samples,)*) – Predicted target values for X.

`predict_joint_log_proba`(*X*)

Return joint log probability estimates for the test vector X.

For each row x of X and class y, the joint log probability is given by `log P(x, y) = log P(y) + log P(x|y)`, where `log P(y)` is the class prior probability and `log P(x|y)` is the class-conditional probability.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The input samples.

Returns
C (*ndarray of shape (n_samples, n_classes)*) – Returns the joint log-probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute classes_.

`predict_log_proba`(*X*)

Return log-probability estimates for the test vector X.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The input samples.

Returns
C (*array-like of shape (n_samples, n_classes)*) – Returns the log-probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute classes_.

`predict_proba`(*X*)

Return probability estimates for the test vector X.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The input samples.

Returns
C (*array-like of shape (n_samples, n_classes)*) – Returns the probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute classes_.

`score`(*X*, *y*, *sample_weight=None*)

Return accuracy on provided data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns
score (*float*) – Mean accuracy of `self.predict(X)` w.r.t. *y*.

set_fit_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *NaiveBayes*

Configure whether metadata should be requested to be passed to the `fit` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters

sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `fit`.

Returns

self (*object*) – The updated object.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self (*estimator instance*) – Estimator instance.

set_partial_fit_request(*, *classes: bool | None | str = '$UNCHANGED$'*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *NaiveBayes*

Configure whether metadata should be requested to be passed to the `partial_fit` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `partial_fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `partial_fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters

- **classes** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for classes parameter in partial_fit.
- **sample_weight** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for sample_weight parameter in partial_fit.

Returns

self (*object*) – The updated object.

set_score_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *NaiveBayes*

Configure whether metadata should be requested to be passed to the score method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with enable_metadata_routing=True (see sklearn.set_config()). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- True: metadata is requested, and passed to score if provided. The request is ignored if metadata is not provided.
- False: metadata is not requested and the meta-estimator will not pass it to score.
- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters

sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for sample_weight parameter in score.

Returns

self (*object*) – The updated object.

asreview.models.classifiers.Logistic

class asreview.models.classifiers.**Logistic**(*penalty='deprecated'*, *, *C=1.0*, *l1_ratio=0.0*, *dual=False*, *tol=0.0001*, *fit_intercept=True*, *intercept_scaling=1*, *class_weight=None*, *random_state=None*, *solver='lbfgs'*, *max_iter=100*, *verbose=0*, *warm_start=False*, *n_jobs=None*)

Bases: LogisticRegression

Logistic regression classifier.

Based on the sklearn implementation of the logistic regression sklearn.linear_model.LogisticRegression.

Methods

`__init__`([penalty, C, l1_ratio, dual, tol, ...])	
decision_function(X)	Predict confidence scores for samples.
densify()	Convert coefficient matrix to dense array format.
fit(X, y[, sample_weight])	Fit the model according to the given training data.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class labels for samples in X.
predict_log_proba(X)	Predict logarithm of probability estimates.
predict_proba(X)	Probability estimates.
score(X, y[, sample_weight])	Return accuracy on provided data and labels.
set_callbacks(*callbacks)	Set callbacks for the estimator.
set_fit_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `fit` method.
set_params(**params)	Set the parameters of this estimator.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.
sparsify()	Convert coefficient matrix to sparse format.

Attributes

label
name

decision_function(*X*)

Predict confidence scores for samples.

The confidence score for a sample is proportional to the signed distance of that sample to the hyperplane.

Parameters

X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The data matrix for which we want to get the confidence scores.

Returns

scores (*ndarray of shape (n_samples,) or (n_samples, n_classes)*) – Confidence scores per *(n_samples, n_classes)* combination. In the binary case, confidence score for *self.classes_[1]* where >0 means this class would be predicted.

densify()

Convert coefficient matrix to dense array format.

Converts the `coef_` member (back) to a numpy.ndarray. This is the default format of `coef_` and is required for fitting, so calling this method is only required on models that have previously been sparsified; otherwise, it is a no-op.

Returns

self – Fitted estimator.

fit(*X*, *y*, *sample_weight=None*)

Fit the model according to the given training data.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training vector, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*array-like of shape (n_samples,)*) – Target vector relative to X.
- **sample_weight** (*array-like of shape (n_samples,) default=None*) – Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

  Added in version 0.17: *sample_weight* support to LogisticRegression.

Returns
self – Fitted estimator.

Notes

The SAGA solver supports both float64 and float32 bit arrays.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Logistic regression'

name = 'logistic'

predict(*X*)

Predict class labels for samples in X.

Parameters
X (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The data matrix for which we want to get the predictions.

Returns
y_pred (*ndarray of shape (n_samples,)*) – Vector containing the class labels for each sample.

predict_log_proba(*X*)

Predict logarithm of probability estimates.

The returned estimates for all classes are ordered by the label of classes.

Parameters
X (*array-like of shape (n_samples, n_features)*) – Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns
T (*array-like of shape (n_samples, n_classes)*) – Returns the log-probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

`predict_proba`(*X*)

Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multiclass / multinomial problem the softmax function is used to find the predicted probability of each class.

Parameters
X (*array-like of shape (n_samples, n_features)*) – Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns
T (*array-like of shape (n_samples, n_classes)*) – Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

`score`(*X*, *y*, *sample_weight=None*)

Return accuracy on provided data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns
score (*float*) – Mean accuracy of `self.predict(X)` w.r.t. *y*.

`set_callbacks`(**callbacks*)

Set callbacks for the estimator.

Parameters
***callbacks** (*callback instances*) – The callbacks to set.

Returns
self (*estimator instance*) – The estimator instance itself.

`set_fit_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *Logistic*

Configure whether metadata should be requested to be passed to the `fit` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.

- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `fit`.

Returns
self (*object*) – The updated object.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

set_score_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *Logistic*

Configure whether metadata should be requested to be passed to the `score` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self (*object*) – The updated object.

sparsify()

Convert coefficient matrix to sparse format.

Converts the `coef_` member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation.

The `intercept_` member is not converted.

Warning

This method is not supported for estimators fitted with array API inputs (i.e. when `sklearn.config_context()` is used with `array_api_dispatch=True`). The call may succeed but subsequent calls to *predict()* and other methods involving passing arrays may raise or return unexpected results.

Returns

self – Fitted estimator.

Notes

For non-sparse models, i.e. when there are not many zeros in `coef_`, this may actually *increase* memory usage, so use this method with care. A rule of thumb is that the number of zero elements, which can be computed with `(coef_ == 0).sum()`, must be more than 50% for this to provide significant benefits.

After calling this method, further fitting with the partial_fit method (if any) will not work until you call densify.

asreview.models.feature_extractors

Classes

Tfidf([columns, sep, lowercase, stop_words, ...])	TF-IDF feature extraction.
OneHot([columns, sep, lowercase, ...])	One-hot feature extraction.

asreview.models.feature_extractors.Tfidf

class asreview.models.feature_extractors.**Tfidf**(*columns=['title', 'abstract']*, *sep=' '*, *lowercase=True*, *stop_words=None*, *token_pattern='(?u)\\b\\w\\w+\\b'*, *ngram_range=(1, 1)*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *norm='l2'*, *use_idf=True*, *smooth_idf=True*, *sublinear_tf=False*, ***kwargs*)

Bases: `Pipeline`

TF-IDF feature extraction.

Based on the sklearn implementation of the TF-IDF feature extraction sklearn.feature_extraction.text.TfidfVectorizer.

Parameters

- **columns** (*list, default=["title", "abstract"]*) – See TextMerger
- **sep** (*str, default=" "*) – See TextMerger

- **lowercase** (*bool, default=True*) – See ScikitLearn CountVectorizer
- **stop_words** (*{'english'} or list or None, default=None*) – See ScikitLearn CountVectorizer
- **token_pattern** (*str or None, default=r"(?u)\b\w\w+\b"*) – See ScikitLearn CountVectorizer
- **ngram_range** (*tuple (min_n, max_n), default=(1,1)*) – See ScikitLearn CountVectorizer
- **max_df** (*float in range [0.0, 1.0] or int, default=1.0*) – See ScikitLearn CountVectorizer
- **min_df** (*float in range [0.0, 1.0] or int, default=1*) – See ScikitLearn CountVectorizer
- **max_features** (*int, default=None*) – See ScikitLearn CountVectorizer
- **vocabulary** (*Mapping or iterable, default=None*) – See ScikitLearn CountVectorizer
- **binary** (*bool, default=False*) – See ScikitLearn CountVectorizer
- **norm** (*{"l1", "l2"} or None, default="l2"*) – See ScikitLearn CountVectorizer
- **use_idf** (*bool, default=True*) – See ScikitLearn CountVectorizer
- **smooth_idf** (*bool, default=True*) – See ScikitLearn CountVectorizer
- **sublinear_tf** (*bool, default=False*) – See ScikitLearn CountVectorizer
- ****kwargs** (*dict*) – See ScikitLearn CountVectorizer for additional parameters

Methods

`__init__`([columns, sep, lowercase, ...])	
decision_function(X, **params)	Transform the data, and apply *decision_function* with the final estimator.
fit(X[, y])	Fit the model.
fit_predict(X[, y])	Transform the data, and apply *fit_predict* with the final estimator.
fit_transform(X[, y])	Fit the model and transform with the final estimator.
get_feature_names_out([input_features])	Get output feature names for transformation.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
inverse_transform(X, **params)	Apply *inverse_transform* for each step in a reverse order.
predict(X, **params)	Transform the data, and apply *predict* with the final estimator.
predict_log_proba(X, **params)	Transform the data, and apply *predict_log_proba* with the final estimator.
predict_proba(X, **params)	Transform the data, and apply *predict_proba* with the final estimator.
score(X[, y, sample_weight])	Transform the data, and apply *score* with the final estimator.
score_samples(X)	Transform the data, and apply *score_samples* with the final estimator.
set_callbacks(*callbacks)	Set callbacks for the estimator.

continues on next page

Table 51 – continued from previous page

set_output(*[, transform])	Set the output container when *"transform"* and *"fit_transform"* are called.
set_params(**kwargs)	Set the parameters of this estimator.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.
transform(X, **params)	Transform the data, and apply *transform* with the final estimator.

Attributes

*classes_*	The classes labels.
*feature_names_in_*	Names of features seen during first step *fit* method.
label	
*n_features_in_*	Number of features seen during first step *fit* method.
name	
named_steps	Access the steps by name.

property classes_

The classes labels. Only exist if the last step is a classifier.

decision_function(*X*, ***params*)

Transform the data, and apply *decision_function* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *decision_function* method. Only valid if the final estimator implements *decision_function*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of string -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

y_score (*ndarray of shape (n_samples, n_classes)*) – Result of calling *decision_function* on the final estimator.

property feature_names_in_

Names of features seen during first step *fit* method.

fit(*X*, *y=None*, ***params*)

Fit the model.

Fit all the transformers one after the other and sequentially transform the data. Finally, fit the transformed data using the final estimator.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.

- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters passed to the `fit` method of each step, where each parameter name is prefixed such that parameter `p` for step `s` has key `s__p`.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True* is set via `set_config()`.

  See Metadata Routing User Guide for more details.

Returns
self (*object*) – Pipeline with fitted steps.

fit_predict(*X*, *y=None*, ***params*)

Transform the data, and apply *fit_predict* with the final estimator.

Call *fit_transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *fit_predict* method. Only valid if the final estimator implements *fit_predict*.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the `predict` called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

  Note that while this may be used to return uncertainties from some models with `return_std` or `return_cov`, uncertainties that are generated by the transformations in the pipeline are not propagated to the final estimator.

Returns
y_pred (*ndarray*) – Result of calling *fit_predict* on the final estimator.

fit_transform(*X*, *y=None*, ***params*)

Fit the model and transform with the final estimator.

Fit all the transformers one after the other and sequentially transform the data. Only valid if the final estimator either implements *fit_transform* or *fit* and *transform*.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.

- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters passed to the `fit` method of each step, where each parameter name is prefixed such that parameter `p` for step `s` has key `s__p`.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns
Xt (*ndarray of shape (n_samples, n_transformed_features)*) – Transformed samples.

get_feature_names_out(*input_features=None*)

Get output feature names for transformation.

Transform input features using the pipeline.

Parameters
input_features (*array-like of str or None, default=None*) – Input features.

Returns
feature_names_out (*ndarray of str objects*) – Transformed feature names.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRouter*) – A `MetadataRouter` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Returns the parameters given in the constructor as well as the estimators contained within the *steps* of the *Pipeline*.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*mapping of string to any*) – Parameter names mapped to their values.

inverse_transform(*X*, ***params*)

Apply *inverse_transform* for each step in a reverse order.

All estimators in the pipeline must support *inverse_transform*.

Parameters

- **X** (*array-like of shape (n_samples, n_transformed_features)*) – Data samples, where `n_samples` is the number of samples and `n_features` is the number of features. Must fulfill input requirements of last step of pipeline's `inverse_transform` method.
- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

X_original (*ndarray of shape (n_samples, n_features)*) – Inverse transformed data, that is, data in the original feature space.

`label = 'TF-IDF'`

`property n_features_in_`

Number of features seen during first step *fit* method.

`name = 'tfidf'`

`property named_steps`

Access the steps by name.

Read-only attribute to access any step by given name. Keys are steps names and values are the steps objects.

`predict`(*X*, ***params*)

Transform the data, and apply *predict* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict* method. Only valid if the final estimator implements *predict*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the `predict` called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True* is set via `set_config()`.

  See Metadata Routing User Guide for more details.

  Note that while this may be used to return uncertainties from some models with `return_std` or `return_cov`, uncertainties that are generated by the transformations in the pipeline are not propagated to the final estimator.

Returns

y_pred (*ndarray*) – Result of calling *predict* on the final estimator.

`predict_log_proba`(*X*, ***params*)

Transform the data, and apply *predict_log_proba* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict_log_proba* method. Only valid if the final estimator implements *predict_log_proba*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the *predict_log_proba* called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns

y_log_proba (*ndarray of shape (n_samples, n_classes)*) – Result of calling *predict_log_proba* on the final estimator.

predict_proba(*X*, ***params*)

Transform the data, and apply *predict_proba* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict_proba* method. Only valid if the final estimator implements *predict_proba*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the *predict_proba* called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns

y_proba (*ndarray of shape (n_samples, n_classes)*) – Result of calling *predict_proba* on the final estimator.

score(*X*, *y=None*, *sample_weight=None*, ***params*)

Transform the data, and apply *score* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *score* method. Only valid if the final estimator implements *score*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Targets used for scoring. Must fulfill label requirements for all steps of the pipeline.
- **sample_weight** (*array-like, default=None*) – If not None, this argument is passed as `sample_weight` keyword argument to the `score` method of the final estimator.

- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns
score (*float*) – Result of calling *score* on the final estimator.

score_samples(*X*)

Transform the data, and apply *score_samples* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *score_samples* method. Only valid if the final estimator implements *score_samples*.

Parameters
X (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

Returns
y_score (*ndarray of shape (n_samples,)*) – Result of calling *score_samples* on the final estimator.

set_callbacks(**callbacks*)

Set callbacks for the estimator.

Parameters
***callbacks** (*callback instances*) – The callbacks to set.

Returns
self (*estimator instance*) – The estimator instance itself.

set_output(*, *transform=None*)

Set the output container when *"transform"* and *"fit_transform"* are called.

Calling *set_output* will set the output of all estimators in *steps*.

Parameters
transform (*{"default", "pandas", "polars"}, default=None*) – Configure output of *transform* and *fit_transform*.

- *"default"*: Default output format of a transformer
- *"pandas"*: DataFrame output
- *"polars"*: Polars output
- *None*: Transform configuration is unchanged

Added in version 1.4: *"polars"* option was added.

Returns
self (*estimator instance*) – Estimator instance.

set_params(***kwargs*)

Set the parameters of this estimator.

Valid parameter keys can be listed with `get_params()`. Note that you can directly set the parameters of the estimators contained in *steps*.

Parameters
****kwargs** (*dict*) – Parameters of this estimator or parameters of estimators contained in *steps*. Parameters of the steps may be set using its name and the parameter name separated by a '__'.

Returns
self (*object*) – Pipeline class instance.

`set_score_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *Tfidf*

Configure whether metadata should be requested to be passed to the `score` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self (*object*) – The updated object.

`transform`(*X*, ***params*)

Transform the data, and apply *transform* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *transform* method. Only valid if the final estimator implements *transform*.

This also works where final estimator is *None* in which case all prior transformations are applied.

Parameters

- **X** (*iterable*) – Data to transform. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns
Xt (*ndarray of shape (n_samples, n_transformed_features)*) – Transformed data.

asreview.models.feature_extractors.OneHot

class asreview.models.feature_extractors.**OneHot**(*columns=['title', 'abstract']*, *sep=' '*, *lowercase=True*, *stop_words=None*, *token_pattern='(?u)\\b\\w\\w+\\b'*, *ngram_range=(1, 1)*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, ***kwargs*)

Bases: Pipeline

One-hot feature extraction.

Based on the sklearn implementation of the one-hot feature extraction sklearn.feature_extraction.text.CountVectorizer with binary=True.

Parameters

- **columns** (*list, default=["title", "abstract"]*) – See TextMerger
- **sep** (*str, default=" "*) – See TextMerger
- **lowercase** (*bool, default=True*) – See ScikitLearn CountVectorizer
- **stop_words** (*{'english'} or list or None, default=None*) – See ScikitLearn CountVectorizer
- **token_pattern** (*str or None, default=r"(?u)\bww+\b"*) – See ScikitLearn CountVectorizer
- **ngram_range** (*tuple (min_n, max_n), default=(1,1)*) – See ScikitLearn CountVectorizer
- **max_df** (*float in range [0.0, 1.0] or int, default=1.0*) – See ScikitLearn CountVectorizer
- **min_df** (*float in range [0.0, 1.0] or int, default=1*) – See ScikitLearn CountVectorizer
- **max_features** (*int, default=None*) – See ScikitLearn CountVectorizer
- **vocabulary** (*Mapping or iterable, default=None*) – See ScikitLearn CountVectorizer
- ****kwargs** (*dict*) – See ScikitLearn CountVectorizer for additional parameters

Methods

__init__([columns, sep, lowercase, ...])	
decision_function(X, **params)	Transform the data, and apply *decision_function* with the final estimator.
fit(X[, y])	Fit the model.
fit_predict(X[, y])	Transform the data, and apply *fit_predict* with the final estimator.
fit_transform(X[, y])	Fit the model and transform with the final estimator.
get_feature_names_out([input_features])	Get output feature names for transformation.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
inverse_transform(X, **params)	Apply *inverse_transform* for each step in a reverse order.
predict(X, **params)	Transform the data, and apply *predict* with the final estimator.

continues on next page

Table 53 – continued from previous page

predict_log_proba(X, **params)	Transform the data, and apply *predict_log_proba* with the final estimator.
predict_proba(X, **params)	Transform the data, and apply *predict_proba* with the final estimator.
score(X[, y, sample_weight])	Transform the data, and apply *score* with the final estimator.
score_samples(X)	Transform the data, and apply *score_samples* with the final estimator.
set_callbacks(*callbacks)	Set callbacks for the estimator.
set_output(*[, transform])	Set the output container when *"transform"* and *"fit_transform"* are called.
set_params(**kwargs)	Set the parameters of this estimator.
set_score_request(*[, sample_weight])	Configure whether metadata should be requested to be passed to the `score` method.
transform(X, **params)	Transform the data, and apply *transform* with the final estimator.

Attributes

*classes_*	The classes labels.
*feature_names_in_*	Names of features seen during first step *fit* method.
label	
*n_features_in_*	Number of features seen during first step *fit* method.
name	
named_steps	Access the steps by name.

property classes_

The classes labels. Only exist if the last step is a classifier.

decision_function(*X*, ***params*)

Transform the data, and apply *decision_function* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *decision_function* method. Only valid if the final estimator implements *decision_function*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of string -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

y_score (*ndarray of shape (n_samples, n_classes)*) – Result of calling *decision_function* on the final estimator.

property feature_names_in_

Names of features seen during first step *fit* method.

fit(*X*, *y=None*, ***params*)

Fit the model.

Fit all the transformers one after the other and sequentially transform the data. Finally, fit the transformed data using the final estimator.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters passed to the `fit` method of each step, where each parameter name is prefixed such that parameter `p` for step `s` has key `s__p`.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True* is set via `set_config()`.

  See Metadata Routing User Guide for more details.

Returns

self (*object*) – Pipeline with fitted steps.

fit_predict(*X*, *y=None*, ***params*)

Transform the data, and apply *fit_predict* with the final estimator.

Call *fit_transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *fit_predict* method. Only valid if the final estimator implements *fit_predict*.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the `predict` called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

  Note that while this may be used to return uncertainties from some models with `return_std` or `return_cov`, uncertainties that are generated by the transformations in the pipeline are not propagated to the final estimator.

Returns

y_pred (*ndarray*) – Result of calling *fit_predict* on the final estimator.

fit_transform(*X*, *y=None*, ***params*)

Fit the model and transform with the final estimator.

Fit all the transformers one after the other and sequentially transform the data. Only valid if the final estimator either implements *fit_transform* or *fit* and *transform*.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters passed to the `fit` method of each step, where each parameter name is prefixed such that parameter `p` for step `s` has key `s__p`.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns

Xt (*ndarray of shape (n_samples, n_transformed_features)*) – Transformed samples.

get_feature_names_out(*input_features=None*)

Get output feature names for transformation.

Transform input features using the pipeline.

Parameters

input_features (*array-like of str or None, default=None*) – Input features.

Returns

feature_names_out (*ndarray of str objects*) – Transformed feature names.

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRouter*) – A `MetadataRouter` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Returns the parameters given in the constructor as well as the estimators contained within the *steps* of the *Pipeline*.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*mapping of string to any*) – Parameter names mapped to their values.

inverse_transform(*X*, ***params*)

Apply *inverse_transform* for each step in a reverse order.

All estimators in the pipeline must support *inverse_transform*.

Parameters

- **X** (*array-like of shape (n_samples, n_transformed_features)*) – Data samples, where `n_samples` is the number of samples and `n_features` is the number of features. Must fulfill input requirements of last step of pipeline's `inverse_transform` method.
- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

X_original (*ndarray of shape (n_samples, n_features)*) – Inverse transformed data, that is, data in the original feature space.

label = 'OneHot'

property n_features_in_

Number of features seen during first step *fit* method.

name = 'onehot'

property named_steps

Access the steps by name.

Read-only attribute to access any step by given name. Keys are steps names and values are the steps objects.

predict(*X*, ***params*)

Transform the data, and apply *predict* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict* method. Only valid if the final estimator implements *predict*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the `predict` called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True* is set via `set_config()`.

  See Metadata Routing User Guide for more details.

Note that while this may be used to return uncertainties from some models with `return_std` or `return_cov`, uncertainties that are generated by the transformations in the pipeline are not propagated to the final estimator.

Returns

y_pred (*ndarray*) – Result of calling *predict* on the final estimator.

`predict_log_proba`(*X*, ***params*)

Transform the data, and apply *predict_log_proba* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict_log_proba* method. Only valid if the final estimator implements *predict_log_proba*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the *predict_log_proba* called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns

y_log_proba (*ndarray of shape (n_samples, n_classes)*) – Result of calling *predict_log_proba* on the final estimator.

`predict_proba`(*X*, ***params*)

Transform the data, and apply *predict_proba* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict_proba* method. Only valid if the final estimator implements *predict_proba*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) –
  - If *enable_metadata_routing=False* (default): Parameters to the *predict_proba* called at the end of all transformations in the pipeline.
  - If *enable_metadata_routing=True*: Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 0.20.

  Changed in version 1.4: Parameters are now passed to the `transform` method of the intermediate steps as well, if requested, and if *enable_metadata_routing=True*.

  See Metadata Routing User Guide for more details.

Returns

y_proba (*ndarray of shape (n_samples, n_classes)*) – Result of calling *predict_proba* on the final estimator.

`score`(*X*, *y=None*, *sample_weight=None*, ***params*)

Transform the data, and apply *score* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *score* method. Only valid if the final estimator implements *score*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable, default=None*) – Targets used for scoring. Must fulfill label requirements for all steps of the pipeline.
- **sample_weight** (*array-like, default=None*) – If not None, this argument is passed as `sample_weight` keyword argument to the `score` method of the final estimator.
- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

score (*float*) – Result of calling *score* on the final estimator.

`score_samples`(*X*)

Transform the data, and apply *score_samples* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *score_samples* method. Only valid if the final estimator implements *score_samples*.

Parameters

X (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

Returns

y_score (*ndarray of shape (n_samples,)*) – Result of calling *score_samples* on the final estimator.

`set_callbacks`(**callbacks*)

Set callbacks for the estimator.

Parameters

***callbacks** (*callback instances*) – The callbacks to set.

Returns

self (*estimator instance*) – The estimator instance itself.

`set_output`(***, *transform=None*)

Set the output container when *"transform"* and *"fit_transform"* are called.

Calling *set_output* will set the output of all estimators in *steps*.

Parameters

transform (*{"default", "pandas", "polars"}, default=None*) – Configure output of *transform* and *fit_transform*.

- *"default"*: Default output format of a transformer

- *"pandas"*: DataFrame output
- *"polars"*: Polars output
- *None*: Transform configuration is unchanged

Added in version 1.4: *"polars"* option was added.

Returns
self (*estimator instance*) – Estimator instance.

`set_params`(***kwargs*)

Set the parameters of this estimator.

Valid parameter keys can be listed with `get_params()`. Note that you can directly set the parameters of the estimators contained in *steps*.

Parameters
****kwargs** (*dict*) – Parameters of this estimator or parameters of estimators contained in *steps*. Parameters of the steps may be set using its name and the parameter name separated by a '__'.

Returns
self (*object*) – Pipeline class instance.

`set_score_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *OneHot*

Configure whether metadata should be requested to be passed to the `score` method.

Note that this method is only relevant when this estimator is used as a sub-estimator within a meta-estimator and metadata routing is enabled with `enable_metadata_routing=True` (see `sklearn.set_config()`). Please check the User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self (*object*) – The updated object.

`transform`(*X*, ***params*)

Transform the data, and apply *transform* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *transform* method. Only valid if the final estimator implements *transform*.

This also works where final estimator is *None* in which case all prior transformations are applied.

Parameters

- **X** (*iterable*) – Data to transform. Must fulfill input requirements of first step of the pipeline.
- ****params** (*dict of str -> object*) – Parameters requested and accepted by steps. Each step must have requested certain metadata for these parameters to be forwarded to them.

  Added in version 1.4: Only available if *enable_metadata_routing=True*. See Metadata Routing User Guide for more details.

Returns

Xt (*ndarray of shape (n_samples, n_transformed_features)*) – Transformed data.

asreview.models.queriers

Classes

Max()	Maximum query strategy.
HybridMaxRandom([probability, random_state])	95% Maximum and 5% Random query strategy.
HybridMaxUncertainty([probability, u, ...])	95% Maximum and 5% Uncertainty query strategy.
Uncertainty([u, proba])	Uncertainty query strategy.
Random([random_state])	Random query strategy.
TopDown()	Top-down query strategy.

asreview.models.queriers.Max

class asreview.models.queriers.**Max**

Bases: QueryMixin, BaseEstimator

Maximum query strategy.

Choose the most likely samples to be included according to the model.

Methods

__init__()	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Rank the instances of the feature matrix.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

`get_params`(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

`label = 'Maximum'`

`name = 'max'`

`query`(*p*)

Rank the instances of the feature matrix.

Parameters
p (*numpy.ndarray*) – The probability of inclusion for each record in the feature matrix.

Returns
numpy.ndarray – The QueryStrategy ranks the row numbers of the feature matrix. It returns an array of shape (len(X),) containing the row indices in ranked order.

`set_params`(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

asreview.models.queriers.HybridMaxRandom

`class asreview.models.queriers.HybridMaxRandom`(*probability=0.95*, *random_state=None*)

Bases: `QueryMixin`, `BaseEstimator`

95% Maximum and 5% Random query strategy.

A mix of maximum and random query strategies with a mix ratio of 0.95. At each query 95% of the instances would be sampled with the maximum query strategy after which the remaining 5% would be sampled with the random query strategy.

Parameters

- **probability** (*float*) – The probability of sampling with the maximum query strategy.
- **random_state** (*int, RandomState*) – Random

Methods

`__init__`([probability, random_state])

continues on next page

Table 58 – continued from previous page

get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Rank the instances of the feature matrix.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Mixed (95% Maximum and 5% Random)'

name = 'max_random'

query(*p*)

Rank the instances of the feature matrix.

Parameters
p (*numpy.ndarray*) – The probability of inclusion for each record in the feature matrix.

Returns
numpy.ndarray – The QueryStrategy ranks the row numbers of the feature matrix. It returns an array of shape (len(X),) containing the row indices in ranked order.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

asreview.models.queriers.HybridMaxUncertainty

class asreview.models.queriers.**HybridMaxUncertainty**(*probability=0.95*, *u=None*, *proba=None*, *random_state=None*)

Bases: QueryMixin, BaseEstimator

95% Maximum and 5% Uncertainty query strategy.

A mix of maximum and random query strategies with a mix ratio of 0.95. At each query 95% of the instances would be sampled with the maximum query strategy after which the remaining 5% would be sampled with the uncertainty query strategy.

Methods

__init__([probability, u, proba, random_state])	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Rank the instances of the feature matrix.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*dict*) – Parameter names mapped to their values.

label = 'Mixed (95% Maximum and 5% Uncertainty)'

name = 'max_uncertainty'

query(*p*)

Rank the instances of the feature matrix.

Parameters

p (*numpy.ndarray*) – The probability of inclusion for each record in the feature matrix.

Returns

numpy.ndarray – The QueryStrategy ranks the row numbers of the feature matrix. It returns an array of shape (len(X),) containing the row indices in ranked order.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

asreview.models.queriers.Uncertainty

class asreview.models.queriers.**Uncertainty**(*u: float = None*, *proba: bool = True*)

Bases: QueryMixin, BaseEstimator

Uncertainty query strategy.

Choose the most uncertain samples according to the model (i.e. closest to 0.5 probability). If decision functions are used, the samples closest to the decision boundary are chosen (0).

Methods

__init__([u, proba])	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Query instances.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Uncertainty'

name = 'uncertainty'

query(*p*)

Query instances.

Parameters
p (*np.array*) – The probabilities of the instances.

Returns
np.array – The indices of the instances to be queried.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

asreview.models.queriers.Random

class asreview.models.queriers.**Random**(*random_state=None*)

Bases: QueryMixin, BaseEstimator

Random query strategy.

Choose the samples to be included at random.

Methods

__init__([random_state])	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Query instances.
set_params(**params)	Set the parameters of this estimator.

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

`label = 'Random'`

`name = 'random'`

`query`(*p*)

Query instances.

Parameters

- **p** (*np.array*) – The probabilities of the instances.
- **random_state** (*int, RandomState*) – Random state for shuffling the indices.

Returns
np.array – The indices of the instances to be queried.

`set_params`(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

asreview.models.queriers.TopDown

`class asreview.models.queriers.TopDown`

Bases: `QueryMixin`, `BaseEstimator`

Top-down query strategy.

Query the records in a top-down fashion.

Methods

`__init__`()	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
query(p)	Query instances.
set_params(**params)	Set the parameters of this estimator.

Attributes

label

continues on next page

Table 67 – continued from previous page

name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params (*dict*) – Parameter names mapped to their values.

label = 'Top-down'

name = 'top_down'

query(*p*)

Query instances.

Parameters

p (*np.array*) – The probabilities of the instances.

Returns

np.array – The indices of the instances to be queried.

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self (*estimator instance*) – Estimator instance.

asreview.models.stoppers

Stopper mechanisms for the review process.

The stopper mechanisms determine when the review process should be stopped. This can be based on the properties of the results table or the input dataset.

Warning

This module is experimental and might change.

Classes

LastRelevant()	Stop after last relevant record.
NLabeled(n)	Stop the review after n have been labeled.
QuantileLabeled(quantile)	Stop the review after a certain quantile of the records have been labeled.
IsFittable()	Stop the review after both classes are found.
NConsecutiveIrrelevant(n)	Stop the review after n irrelevant records have been labeled in a row.

asreview.models.stoppers.LastRelevant

class asreview.models.stoppers.**LastRelevant**

Bases: BaseEstimator

Stop after last relevant record.

The stopping mechanism stops the review when all records have been labeled.

Parameters
value (*int, str*) – Number of labels to stop the review at. If set to "min", the review will stop when all relevant records are found.

Methods

__init__()	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
stop(results, data)	

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Last Relevant'

name = 'last_relevant'

set_params(**params)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self (*estimator instance*) – Estimator instance.

stop(*results*, *data*)

asreview.models.stoppers.NLabeled

class asreview.models.stoppers.**NLabeled**(*n*)

Bases: BaseEstimator

Stop the review after n have been labeled.

Parameters

n (*int, tuple*) – Number of labels to stop the review at. If tuple, the first element is the number of relevant records to find, the second element is the number of irrelevant records to find.

Methods

__init__(n)	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
stop(results, data)	

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns

routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and

contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

`label = 'N Labeled'`

`name = 'n_labeled'`

`set_params`(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

`stop`(*results*, *data*)

asreview.models.stoppers.QuantileLabeled

`class asreview.models.stoppers.QuantileLabeled`(*quantile*)

Bases: `BaseEstimator`

Stop the review after a certain quantile of the records have been labeled.

Parameters
quantile (*float*) – Quantile of records to label before stopping the review.

Methods

`__init__`(quantile)	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
stop(results, data)	

Attributes

label
name

`get_metadata_routing`()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

`get_params`(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Quantile Labeled'

name = 'q_labeled'

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

stop(*results*, *data*)

asreview.models.stoppers.IsFittable

class asreview.models.stoppers.**IsFittable**

Bases: *NLabeled*

Stop the review after both classes are found.

Methods

__init__()	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
stop(results, data)	

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A MetadataRequest encapsulating routing information.

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

label = 'Is Fittable'

name = 'is_fittable'

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

stop(*results*, *data*)

asreview.models.stoppers.NConsecutiveIrrelevant

class asreview.models.stoppers.**NConsecutiveIrrelevant**(*n*)

Bases: BaseEstimator

Stop the review after n irrelevant records have been labeled in a row.

Parameters
n (*int*) – Number of irrelevant records in a row to stop the review at.

Methods

__init__(n)	
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
stop(results, data)	

Attributes

label
name

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing (*MetadataRequest*) – A `MetadataRequest` encapsulating routing information.

`get_params`(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params (*dict*) – Parameter names mapped to their values.

`label = 'N Consecutive Irrelevant'`

`name = 'n_consecutive_irrelevant'`

`set_params`(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self (*estimator instance*) – Estimator instance.

`stop`(*results*, *data*)

asreview.metrics

Performance metrics for activate learning results.

Functions

loss(labels)	Compute the loss of the labels.
ndcg(labels)	Compute the Normalized Discounted Cumulative Gain (NDCG)

asreview.metrics.loss

`asreview.metrics.loss`(*labels: list[int]*)

Compute the loss of the labels.

Parameters
labels (*list*) – List of labels.

Returns
float – The loss of the labels.

asreview.metrics.ndcg

asreview.metrics.**ndcg**(*labels: list[int]*)

Compute the Normalized Discounted Cumulative Gain (NDCG)

Basesd on: https://doi.org/10.48550/arXiv.1304.6480

Parameters
labels (*list*) – List of binary labels (0 or 1).

Returns
float – The NDCG score.

asreview.datasets

Classes

BaseDataGroup(*datasets)	
BaseDataSet(dataset_id[, filepath, title, ...])	
DatasetManager()	
SynergyDataGroup()	Datasets available in the SYNERGY dataset.
SynergyDataSet(dataset_id[, filepath, ...])	

asreview.datasets.BaseDataGroup

class asreview.datasets.**BaseDataGroup**(**datasets*)

Bases: ABC

Methods

__init__(*datasets)	Group of datasets.
append(dataset)	Append dataset to group.
find(dataset_id)	Find dataset in the group.

Attributes

description
group_id
url

append(*dataset*)

Append dataset to group.

dataset: asreview.datasets.BaseDataSet
A asreview BaseDataSet-like object.

abstract property description

find(*dataset_id*)

Find dataset in the group.

Parameters
dataset_id (*str*) – Identifier of the dataset to look for. It can also be one of the aliases. Case insensitive.

Returns
asreview.datasets.BaseDataSet – Returns base dataset with the given dataset_id.

abstract property group_id

url = None

asreview.datasets.BaseDataSet

class asreview.datasets.**BaseDataSet**(*dataset_id*, *filepath=None*, *title=None*, *description=None*, *authors=None*, *topic=None*, *link=None*, *reference=None*, *img_url=None*, *license=None*, *year=None*, *aliases=None*, ***kwargs*)

Bases: object

Methods

__init__(dataset_id[, filepath, title, ...])	Base class for metadata of dataset.
to_file(path)	

Attributes

filename
reader

property filename

property reader

to_file(*path*)

asreview.datasets.DatasetManager

class asreview.datasets.**DatasetManager**

Bases: object

Methods

__init__()	
find(dataset_id)	Find a dataset.
list([include, exclude, serialize, ...])	List the available datasets.

Attributes

groups

find(*dataset_id*)

Find a dataset.

Parameters
dataset_id (*str, iterable*) – Look for this term in aliases within any dataset. A group

can be specified by setting dataset_id to 'group_id:dataset_id'. This can be helpful if the dataset_id is not unique. The dataset_id can also be a non-string iterable, in which case a list will be returned with all terms. Dataset_ids should not contain semicolons (:). Return None if the dataset could not be found.

Returns
BaseDataSet – Return the dataset with dataset_id.

property groups

list(*include=None*, *exclude=None*, *serialize=True*, *raise_on_error=False*)

List the available datasets.

Parameters

- **include** (*str, iterable*) – List of groups to include
- **exclude** (*str, iterable*) – List of groups to exclude from all groups.
- **serialize** (*bool*) – Make returned list serializable.
- **raise_on_error** (*bool*) – Raise error when entry point can't be loaded.

Returns
list – List with datasets as values.

asreview.datasets.SynergyDataGroup

class asreview.datasets.**SynergyDataGroup**

Bases: *BaseDataGroup*

Datasets available in the SYNERGY dataset.

Methods

__init__()	Group of datasets.
append(dataset)	Append dataset to group.
find(dataset_id)	Find dataset in the group.

Attributes

description
group_id
url

append(*dataset*)

Append dataset to group.

dataset: asreview.datasets.BaseDataSet
A asreview BaseDataSet-like object.

description = 'SYNERGY'

find(*dataset_id*)

Find dataset in the group.

Parameters

dataset_id (*str*) – Identifier of the dataset to look for. It can also be one of the aliases. Case insensitive.

Returns
asreview.datasets.BaseDataSet – Returns base dataset with the given dataset_id.

group_id = 'synergy'

url = 'https://asreview.ai/synergy'

asreview.datasets.SynergyDataSet

class asreview.datasets.**SynergyDataSet**(*dataset_id*, *filepath=None*, *title=None*, *description=None*, *authors=None*, *topic=None*, *link=None*, *reference=None*, *img_url=None*, *license=None*, *year=None*, *aliases=None*, ***kwargs*)

Bases: *BaseDataSet*

Methods

__init__(dataset_id[, filepath, title, ...])	Base class for metadata of dataset.
to_file([path])	

Attributes

filename
reader

property filename

property reader

to_file(*path=None*)

4.1.6 Indices and tables

- genindex
- modindex

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y