ASReview Software Documentation

ASReview Core Development Team, Utrecht University

Apr 28, 2021

Introduction

Welcome to the documentation of ASReview! It includes an *installation* guide, a *quick tour*, details how to prepare your *dataset* and much more - enjoy!

ASReview is open source research software developed by *researchers* at Utrecht University and is published under the Apache 2.0 licence.

About ASReview

Anyone who goes through the process of screening large amounts of texts knows how labor intensive this can be. The future will be an interaction with machine learning algorithms to deal with the enormous increase of available text. Therefore, an open source machine learning-aided pipeline applying active learning was developed at Utrecht University, titled **ASReview**. ASReview aims to help scholars and practitioners to get an overview of the most relevant records for their work as efficiently as possible, while being transparent in the process.

1.1 Research Team

ASReview is a research project coordinated by Rens van de Schoot (full Professor at the Department of Methodology & Statistics and ambassador of the focus area Applied Data Science at Utrecht University, The Netherlands), together with Jonathan de Bruin, Lead engineer of the ASReview project and working at the Information and Technology Services department at Utrecht University.

The advisory board consists of machine learning expert Daniel Oberski, associate professor at Utrecht University's Department of Methodology & Statistics, and the department of Biostatistics at the Julius Center, University Medical Center Utrecht), full professor Lars Tummers (Professor of Public Management and Behavior at Utrecht University), Ayoub Bagheri (NLP-expert at Utrecht University), Bianca Kramer (Open Science expert at the Utrecht University library), Jan de Boer (Information specialist at the Utrecht university library), Felix Weijdema (Systematic review specialist at the Utrecht University library), and Martijn Huijts (UX-expert at the department Test and Quality Services at Utrecht University).

The artwork for the ASReview project is created by Joukje Willemsen.

Moreover, many others contribute to the project, like researchers Gerbrich Ferdinands and Laura Hofstee, as well as many students like Yongchao Terry Ma, Sofie van den Brand, Sybren Hindriks, and Albert Harkema. See ASReview project members for an overview of the people involved in the project.

ASReview thanks the community for their contributions. A full list of code contributors can be found on the contributors' page on Github.
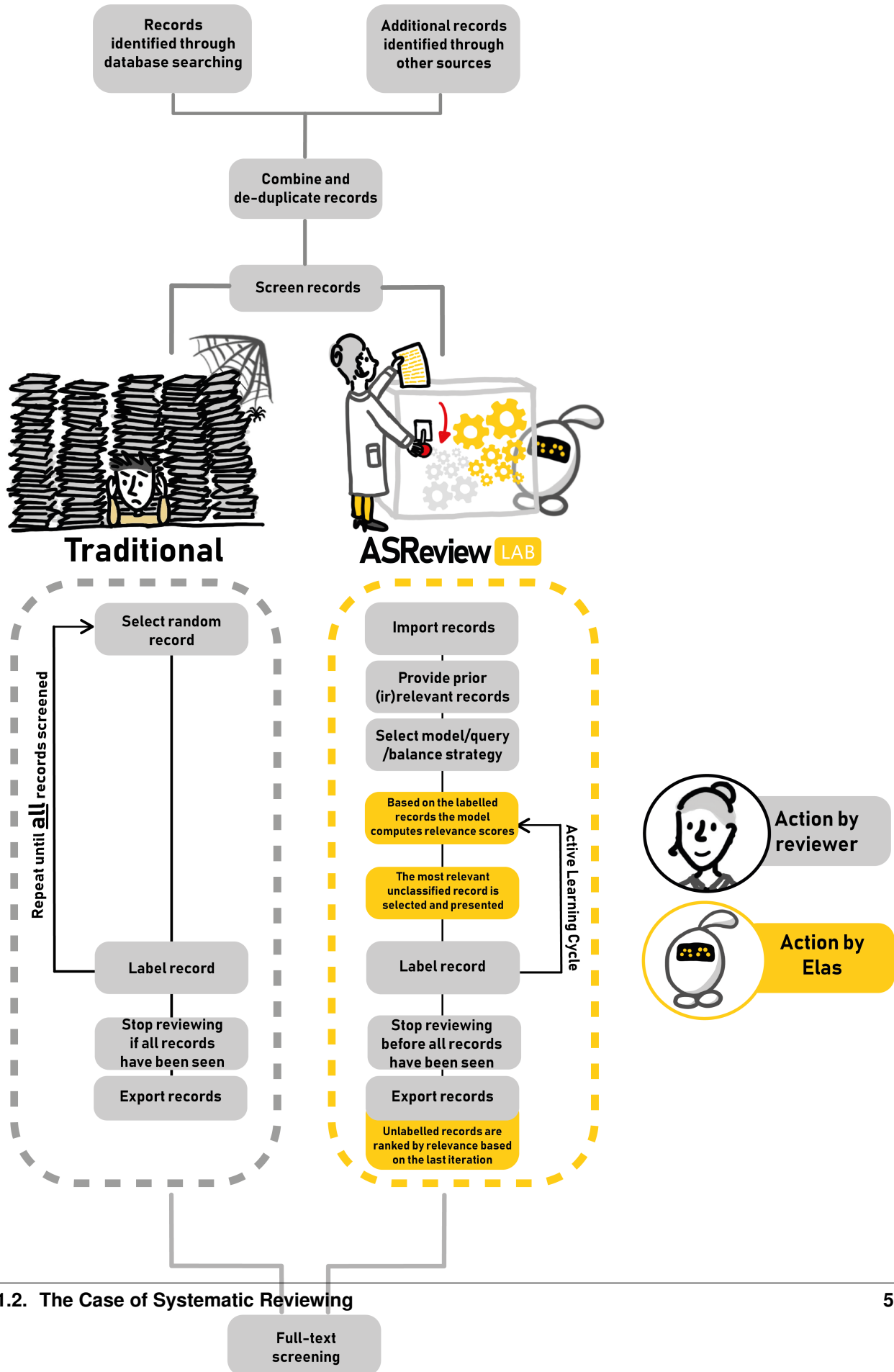
1.2 The Case of Systematic Reviewing

ASReview is a generic tool for the screening of *any type of text*, but often use the case of systematic reviewing is ued to illustrate its usefulness, see also the blog post ASReview Class 101.

With the emergence of online publishing, the number of scientific papers on any topic, e.g. COVID19, is skyrocketing. Simultaneously, the public press and social media also produce data by the second. All this textual data presents opportunities to scholars, but it also confronts them with new challenges. To summarize all this data, researchers write systematic reviews, providing essential, comprehensive overviews of relevant topics. To achieve this, they have to screen (tens of) thousands of studies by hand for inclusion in their overview. As truly relevant papers are very sparse (i.e., often <10%), this is an extremely imbalanced data problem. The process of finding these rare relevant papers is error prone and very time intensive.

The rapidly evolving field of machine learning (ML) has allowed the development of ML-aided pipelines that assist in finding relevant texts for such search tasks. A well-established approach to increase the efficiency of title and abstract screening is determining prioritization with *active learning*, which is very effective for *systematic reviewing*.

The goal of ASReview is to help scholars and practitioners to get an overview of the most relevant records for their work as efficiently as possible, while being transparent in the process. It is uses active learning, allows multiple ML-models, and ships with a benchmark mode which is especially useful for comparing and designing algorithms. Furthermore, it is intended to be easily extensible, allowing third parties to add modules that enhance the pipeline and can process any text (although we consider systematic reviewing as a very useful approach).

Records identified through database searching

Additional records identified through other sources

Combine and de-duplicate records

Screen records

Traditional

ASReview LAB

Repeat until <u>all</u> records screened

Select random record

Label record

Stop reviewing if all records have been seen

Export records

Import records

Provide prior (ir)relevant records

Select model/query /balance strategy

Based on the labelled records the model computes relevance scores

The most relevant unclassified record is selected and presented

Label record

Stop reviewing before all records have been seen

Export records

Unlabelled records are ranked by relevance based on the last iteration

Active Learning Cycle

Action by reviewer

Action by Elas

Full-text screening

The Zen of Elas

Elas is the mascotte of ASReview and your Electronic Learning Assistant who will guide you through the interactive process of making decisions using Artificial Intelligence in ASReview. Elas comes with some important principles:

Humans are the Oracle It's the interaction between humans and machines which will take science a major leap forward. We believe a human should make the final decision about whether to mark a record as relevant/irrelevant (hence, is the Oracle), and the software merely orders the records on the relevance score as predicted by the model in each iteration of the active learning cycle.

Open & Transparent We are strong proponents of open science, and therefore ASReview code is free and openly available. We value your privacy and hence do not get to see any of your data (everything stays on your device). We do hope you believe like us in the FAIR data principles and publish your data, results, and project file on a data repository.

Unbiasedness ASReview signed the DORA-declaration. ASReview only presents text for unbiased decision making. When screening, for example, academic papers, ASReview show titles and abstracts only, and does not present authors or journal names. This way, you can focus on what is truly important (the content) and don't get tempted to use irrelevant information.

AI-aided Interface Simplicity is the ultimate sophistication (Davinci), and, therefore, we keep the front-end as clean as possible. Boring but efficient because the *magic* happens under the hood.

Garbage in garbage out ASReview focuses on the machine learning part of the pipeline and not on the preprocessing or postprocessing of the data (which reference managers are designed for). Be aware of the principle GIGO and check the quality of your data first. Don't blame Elas if the performance is not as good as expected due to low-quality input data.

The Zen of ELAS

Unbiasedness

Open & Transparent

AI-aided Interface

Humans are the Oracle

Garbage in garbage out
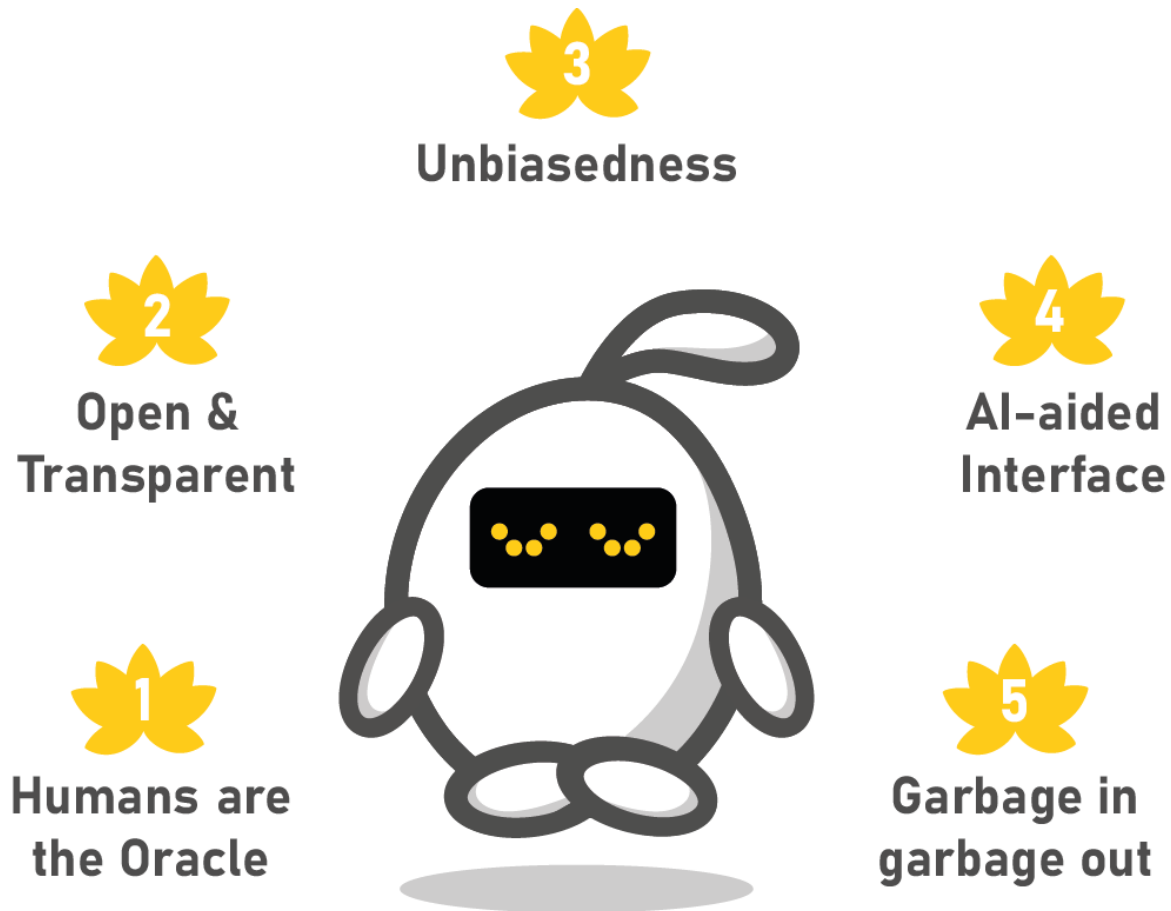
Installation

3.1 Install ASReview

ASReview software requires having Python 3.6 or higher installed. Detailed step-by-step instructions to install Python (and ASReview) are available for Windows and MacOS users.

Install the ASReview software with Pip by running the following command in the *CMD.exe* (Windows) or *Terminal* (MacOS/Linux):

```
pip install asreview
```

Start the application with the following command (in CMD.exe or Terminal):

```
asreview lab
```

You are now ready to start your first Automated Systematic Review!

See *Troubleshooting* for common problems.

3.2 Upgrade ASReview

Upgrade ASReview software with

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

3.3 Uninstall ASReview

Remove ASReview with

```
pip uninstall asreview
```

Enter y to confirm.

> **Warning:** Note that your project files will **not** delete with this action. You find them in the *.asreview* folder in your home folder.

3.4 Server Installation

It is possible to run the ASReview software on a server or custom domain. Use the flags *ip* and *port* for configuration. ASReview should only be used in closed networks.

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

> **Warning:** Don't use the development server in production. Read the Flask documentation about deploying a Flask app to production.

3.5 Install with Docker

For a quickstart of ASReview LAB using Docker and without the need to install anything else, the latest version of the ASReview LAB can be started as well via Docker like this:

```
docker run -p 5000:5000 asreview/asreview
```

This will start the ASReview LAB server on port 5000 with default command line options and make it accessible to the host at http://localhost:5000 More advanced command line options can be given afterwards, like this:

```
docker run -p 9000:9000 asreview/asreview  --port 9000
```

For more information, see the GitHub page.

3.6 Troubleshooting

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

3.6.1 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 with the following command:

```
python --version
```

If this doesn't return 3.6 or higher, then Python isn't or not correctly installed.

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

However, there is a simple way to deal with correct environment variables by ading *python -m* in front of the command. For example:

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

3.6.2 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 decription of the program, use *python -m* in front of all your commands. For example:

```
python -m asreview lab
```

3.6.3 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) are available for Windows and MacOS users.

3.6.4 Remove temporary files

In case ASReview runs into unexpected errors or doesn't work as expected, it is advised to try to remove temporary files from the project first. These files can be found in the `.asreview/` folder in your home directory. However, the easiest way to remove these files is with:

```
asreview lab --clean-all-projects
```

This will safely remove temporay files, nothing will harm your review. To clean a specific project, use

```
asreview lab --clean-project my-project
```

in which `my_project` is your project name.

CHAPTER 4

Prepare your Data

To perform a systematic review, ASReview requires a dataset representing all records (e.g., abstracts of scientific papers) obtained in a systematic search. To create such a dataset for a systematic review, typically an online library search is performed for all studies related to a particular topic.

It is possible to use your own dataset with unlabeled, partly labeled (where the labeled records are used for training a model for the unlabeled records), or fully labeled records (used for the Simulation mode). For testing and demonstrating ASReview (used for the Exploration mode), the software offers *Benchmark Datasets*. Also, a plugin with *Corona related publications* is available.

> **Warning:** If you upload your own data, make sure to remove duplicates and to retrieve as many abstracts as possible (don't know how?). With clean data you benefit most from what *active learning* has to offer.

4.1 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:

- **RIS file format** (wikipedia) with extensions `.ris` or `.txt`. RIS file formats 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.

- **Tabular datasets** with extensions `.csv`, `.tab`, `.tsv`, `.xlsx`, or `.xls`. 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
ID	record_id	no
Title	title, primary_title	yes*
Abstract	abstract, abstract note	yes*
Keywords	keywords	no
Authors	authors, author names, first_authors	no
DOI	doi	no
Included	final_included, label, label_included, included_label, included_final, included, included_flag, include	no
debug_label	debug_label	no

* Only a title or an abstract is mandatory.

ID If your data contains a column titled `record_id` it needs to consists only of integers, and it should contain no missing data and no duplicates, otherwise you will receive an error. If there is no `record_id` it will be autmtically generated by the software. This column can also be used for the Simulation Mode to select prior knowledge.

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 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. 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, and `1` = relevant/included. Different column names are allowed, see the table. The use is twofold:

- **Screening**: In ASReview LAB, if labels are available for a part of the dataset (see *partly labeled data*), 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.

- **Simulation**: In the *ASReview command line interface for simulations*, the column containing the labels is used to simulate a systematic review run. Only records containing labels are used for the simulation, unlabeled records are ignored.

Note: Files exported with ASReview LAB contain the column `included` and can be used for prior knowledge.

debug_label You can explore a existing fully labeled dataset in the Exploraton Mode. A column called named `debug_label` is required, indicating the relevant and irrelevant records with ones and zeroes. The relevant records will be displayed in green during screening. This option is useful for training purposes, presentations, and workshops.

4.2 Compatibility

4.2.1 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.

- • X = 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".

4.2.2 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
CINHAL (EBSCO)	X	N/A	X	N/A
Cochrane		N/A		N/A
Embase		N/A		
Eric (Ovid)	X	N/A	N/A	X
Psychinfo (Ovid)	X	N/A	N/A	X
Pubmed	X	N/A	X	N/A
Scopus		N/A		N/A
Web of Science	X	X	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.

Warning: 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.

4.2.3 Systematic Review Software

There are several software packages available for systematic reviewing, see for an overview. 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.

4.3 Benchmark Datasets

The ASReview software contains a large amount of benchmark datasets that can be used in the *exploration* or *simulation* mode. The labelled datasets are PRISMA-based reviews on various research topics, are available under an open licence and are automatically harvested from the dataset repository. See index.csv for all available properties.

4.3.1 Featured Datasets

Some featured datasets are:

- The *PTSD Trajectories* data by Van de Schoot et al. (2017, 2018) stems from a review of longitudinal studies that applied unsupervised machine learning techniques on longitudinal data of self-reported symptoms of posttraumatic stress assessed after trauma exposure. In total, 5,782 studies were obtained by searching Pubmed, Embase, PsychInfo, and Scopus, and through a snowballing strategy in which both the references and the citation of the included papers were screened. Thirty-eight studies were included in the review (0.66%).

- The *Virus Metagenomics* data by Kwok et al. (2020) which systematically described studies that performed viral Metagenomic Next-Generation Sequencing (mNGS) in common livestock such as cattle, small ruminants, poultry, and pigs.44 Studies were retrieved from Embase (n = 1,806), Medline (n = 1,384), Cochrane Central (n = 1), Web of Science (n = 977), and Google Scholar (n = 200, the top relevant references). After deduplication this led to 2,481 studies obtained in the initial search, of which 120 inclusions (4.84%).

- The *Software Fault Prediction* by Hall et al. (2012) stems from a systematic review of studies on fault prediction in software engineering. Studies were obtained from ACM Digital Library, IEEExplore and the ISI Web of Science. Additionally, a snowballing strategy and a manual search were conducted, accumulating to 8,911 publications of which 104 were included in the systematic review (1.2%).

- The *ACEinhibitors* by Cohen et al. (2006) data stems from a systematic review on the efficacy of Angiotensin-converting enzyme (ACE) inhibitors. The data is a subset of 2,544 publications from the TREC 2004 Genomics Track document corpus48. This is a static subset from all MEDLINE records from 1994 through 2003, which allows for replicability of results. Forty-one publications were included in the review (1.6%).

4.3.2 Results

For the featured datasets, the animated plots below show how fast you can find the relevant papers by using ASReview LAB compared to random screening papers one by one. These animated plots are all based on a single run per dataset in which only one paper was added as relevant and one as irrelevant.

PTSD Trajectories:

38 inclusions out of 5,782 papers

Virus Metagenomics:

120 inclusions out of 2,481 papers

Software Fault Prediction:

104 inclusions out of 8,911 papers

ACEinhibitors:

41 inclusions out of 2,544 papers

Cite, Donate and Contribute

5.1 Cite

The paper published in Nature Machine Intelligence can be used to cite the **ASReview project**.

For citing the software **ASReview Lab**, refer to the specific release of the software. The menu on the right (in Zenodo) can be used to find the citation format of prevalence.

For citing the documentation (or to download the pdf) go to Zenodo.

More studies related to the project can be found on the ASReview website.

5.2 Donate

The ASReview software is Free and Open Source Software (FOSS). To support the development, you can donate via the ASReview crowdfunding platform. Even small donations are highly appreciated!

5.3 Collaborate

Another option is to include budget for ASReview in your grant proposal. If you are interested in this option, contact Prof. Dr. Rens van de Schoot.

5.4 Contribute

If you discover issues please let us know via Github. If you have ideas to solve your own or other issues, it is highly appreciate to contribute to the development.

Frequently Asked Questions

The FAQ has moved to Github Discussions. The place to find answers to your questions regarding ASReview and to ask new questions!

Vocabulary

The ASReview project makes use of standardized terminology for all communication regarding ASReview and its underlying technology. An overview of terms and usage can be found in the table below.

Term	Usage
ASReview	Means "Active learning for Systematic Reviews" or "AI-assisted Systematic Reviews", depending on context. Avoid this explanation, only use as tagline.
ASReview project	Use ASReview project as an encompassing term for all work that is done by the ASReview team members and ASReview contributors.
ASReview LAB	Use to indicate the user-friendly interface that has been developed for researchers to use.
ASReview CLI	Use to indicate the command line interface that has been developed for advanced options or for running simulations studies.
team members	UU employees and students and who have permission to devote hours to the ASReview project.
contributors	Everyone contributing to the ASReview project (through GitHub)
Extension	Use to indicate additional elements to the ASReview software, such as the ASReview visualisation extension, or the ASReview CORD-19 extension.
ELAS	Our Electronic Learning ASsistent. Name of our mascot. Use for storytelling and to increase explainability.
User **Screener**	The human annotator who labels records. Replacement term when context is PRISMA-based reviewing.
Records	The data points that need to be labeled. The records can contain both information that is used for training the active learning model, and information that is not used for this purpose. In the case of systematic reviewing, a record is meta-data for a scientific publication. Here, the information that is used for training purposes is the text in the title and abstract of the publication. The information that is not used for training typically consists of other metadata, for example, the authors, journal, or DOI of the publication.
Screening **Labeling** **Reviewing** **Classifying**	All terms can be used to indicate the decision-making process on the relevancy of records ("irrelevant" or "relevant").
Active learning model	Use to indicate how the next record to be screened by the user is selected. The model consists of several elements: a query strategy, a feature extraction technique, a classifier, and a balance strategy.
Finished	Whenever **the user decides** that the reviewing process has been completed or if **all records are labeled**.
Published	Whenever the data and **ASReview project file** are openly published on, for example, the OSF.

CHAPTER 8

Overview

ASReview LAB is user-friendly software for exploring the future of AI in systematic reviews. The software implements an *Oracle Mode*, an *Exploration Mode*, and a *Simulation Mode*. The oracle mode provides a screening interface that can be used to perform an AI-aided systematic review with the interaction of the reviewer (the oracle). The simulation mode can be used to simulate the performance of ASReview on existing systematic reviews. The exploration mode can be used for exploring a fully labeled dataset in combination with the user-friendly screening interface of the oracle mode. This mode is excellent for teaching purposes.

The source code of ASReview is available open source under an Apache-2.0 license on GitHub. Compiled and packaged versions of the software are available on the Python Package Index or Docker Hub.

If you encounter any issues during the process, first consult the *Frequently Asked Questions* or the Troubleshooting in the *installation manual*. If you cannot find your problem, file an issue via Github.

Oracle Mode

Simulation Mode

Exploration Mode
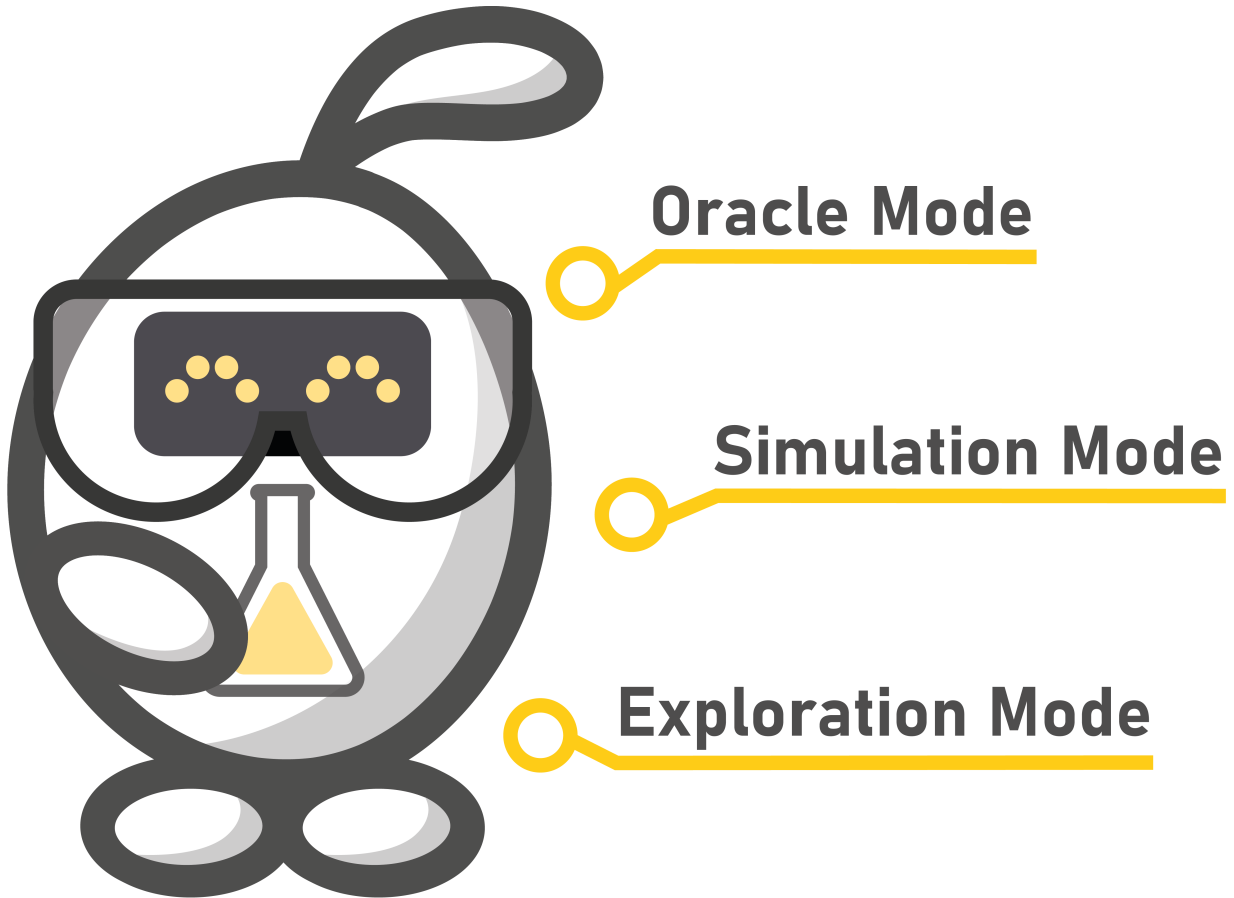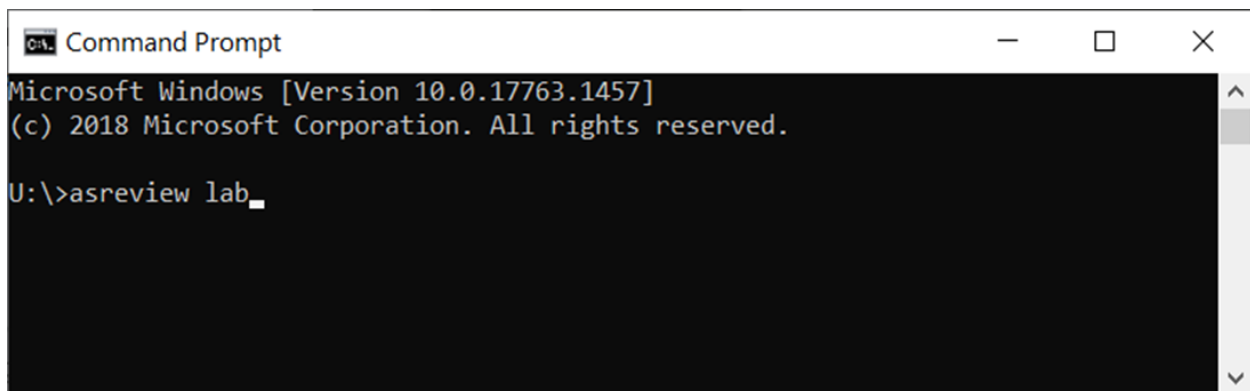
ASReview LAB

**Research software to explore the
future of AI in Systematic Reviews**

Start, Open and Import a Project

This is a quick tour on launching ASReview and basic project routines like starting, opening and importing a project. It assumes you have ASReview installed. If this is not the case, see the *Installation* guide.

9.1 Launch ASReview Lab

Launch ASReview LAB by running the following command in the command line (*CMD.exe* for Windows or *Terminal* for MacOS/Linux):


```
Command Prompt                                    —   □   ×
Microsoft Windows [Version 10.0.17763.1457]
(c) 2018 Microsoft Corporation. All rights reserved.

U:\>asreview lab_
```
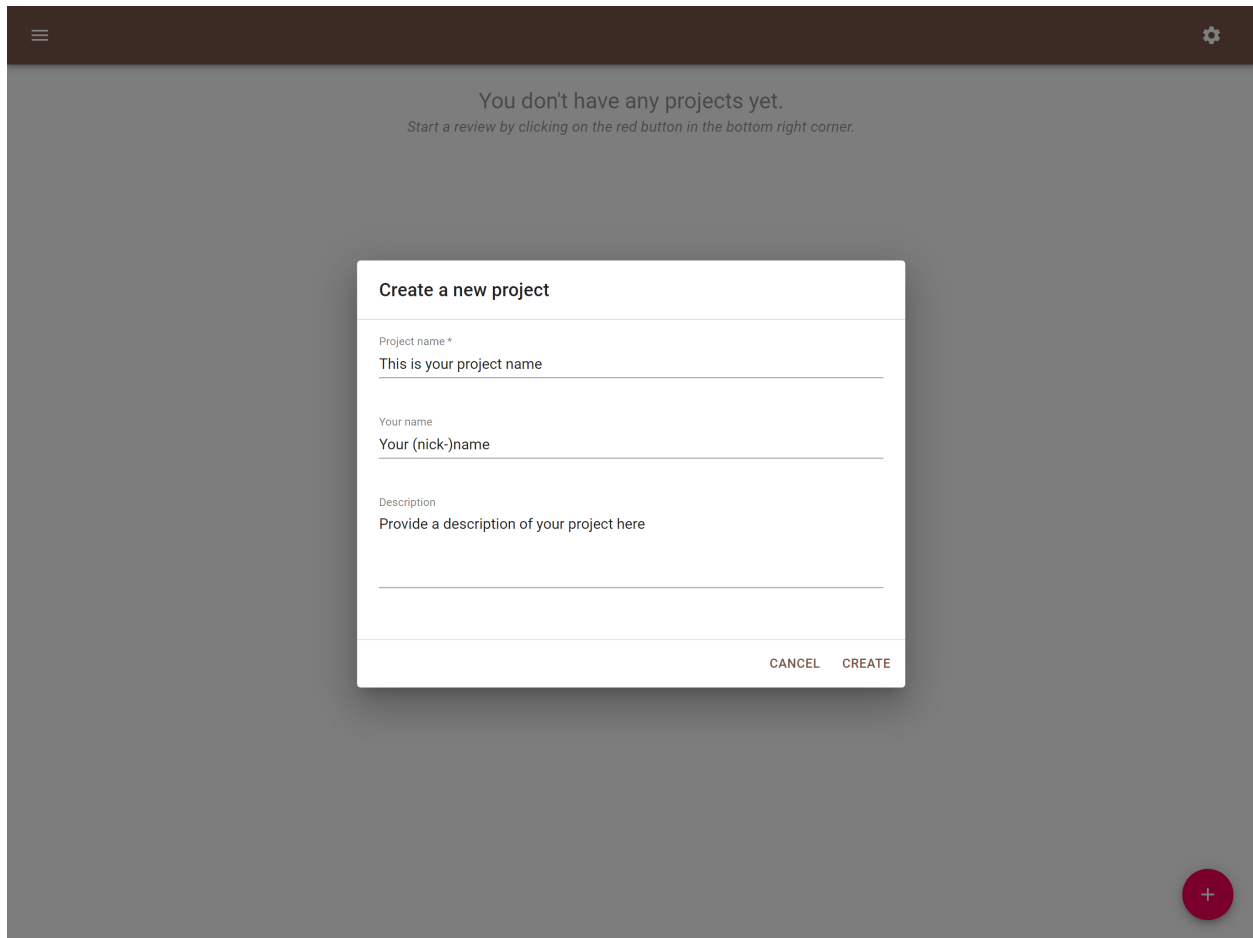
9.2 Create a New Project

To start reviewing a dataset with ASReview LAB, you first need project to initialize a project. Click on the red button in the bottom right corner, select **new** and a pop-screen will appear.

You don't have any projects yet.

Start a review by clicking on the red button in the bottom right corner.

New project   +

Import project   🗀

×
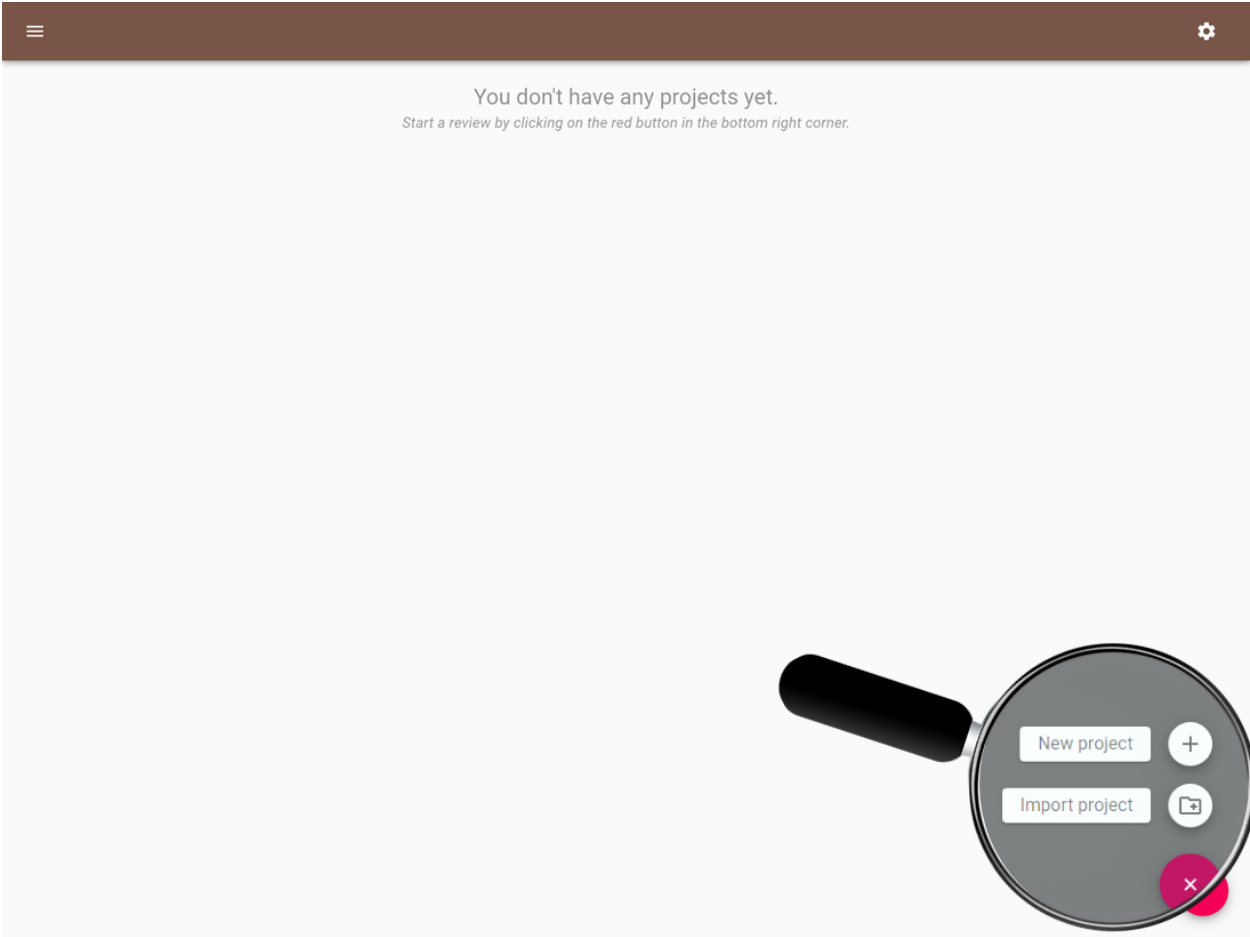
9.3 Provide Project Info

Next, provide a project name (obligatory), your name and a short description on your systematic review project.

9.4 Open a Project

If you want to continue with an existing project, simply click on the title.
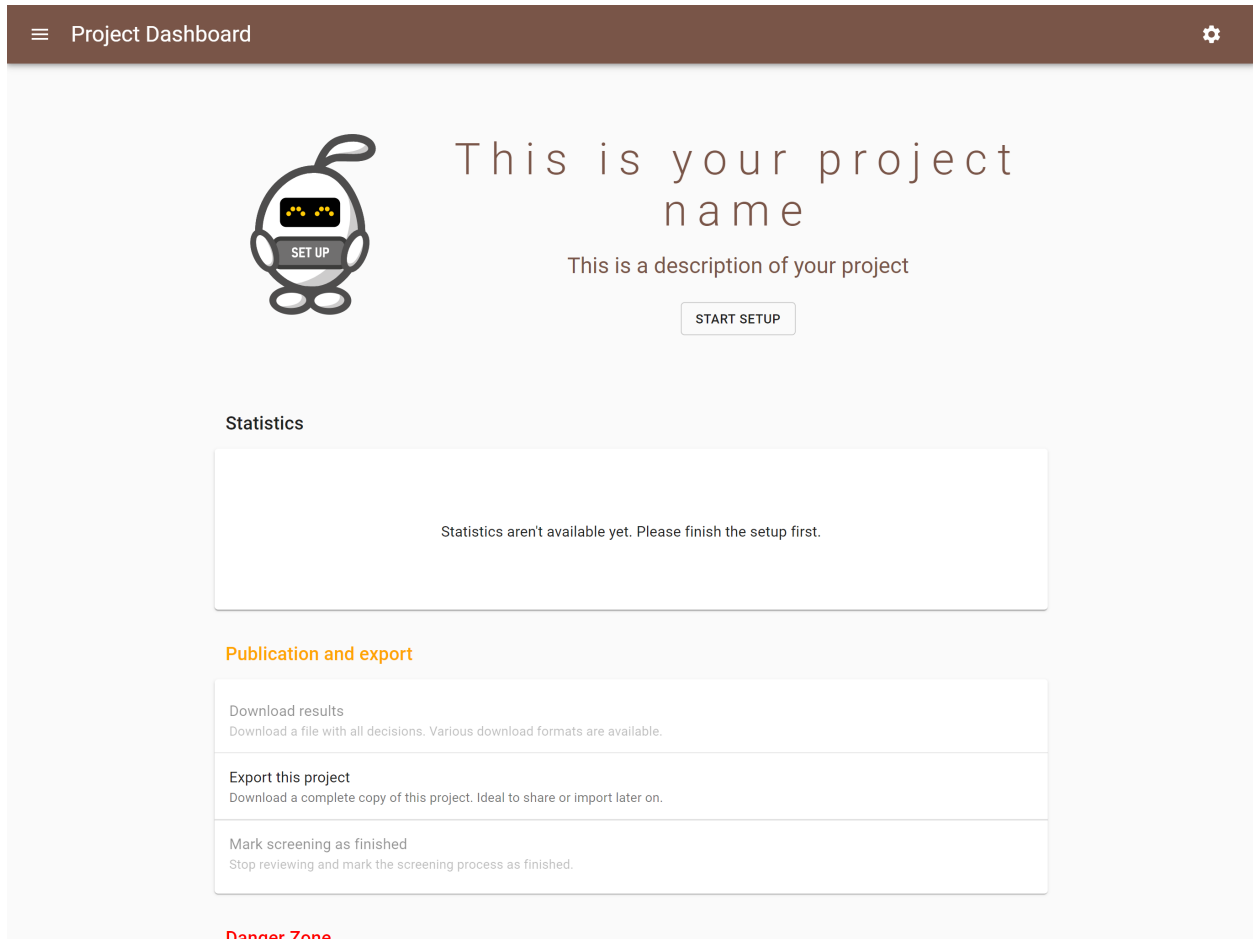
9.5 Import a Project

Another option is to import an ASReview project file (`.asreview` extension). Usually, this is a project exported from ASReview LAB via the *Export Project* panel. Importing can be done by clicking the red button in the bottom right corner of the home page. In the pop-up screen choose the project file from your computer and click *Import*. After a successful project initialization, a project dashboard will be shown.

Test

This was a test

Project 2

This is my second project

Project 1

This is my first project
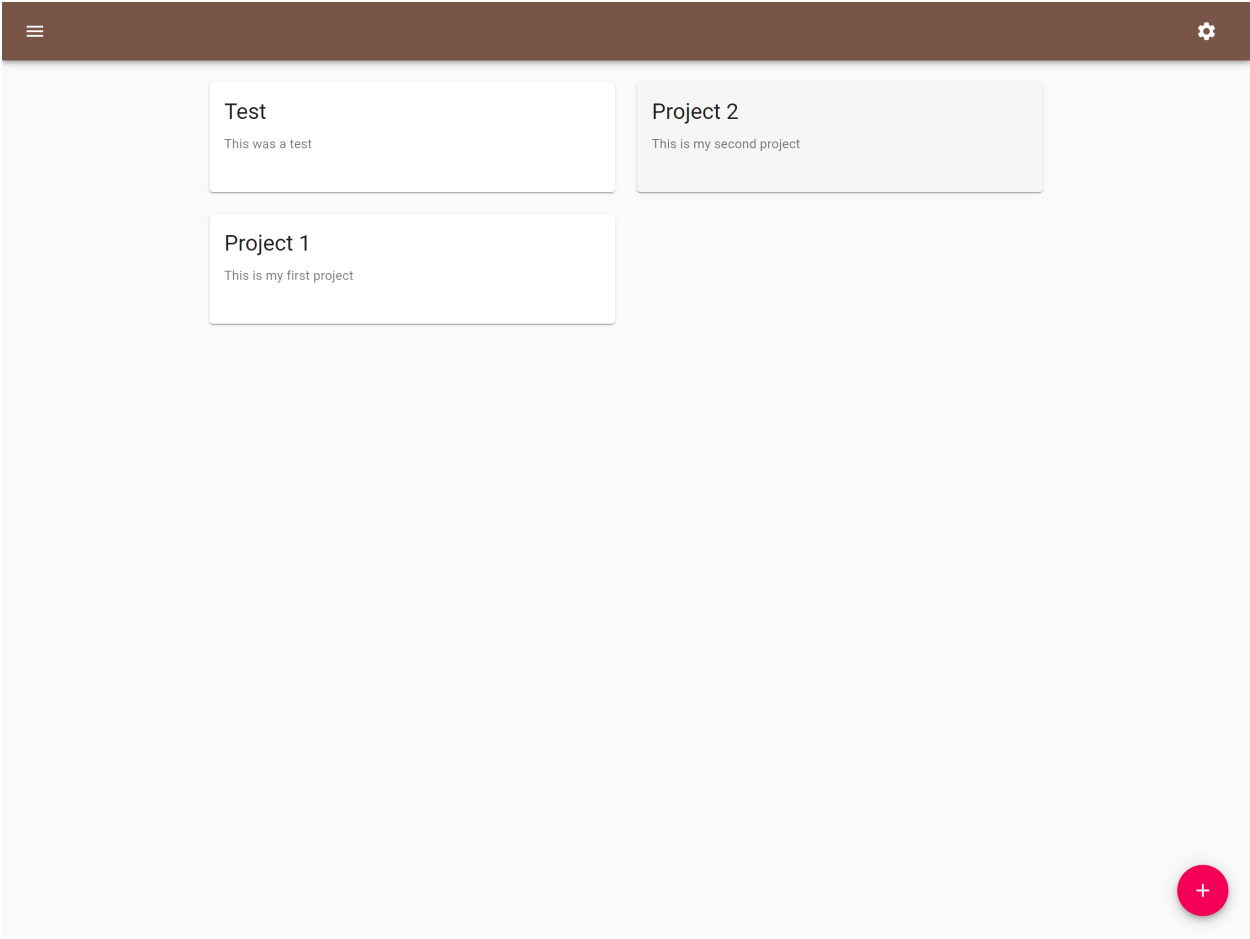
9.6 Project Dashboard

After a successful project initialization, a project dashboard will be shown and you are ready to continue with setting-up the project, like uploading data for the *Oracle Mode* or the *Exploration Mode*. The other options in the project dashboard are described in the *features section*.

Oracle Mode

This is a quick tour in using the ASReview LAB software in Oracle Mode, which is the user-friendly frontend for active learning in systematic reviews for unlabeled data with interaction by the user. A more elaborate instruction can be found in this blogpost on the ASReview website.

This tutorial assumes you have already installed Python and ASReview. If this is not the case, check out the *Installation* page. Also, you should have created a *project*.

10.1 Select Dataset

Select the dataset you want to use, which should contain at least the titles and/or abstracts of all documents (records) you want to screen.

There are four ways to select a dataset:

- Upload your own dataset. Read more about the format on *Prepare your Data*.

- Import a dataset with an URL. Read more about the format on *Prepare your Data*.

- Select a dataset from an *extension* (for example to use the *COVID-19 extension*).

- Choose one of the *benchmark data sets*.

After a successfull upload of the data, move to the next step.

> **Warning:** If you upload your own data, make sure to remove duplicates and to retrieve as many abstracts as possible (don't know how?). With clean data you benefit most from what *active learning* has to offer.

10.2 Select Prior Knowledge

The first iteration of the *active learning cycle* requires some prior knowledge to work. This knowledge is used to train the first model. In this step you need to provide at least one relevant and one irrelevant document. To facilitate this,

Select dataset

| FROM FILE | FROM URL | FROM PLUGIN | BENCHMARK DATASETS |

Drag 'n' drop a file here, or click to a file

Select dataset

my_dataset

✓ Successful upload
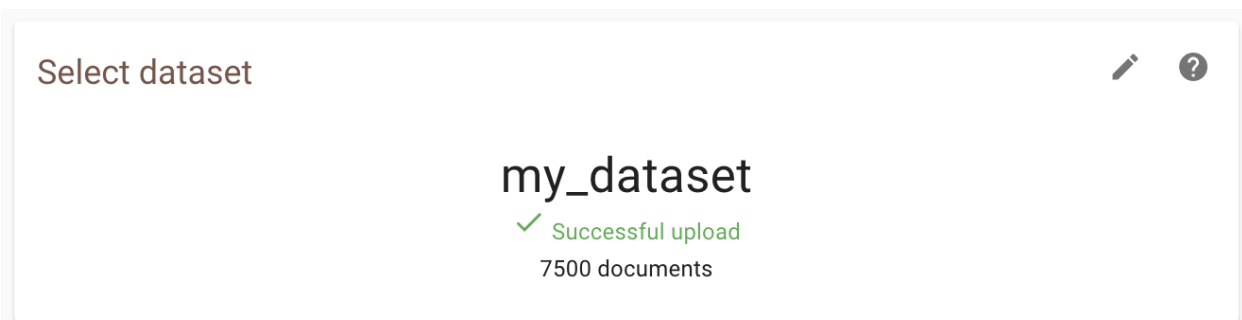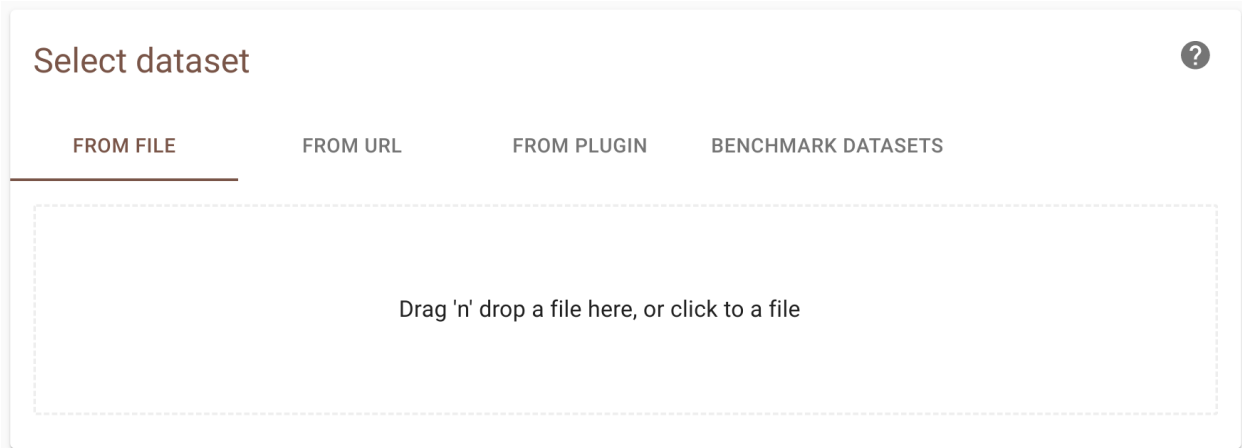
7500 documents

it is possible to *search for specific records* within your dataset (for finding prior relevant papers), ask the software to present a couple of *random documents* (for prior irrelevant papers), or to upload *partly labeled data*.

10.3 Select Active Learning Model

In the next step of the setup, you can *select a model*. The default setup (Naïve Bayes, tf-idf, Max) overall has fast and *excellent performance*, but many more options are *davaialble* . After choosing your model, click on *Finish*. You will return to the project page and the model is trained for the first time.

10.4 Start Reviewing

As soon as the model is ready, a button appears with **Start Review**. Click the button to start screening. ASReview LAB presents you a document to screen and label. If you have selected certainty-based sampling it will be the document with the highest relevance score.

You are asked to make a decision: relevant or irrelevant?

While you review the documents, the software continuously improves its understanding of your decisions, constantly updating the underlying model.

As you keep reviewing documents and providing more labels, the number of unlabeled docuemtns left in the dataset will decline. When to stop is left to the you. The blogpost *ASReview Class 101* provides some tips on stopping with screening.

Select prior knowledge

3 relevant documents
5 irrelevant documents

✓ Enough prior knowledge, feel free to go to the next step.

SEARCH   RANDOM

NEXT

Select Active learning model

Classifier: Naïve Bayes

Query strategy: Max

Feature extraction: tf-idf

10.5 Download Results

During the screening or via the *dashboard* you can download the results with your decisions by clicking the download icon. A dialog will show the download options. Choose from the menu whether you would like to download your results as a CSV or an Excel file and click *Download*.
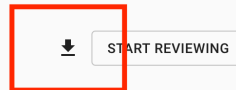
10.6 Return to Project Dashboard

If you want to return to the project dashboard, click the hamburger menu (top left) and click **Project Dashboard**.
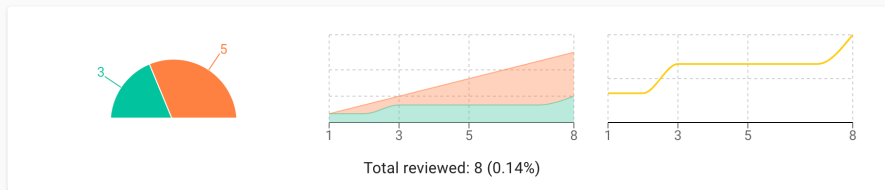
CHAPTER 11

Exploration Mode

The exploration mode can be used to explore to performance of the active learning software and the performance of *different algorithms* on already labeled data. In this mode relevant records are displayed in green and a recall curce can be obtained.

It is assumed you have already installed Python and ASReview. If this is not the case, see *Installation*. Also, you should have created a *project* - the name is not relevant, but is adviced to have a explore-prefix.

11.1 Upload a Benchmark Dataset

Select one of the available *benchmark-datasets*.

11.1.1 Prior Inclusions

In the next step, you are asked to add prior inclusions. Select 1-5 papers of your choice. For the featured datasets you can use the following titles of papers:

PTSD Trajectories:

- Latent trajectories of trauma symptoms and resilience: the 3-year longitudinal prospective USPER study of Danish veterans deployed in Afghanistan

- A Latent Growth Mixture Modeling Approach to PTSD Symptoms in Rape Victims

- Peace and War: Trajectories of Posttraumatic Stress Disorder Symptoms Before, During, and After Military Deployment in Afghanistan

- The relationship between course of PTSD symptoms in deployed U.S. Marines and degree of combat exposure

- Trajectories of trauma symptoms and resilience in deployed US military service members: Prospective cohort study

Virus Metagenomics:

- Detection of viromes of RNA viruses using the next generation sequencing libraries prepared by three methods

Select dataset

FROM FILE   FROM URL   FROM PLUGIN   BENCHMARK DATASETS

Featured benchmark datasets

van de Schoot et al. (2017)   PTSD Trajectories

The GRoLTS-Checklist: Guidelines for Reporting on Latent Trajectory Studies

DOI: 10.1080/10705511.2016.1247646

License: CC-BY Attribution 4.0 International

Location: Link to the dataset

USE DATASET

Kwok et al. (2020)   Virus Metagenomics

Hall et al. (2012)   Software Fault Prediction

Cohen et al. (2006)   ACEInhibitors

- Identification of a novel astrovirus in domestic sheep in Hungary
- Identification of novel bovine group A rotavirus G15P[14] strain from epizootic diarrhea of adult cows by de novo sequencing using a next-generation sequencer
- Detection of novel viruses in porcine fecal samples from China
- Metagenomic Analysis of the Jinding Duck Fecal Virome

Software Fault Prediction:

- Predicting Defect-Prone Software Modules at Different Logical Levels
- Quantitative analysis of faults and failures in a complex software system
- A Comprehensive Empirical Study of Count Models for Software Fault Prediction
- Predicting fault prone modules by the Dempster-Shafer belief networks
- Robust prediction of fault-proneness by random forests

ACEinhibitors:

- Quinapril in patients with congestive heart failure: controlled trial versus captopril.
- Clinical effects of early angiotensin-converting enzyme inhibitor treatment for acute myocardial infarction are similar in the presence and absence of aspirin: systematic overview of individual data from 96,712 randomized patients. Angiotensin-converting Enzyme Inhibitor Myocardial Infarction Collaborative Group.
- Efficacy of different drug classes used to initiate antihypertensive treatment in black subjects: results of a randomized trial in Johannesburg, South Africa.
- Long-term mortality in patients with myocardial infarction: impact of early treatment with captopril for 4 weeks.
- Comparison of perindopril versus captopril for treatment of acute myocardial infarction.

11.1.2 Prior Exclusions

Mark five random papers as irrelevant.

11.1.3 START reviewing

Start reviewing the first 50, 100 or even 200 papers. Abstracts in green are relevenant papers and abstracts in black are irrelevant.

For the *PTSD Trajectories* dataset you expect to find about 7 out of 38 relevant papers after screening 50 papers, 19 after screening 100 papers and 36 after 200 papers.

For the *Virus Metagenomics* dataset you expect to find 20 out of 120 relevant papers after screening 50 papers, 40 after screening 100 papers and 70 after 200 papers

For the *Software Fault Prediction* dataset you expect to find 25 out of 104 relevant papers after screening 50 papers, 48 after screening 100 papers and 88 after 200 papers.

For the *ACEinhibitors* dataset you expect to find 16 out of 41 relevant papers after screening 50 papers, 27 after screening 100 papers and 32 after 200 papers.

11.2 Upload Your own Data for Exploration

You can explore a previously labeled dataset in ASReview LAB by adding an extra column called 'debug_label' to your dataset. This column should indicate the relevant (1) and irrelevant (0) records. The relevant records will show up green during screening.

1. Open ASReview LAB.

2. Start a new project.

3. Click the *Start Setup* button.

4. Select your labeled dataset containing the 'debug_label'.

Simulation Mode

At the moment, the ASReview simulation mode is only available in the command line interface. When using the *ASReview command line interface for simulation*, a fully labeled dataset is required (labeling decisions: `0` = irrelevant, `1` = relevant).

See the following resources for information on running a simulation:

- *ASReview command line interface for simulation*
- *Simulation results*
- *Simulation*

> **Warning:** If you upload your own data, make sure to remove duplicates and to retrieve as many abstracts as possible (don't know how?). With clean data you benefit most from what *active learning* has to offer.
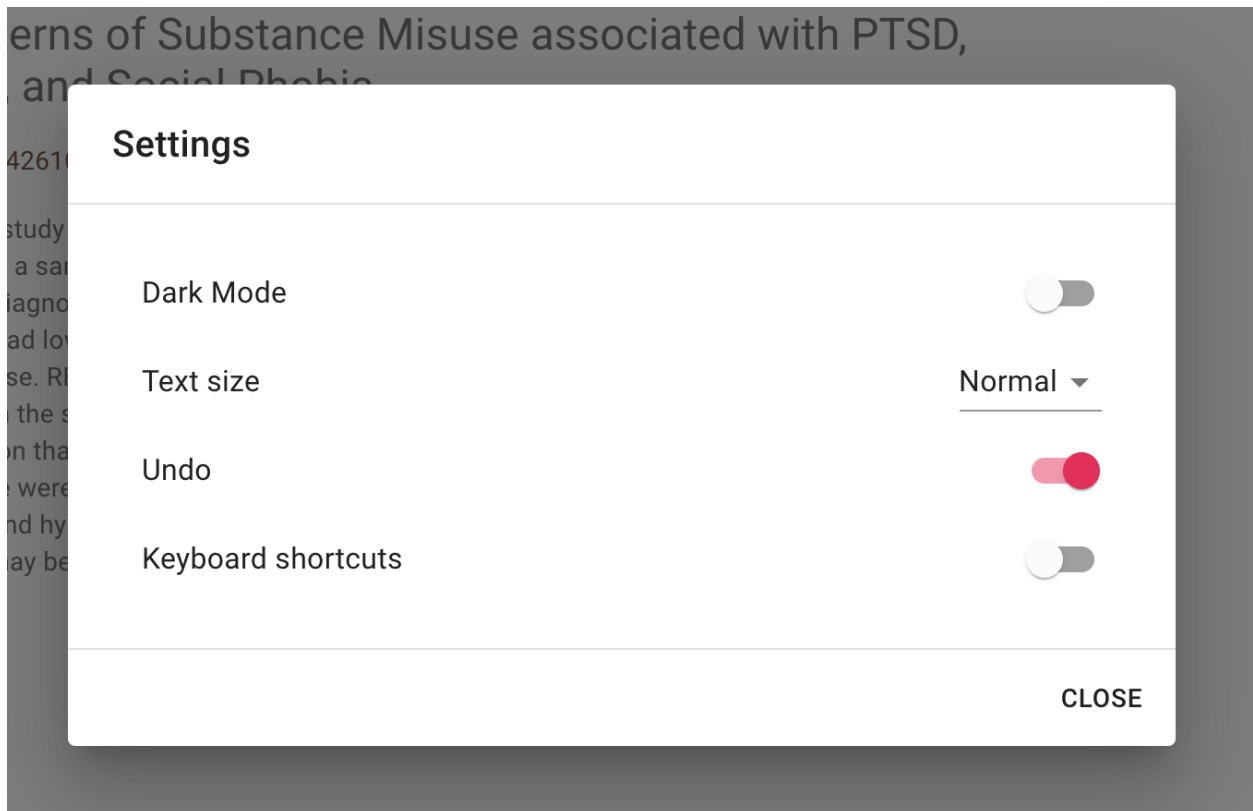
Settings

13.1 Dark Mode

ASReview LAB supports dark mode. By default, the dark mode is disabled. Enable the dark mode as follows:

1. Open ASReview LAB.

2. Click on the settings icon (top right).

3. Toggle the **Dark Mode** field.

Note: Your preference is saved in the browser.

13.2 Text Size

ASReview LAB supports different text sizes. By default, the text size is set to normal. Change the text size as follows:
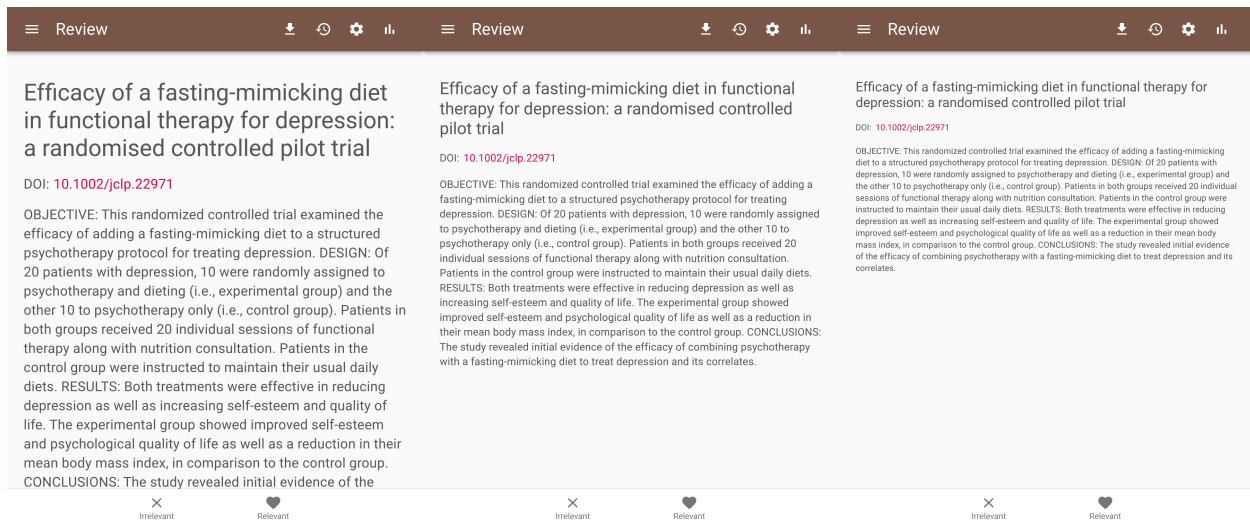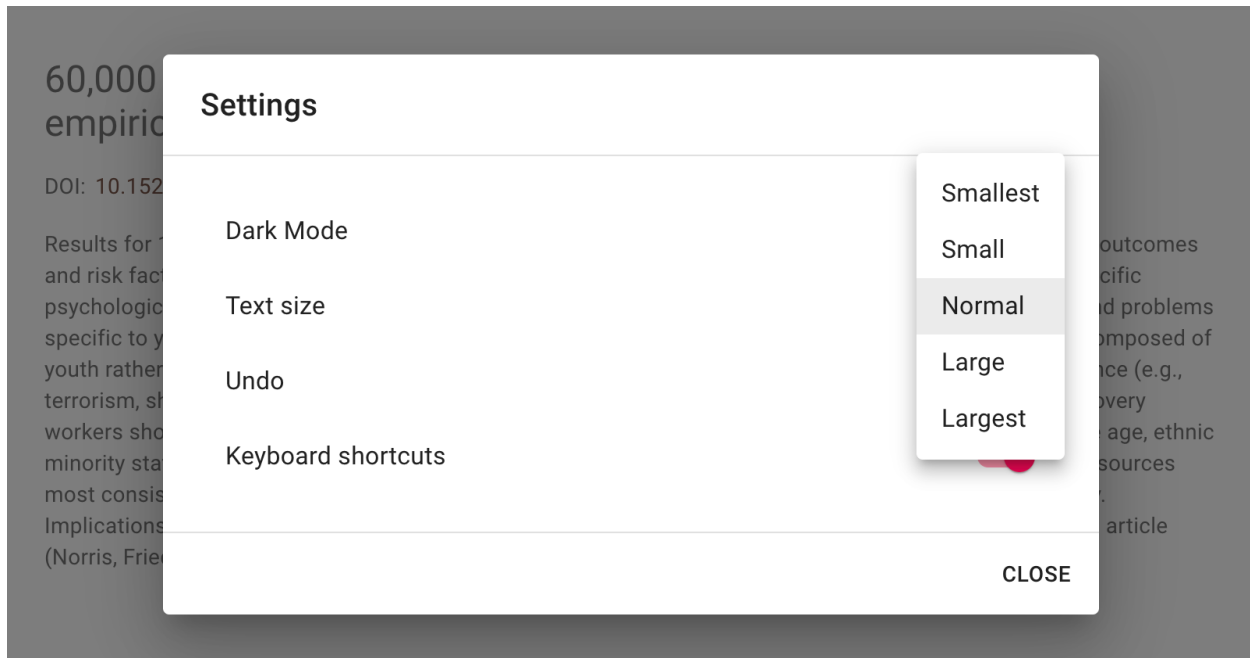
1. Open ASReview LAB.

2. Click on the settings icon (top right).

3. Select the size in the **Text Size** field.

Resulting in:

Note: Your preference is saved in the browser.

13.3 Toggle Undo option

It is possible to enable or disable the *undo option* in the settings menu:

Settings

Dark Mode

Text size

| Smallest |
| Small |
| **Normal** |
| Large |
| Largest |

Undo

Keyboard shortcuts

CLOSE

≡ Review   ⬇ ⟳ ⚙ �ᵢₗᵢ

Efficacy of a fasting-mimicking diet in functional therapy for depression: a randomised controlled pilot trial

DOI: 10.1002/jclp.22971

OBJECTIVE: This randomized controlled trial examined the efficacy of adding a fasting-mimicking diet to a structured psychotherapy protocol for treating depression. DESIGN: Of 20 patients with depression, 10 were randomly assigned to psychotherapy and dieting (i.e., experimental group) and the other 10 to psychotherapy only (i.e., control group). Patients in both groups received 20 individual sessions of functional therapy along with nutrition consultation. Patients in the control group were instructed to maintain their usual daily diets. RESULTS: Both treatments were effective in reducing depression as well as increasing self-esteem and quality of life. The experimental group showed improved self-esteem and psychological quality of life as well as a reduction in their mean body mass index, in comparison to the control group. CONCLUSIONS: The study revealed initial evidence of the

✕ Irrelevant   ♥ Relevant

≡ Review   ⬇ ⟳ ⚙ �

Efficacy of a fasting-mimicking diet in functional therapy for depression: a randomised controlled pilot trial

DOI: 10.1002/jclp.22971

OBJECTIVE: This randomized controlled trial examined the efficacy of adding a fasting-mimicking diet to a structured psychotherapy protocol for treating depression. DESIGN: Of 20 patients with depression, 10 were randomly assigned to psychotherapy and dieting (i.e., experimental group) and the other 10 to psychotherapy only (i.e., control group). Patients in both groups received 20 individual sessions of functional therapy along with nutrition consultation. Patients in the control group were instructed to maintain their usual daily diets. RESULTS: Both treatments were effective in reducing depression as well as increasing self-esteem and quality of life. The experimental group showed improved self-esteem and psychological quality of life as well as a reduction in their mean body mass index, in comparison to the control group. CONCLUSIONS: The study revealed initial evidence of the efficacy of combining psychotherapy with a fasting-mimicking diet to treat depression and its correlates.

✕ Irrelevant   ♥ Relevant

≡ Review   ⬇ ⟳ ⚙ �

Efficacy of a fasting-mimicking diet in functional therapy for depression: a randomised controlled pilot trial

DOI: 10.1002/jclp.22971

OBJECTIVE: This randomized controlled trial examined the efficacy of adding a fasting-mimicking diet to a structured psychotherapy protocol for treating depression. DESIGN: Of 20 patients with depression, 10 were randomly assigned to psychotherapy and dieting (i.e., experimental group) and the other 10 to psychotherapy only (i.e., control group). Patients in both groups received 20 individual sessions of functional therapy along with nutrition consultation. Patients in the control group were instructed to maintain their usual daily diets. RESULTS: Both treatments were effective in reducing depression as well as increasing self-esteem and quality of life. The experimental group showed improved self-esteem and psychological quality of life as well as a reduction in their mean body mass index, in comparison to the control group. CONCLUSIONS: The study revealed initial evidence of the efficacy of combining psychotherapy with a fasting-mimicking diet to treat depression and its correlates.

✕ Irrelevant   ♥ Relevant

13.3. Toggle Undo option   47

1. Open ASReview LAB.

2. Click on the settings icon (top right).

3. Uncheck the **Undo** field.

13.4 Toggle Shortcuts

ASReview LAB supports the use of *keyboard shortcuts* during screening. By default, keyboard shortcuts are disabled. Enable the shortcuts as follows:

1. Open ASReview LAB.

2. Click on the settings icon (top right).

3. Check the **keyboard settings** field.

Pre-Screening

Before you can actually start screening you have to initialize a project, select a dataset, prior knowledge and a model.

14.1 Start Setup

After you have started a project, you are redirected to the project dashboard and you will first be asked to initialize the setup.

1. Open ASReview LAB.

2. Start a new project.

3. Click the *Start Setup* button.

Note: some of the features available in the project dashboard are described in the *Post-Screening* section.

14.2 Select Dataset

To select a dataset:

1. Open ASReview LAB.

2. Start a new project.

3. Click the *Start Setup* button.

4. Choose one of the four options to select a dataset and click upload:

Warning: If you upload your own data, make sure to remove duplicates and to retrieve as many abstracts as possible (don't know how?). With clean data you benefit most from what *active learning* has to offer.

14.2.1 From File

Upload your file by *Drag 'n' Drop*, or select your file via the browser. The data needs to adhere to a *specific format*. If a file is uploaded and recognized as one of the available formats, it will display the message *Successful upload* and state the number of records in the dataset.

14.2.2 From URL

Fill in a link to a file on the Internet. For example, a link from this dataset repository.

14.2.3 From Plugin

Select a file available via a plug-in like the *COVID-19 plugin*.

14.2.4 Benchmark Datasets

Select one of the *benchmark datasets*.

14.3 Partly Labeled Data

If you want to include decisions you've already made prior to setting up your project, you can upload a partly labeled dataset containing labels for part of the data and unlabeled recors you want to screen with ASReview. This might be helpful if you switch from screening in another tool to screening with ASReview, or when updating an existing systematic review with more recent publications.

Currently, this can be done by merging your dataset with labeled and unlabeled records via Excel or another reference manager. Your dataset should contain a column, called *label_included* (or: final_included, label, label_included, included_label, included_final, included, included_flag, include) which is filled with 1's or 0's for the publications that you have already screened, and is empty for the records that you still need to screen using ASReview.

To use a partly labeled dataset:

1. Open ASReview LAB.

2. Start a new project.

3. Click the *Start Setup* button.

4. Select your partly labeled dataset.

ASReview will recognize the column with the labels and show you the number of prior relevant/irrelevant papers in the section *Prior Knowledge*.

14.4 Select Prior Knowledge

The first iteration of the *active learning cycle* requires prior knowledge to work. This knowledge is used to train the first model. In this step you need to provide at least one relevant and one irrelevant document. To facilitate this, it is possible to search within your dataset (for finding prior relevant papers) or ask the software to present a couple of random documents (for prior irrelevant papers).

1. Open ASReview LAB.

2. Start a new project.

3. Click the *Start Setup* button.

4. Select a dataset.

5. Click **Search** or **Random** to select your prior knowledge.

Select prior knowledge   ❓

0 relevant documents
0 irrelevant documents

You don't have prior knowledge yet. Find yourself prior knowledge by searching relevant documents and label some
random documents.

SEARCH   RANDOM

After selecting some prior information, you can click **Next**.

Select prior knowledge   ✏️   ❓

3 relevant documents
5 irrelevant documents

✓ Enough prior knowledge, feel free to go to the next step.

SEARCH   RANDOM

NEXT

14.4.1 Search

Let's start with finding a prior relevant document. The most efficient way to do this is by searching for a specific
document which you already know is relevant. Click the search button and search your dataset by authors, keywords
or title, or a combination thereof. Enter your search terms (press 'enter' to start searching).

Click the document you had in mind and click Relevant (Clicking Irrevant results in an irrelevant document).

The Prior Knowledge step will now show 1 relevant document. This is already enough to proceed to the next step. Note
that there are no restrictions on the number of publications you need to provide, but preferably provide 1-5 relevant
documents.

Select prior knowledge

1 relevant documents
1 irrelevant documents

✓ Enough prior knowledge, however a bit more would help!

SEARCH   RANDOM

Search for a document of interest.

latent growth PTSD   🔍

Search result: latent growth PTSD

Latent Growth Mixture Models to estimate PTSD trajectories
Van de Schoot, R.

Using Bayesian statistics for modeling PTSD through Latent Growth Mixture Modeling: implementation and discussion
Depaoli, S., van de Schoot, R., van Loey, N., Sijbrandij, M.

A Latent Growth Mixture Modeling Approach to PTSD Symptoms in Rape Victims
Armour, C., Shevlin, M., Elklit, A., Mroczek, D.

Heterogeneity in signaled active avoidance learning: Substantive and methodological relevance of diversity in instrumental defensive responses to threat cues
Galatzer-Levy, I. R., Moscarello, J., Blessing, E. M., Klein, J., Cain, C. K., LeDoux, J. E.

636,120 Ways to Have Posttraumatic Stress Disorder
Galatzer-Levy, I. R., Bryant, R. A.

Exploring the association between posttraumatic growth and PTSD: National Study of Jews and Arabs following the 2006 Israeli-Hezbollah War
Hall, Brian J., Hobfoll, Stevan E., Canetti, Daphna, Johnson, Robert J., Palmieri, Patrick A., Galea, Sandro

Prior trauma and PTSD among homeless mothers: Effects on subsequent stress appraisals, coping, posttraumatic growth, and health outcomes
Schuster, Jennifer Lynn

The course of PTSD symptoms among Gulf War veterans: A growth mixture modeling approach
Orcutt, H. K., Erickson, D. J., Wolfe, J.

Posttraumatic stress after a motor vehicle accident: a six-month follow-up study utilizing latent growth modeling
Wu, K. K., Cheung, M. W.

Heterogeneity in threat extinction learning: substantive and methodological considerations for identifying individual difference in response to stress
Galatzer-Levy, I. R., Bonanno, G. A., Bush, D. E., Ledoux, J. E.

If you are done click **Next**.

14.4.2 Random

You also need to provide at least one prior irrelevant document. One way to find an irrelevant document is by labeling a set of random records from the dataset. Given that the majority of documents in the dataset are irrelevant (extremely imbalanced data problem), the documents presented here are likely to be irrelevant for your study. Click on random to show a few random documents. Indicate for each document whether it is relevant or irrelevant.
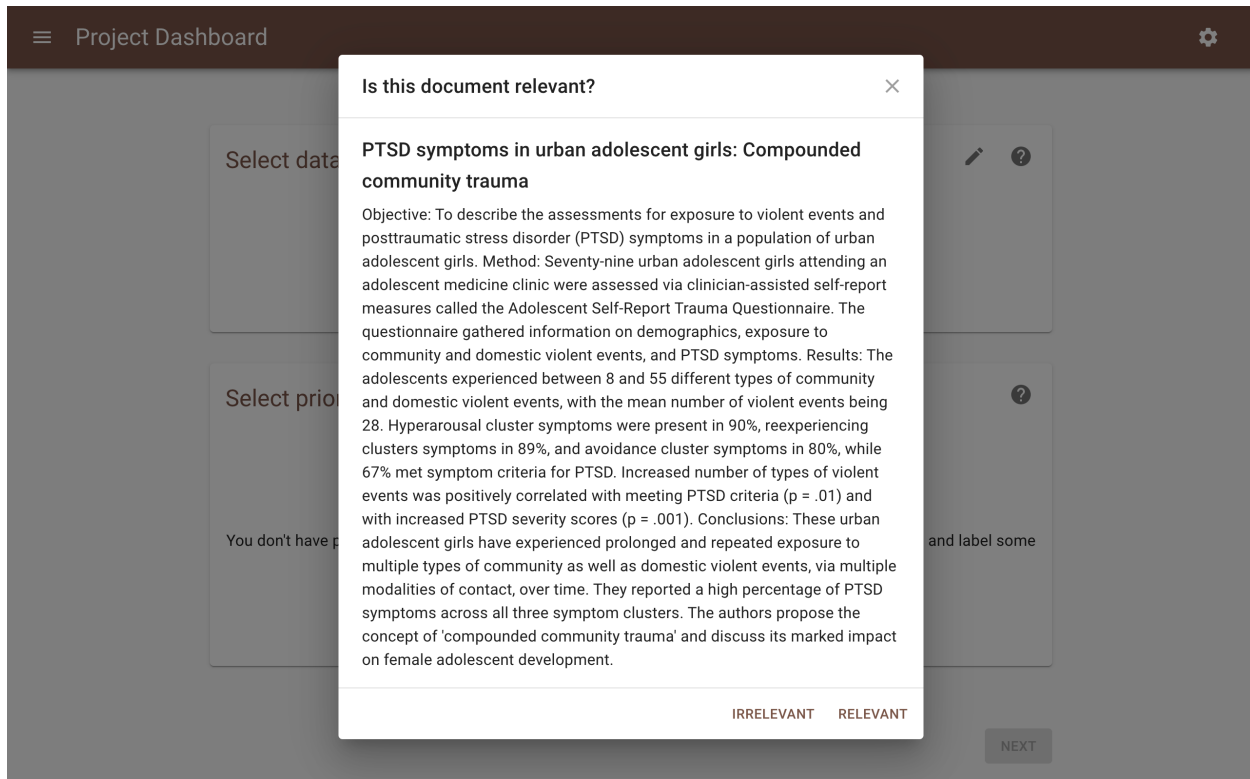
After labeling a couple of randomly selected documents, ASReview LAB will ask you whether you want to stop. Click on **STOP** and click **Next**.

14.5 Select Model

It is possible to change the settings of the Active learning model. There are three ingredients that can be changed in the software: the type of classifier, the query strategy and the feature extraction technique.

To change the default setting:

1. Open ASReview LAB.

2. Start a new project, upload a dataset and select prior knowledge.

3. Click on the **edit** icon (top right).

4. Using the drop-down menu select a different classifier, query strategy or feature extraction technique.

5. Click Finish.

Select Active learning model ✎ ❓

Classifier: Naïve Bayes

Query strategy: Max

Feature extraction: tf-idf

The classifier is the machine learning model used to compute the relevance scores. The available classifiers are Naive Bayes, Support Vector Machine, Logistic Regression, and Random Forest. More classifiers can be selected via the *API*. The default is Naive Bayes, though relatively simplistic, it seems to work quite well on a wide range of datasets.

The query strategy determines which document is shown after the model has computed the relevance scores. With certainty-based is selected the document with the highest relevance score is showed followed by the 2nd in line, etcetera, untill a new model is trained with new relevance scores. When uncertainty-based is selected, the most uncertain docuemtn is sampled according to the model (i.e. closest to 0.5 probability). When random is selected, as it says, randomly select samples with no regard to model assigned probabilities. **Warning**: selecting this option means your review is not going to be accelerated by ASReview.

The feature extraction technique determines the method how text is translated into a vector that can be used by the classifier. The default is TF-IDF (Term Frequency-Inverse Document Frequency) from SKLearn. It works well in combination with Naive Bayes and other fast training models. Another option is Doc2Vec provided by the gensim package which needs to be installed manually. To use it, install the gensim package manually:

```
pip install gensim
```

It takes relatively long to create a feature matrix with this method. However, this only has to be done once per simulation/review. The upside of this method is the dimension-reduction that generally takes place, which makes the modelling quicker.

Screening

The screen in which you provide labels for records shown to you by the software is kept as simple as possible. This is because ASReview wants you to focus on the content of the text so that you can make your decision as a true Oracle. You can access the following features during screening.

15.1 Autosave

Your work is saved automatically. There is no need to press any buttons to save your work anywhere in ASReview LAB.

15.2 Label options

Below the text, ASReview provides two labeling options: *Relevant* or *Irrelevant*.

To make a decision:

1. Open ASReview LAB.

2. Open a project.

3. Click on either the button Relevant or Irrelevant.

4. The next record is presented. Repeat the process of labeling.

> **Warning:** If you are in doubt, think harder and take your time to make a decision, you are the oracle. Based on your input, a new model will be trained in the background. If you make decisions faster than the model needs for computing new relevance scores, you will simply be presented with the record next in line (etcetera) until the model is done training.

Review

Statistics

ASReview: Open Source Software for Efficient and Transparent Active Learning for Systematic Reviews

For many tasks - including but not limited to systematic reviews for research fields - the scientific literature needs to be checked systematically. Currently, scholars and practitioners might screen thousands of studies by hand to determine which studies to include in their review. This process is error prone and inefficient, because of the extremely imbalanced data: only a very small fraction of the studies screened will be relevant. The future of systematic reviewing will be an interaction with machine learning algorithms to deal with the enormous increase of available text. We therefore developed an open source machine learning-aided pipeline applying active learning: ASReview. We demonstrate by means of simulation studies that ASReview can yield far more efficient reviewing than manual reviewing, while exhibiting adequate quality. Furthermore, we describe the different options of the free and open source research software, we show how it can be used for screening the COVID19 literature, and we present the results from a series of user experience tests. We invite the community to contribute to open source projects such as our own, that provide measurable and reproducible improvement over current practice.

Project

Project name: AI-software

Authors: Van de Schoot et al.

Number of publications: 5782

Progress

265

38

Total reviewed: 303 (5.24%)

1   81   161   303

Since last relevant: 106

✕
Irrelevant

♥
Relevant

15.3 Undo Last Decision

In some cases, you might want to change your previous decision. The screening interface of ASReview LAB can be used to return to the previous decision.

1. Open ASReview LAB.

2. Open or create a project.

3. Label the record displayed in the screen as relevant or irrelevant.

4. Click on **Undo** (See picture below).
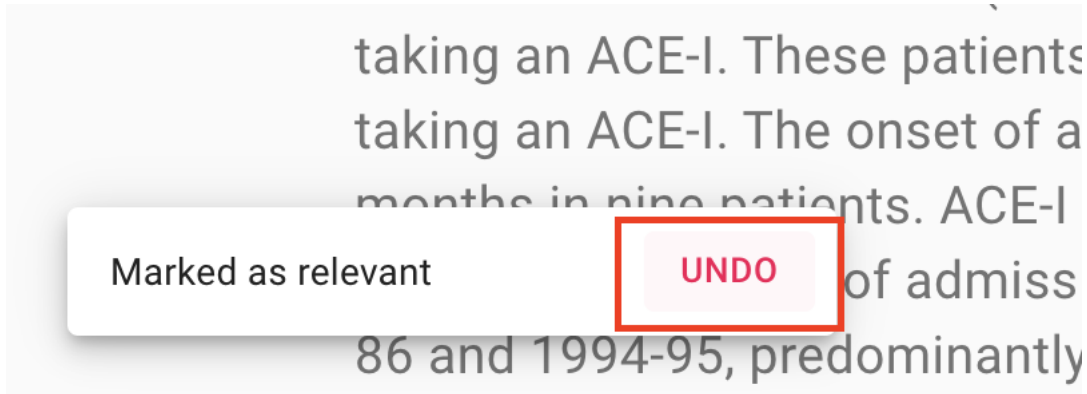
5. Click on **Keep (ir)relevant** or **Convert to (ir)relevant**

6. Continue labeling.

15.4 Review History

An overview of your decisions made during screening can be found in the **Review History** dialog.

15.4.1 Open history

1. Open ASReview LAB.

2. Open or create a project.

3. Start/continue screening.

4. Click on **Review History** in the *menu bar* on top. A dialog will open with the labeled records.

5. With the drop-down list, you can select records to display (all, relevant only, or irrelevant only).

6. By clicking on a title, the full information opens.

15.4.2 Changing decisions

7. To change a label of a record, click **convert to . . .**. The next iteration of the model will take the new label into account.

8. To go back to the overview, click ←.

Review History

All (66)
Relevant (19)
Irrelevant (47)

Six years after the wave. Trajectories of posttraumatic stress following a natural disaster

Trajectories of Posttraumatic Stress Among Urban Residents

Ten-year follow-up study of PTSD diagnosis, symptom severity and psychosocial indices in aging holocaust survivors

Different clinical courses of children exposed to a single incident of psychological trauma: A 30-month prospective follow-up study

Trajectories of psychological distress among low-income, female survivors of Hurricane Katrina

Children's Postdisaster Trajectories of PTS Symptoms: Predicting Chronic Distress

Heterogeneous depression responses to chronic pain onset among middle-aged adults: A prospective study

Exploration of delayed-onset posttraumatic stress disorder after severe injury

An introduction to latent class growth analysis and growth mixture modeling

Longitudinal measures of hostility in deployed military personnel

Beyond normality in the study of bereavement: Heterogeneity in depression outcomes following loss in older

CLOSE

9. To close the Review History, click **Close**.

15.5 Statistics Panel

For unlabeled data, ASReview LAB offers some insightful graphs to keep track of your screening process so far. To open the statistics panel:

1. Open ASReview LAB.

2. Open a project.

3. Start screening.

4. Click the **statistics** icon in the upper-right corner.

5. To close the panel click on the '>' icon.

In the top of the statistics panel the project name, authors and total number of records in the dataset are displayed.

The pie chart on the presents an overview of how many relevant (green) and irrelevant (orange) records have been screened so far. Also, the total number of records screened is displayed, as well as the percentage screened relative to the total number of records in the dataset.
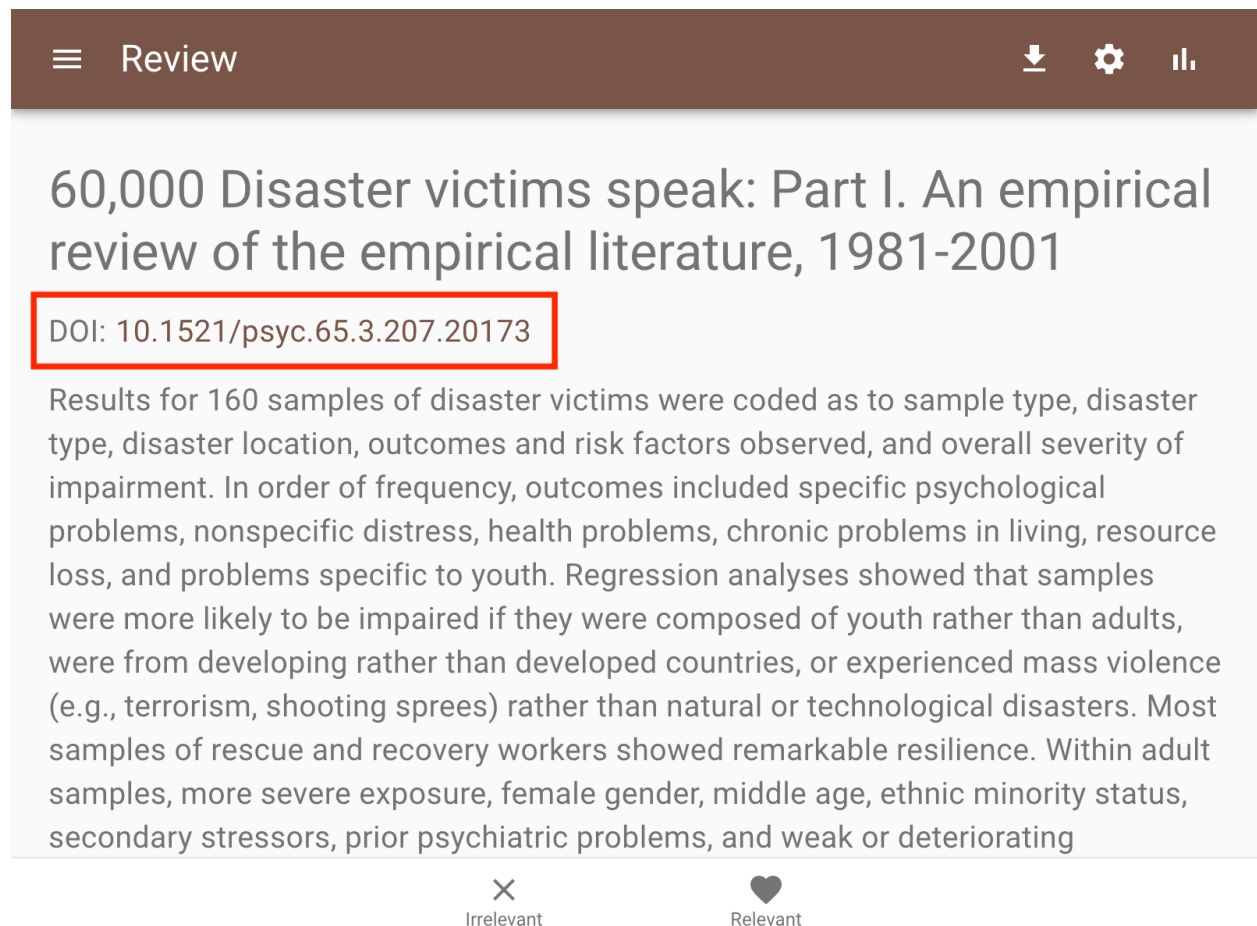
The second plot is a progress plot. On the x-axis the number of records screened is tracked. The y-axis shows a moving average. It displays the ratio between relevant and irrelevant records for a batch of 10 labeled records. If you hoover over the plot you can see the moving average for any batch of 10 labeled records.

Underneath the progress plot, the number of irrelevant records after the last relevant is shown. This statistic might help in deciding when to stop reviewing.

15.6 DOI

If a column with Digital Object Identifiers (DOI) is available in the metadata of your dataset, ASReview Lab will display the DOI with hyperlink during screening. Most of the time, DOIs point to the full-text of a publication. See *datasets* for more information on including DOI values to your datasets. To access the full text:

1. Open ASReview LAB.

2. Open a project

3. Start screening.

4. As soon as a record contains a DOI number, it will be presented below the title.

15.7 Download Results

A file containing all metadata including your decisions can be downloaded any time during the screening process. To download your results:

1. Open ASReview LAB.

2. Open a project.

3. Start screening.

4. Click the **download** icon in the upper-right corner.

5. You will be asked whether you want to save an Excel or a CSV file.

6. You will be asked where to save the file.

Download review result

Download the result of your review. Select a file format (Excel or CSV file).

File type

Excel ▾

Our software is made with love and freely available for everyone. Help the development of the ASReview with a donation: asreview.nl/donate

CANCEL   DOWNLOAD
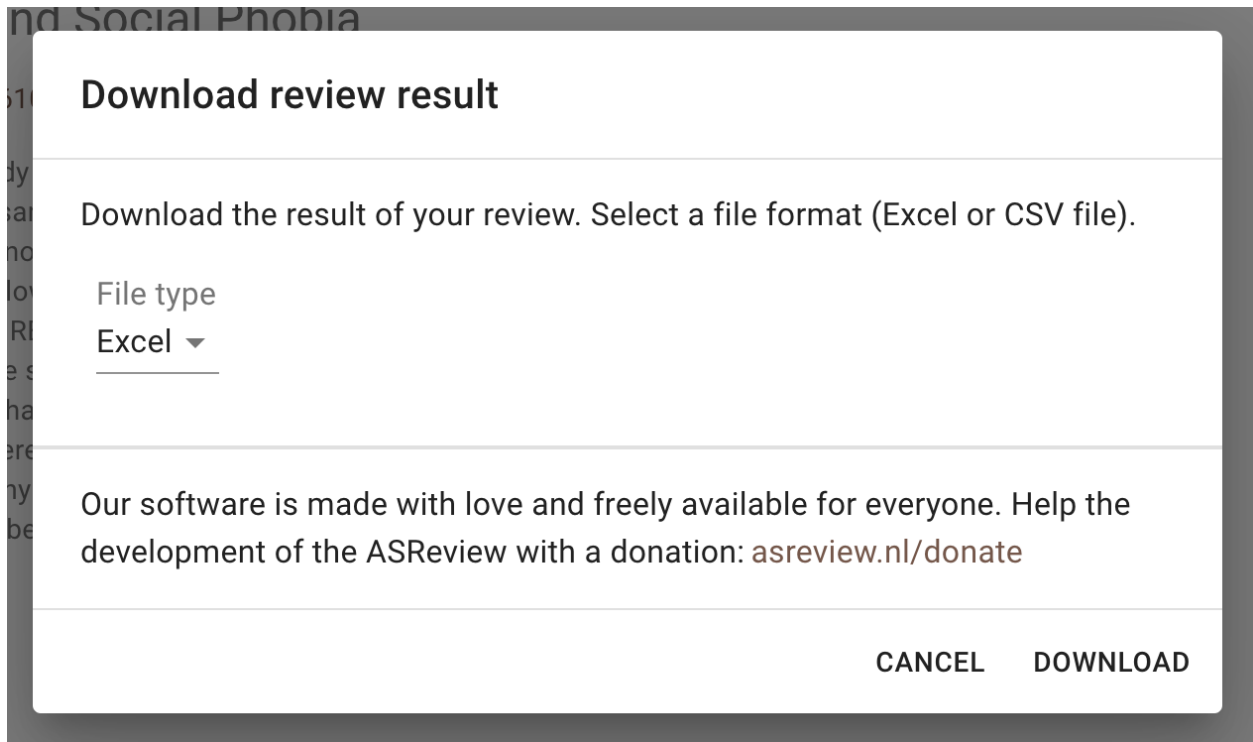
15.8 Hamburger menu

Via the hamburger menu in the left-upper corner you can:

1. Navigate back to the *overview* page containing all your projects (or to start a new project).

2. You can access the *Project Dashboard*.

3. Navigate to the documention via the HELP button.

4. Provide feedback or contribute to the code.

5. Donate to the ASReview project via the ASReview crowdfunding platform

6. Quit the software (your progress was saved automatically)

15.9 Keyboard shortcuts

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

Action	Shortcut
Label record as relevant	**r** or **Shift + r**
Label record as irrelevant	**i** or **Shift + i**
Return to previous decision	**u** or **Shift + u**

Post-Screening

After you stop screening, or anytime you want to take a break, you can return to the project dashboard by clicking the hamburger menu on the top-left. Below, you find the options in the project dashboard.

16.1 Download Results

A file containing all meta-data including your decisions can be downloaded any time during the screening process. To download your results:

1. Open ASReview LAB.

2. Start a new project, upload a dataset and select prior knowledge.

3. Navigate to the Project Dashboard.

4. Click the *download* icon (see screenshot below), or click on *Download Results*.

5. You will be asked whether you want to save an excel or a csv file.

6. You will be asked where to save the file.

Three columns will be added to your dataset:

The column titled **included** contains the labels as provided by the user: 0 = not relevant, 1 = relevant and if missing it means the record is not seen during the screening process. First, all relevant records are presented in the order these shown during the screening process. Then, all records not seen during the screening proces are presented ordered from most to least relevant according to the last iteration of the model. At end of the file all non-relevant records are presented in the order these are shown during the screening proces.

The column titled **asreview_ranking** contains an identifier to preserve the rank ordering as described above.

If present, the column **record_id** contains the values of the original **record_id** as included by the user. If not available, ASReview generates a new record_id, based on the row number and starting at 0.
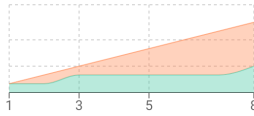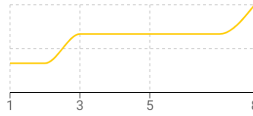
16.2 Export Project

An ASReview project file can be downloaded containing all information needed to replicate the project or to import the project on a different device. To export your project:

1. Open ASReview LAB.

2. Start a new project, upload a dataset and select prior knowledge.

3. Navigate to the Project Dashboard.

4. Click on *Export this project*.

5. You will be asked where to save the ASReview file (extension *.asreview*).

16.3 Finished

When you are done with the review, you can mark the project as finished. To mark your project as finished:

1. Open ASReview LAB.

2. Open a project.

3. Click on *Mark Screening as finished*.

The button to continue screening is now disabled. This can be undone by clicking again on *Mark Screening as finished (undo)*.

16.4 Delete a Project

To permanently delete a project, including ALL files:

1. Open ASReview LAB.

2. Start a new project, upload a dataset and select prior knowledge.

3. Navigate to the Project Dashboard.

4. Click on *Delete this project*.

5. This action cannot be made undone, ASReview LAB will ask you to confirm by typing in the project title.

Overview

ASReview has support for extensions. Extensions are a nice way to extend the functionality of the *ASReview LAB* software or the *command line interface*. See the following officially supported extensions of ASReview:

- *ASReview against COVID-19*: Make literature on COVID-19 available in ASReview.

- *ASReview-visualization*: Plotting functionality for state files produced by ASReview.

- plugin_wordcloud: Create a visual impression of the contents of datasets via a wordcloud.

- `asreview-statistics`: Tool to give some basic properties of a dataset, such as number of papers, number of inclusions. GitHub

- `asreview-hyperopt`: Optimize the hyperparameters of the models in ASReview. GitHub

If an extension is not on this list, or you make one and want it added to this list, make an issue on github.

ASReview against COVID-19

ASReview against COVID-19

For many questions from medical doctors, journalists, policy makers the scientific literature on COVID-19 needs to be checked in a systematic way to avoid biased decision-making. For example, to develop evidence-based medical guidelines to transparently support medical doctors. Medical guidelines rely on comprehensive systematic reviews. Such reviews entail several explicit and reproducible steps, including identifying all likely relevant papers in a standardized way, extracting data from eligible studies, and synthesizing the results into medical guidelines. They need to scan hundreds, or even thousands of COVID-19 related studies, by hand to find relevant papers to include in their overview. This is error prone and extremely time intensive; time we do not have right now!

The software relies on *Active learning* which denotes the scenario in which the reviewer is labeling data that are presented by a machine learning model. The machine learns from the reviewers' decisions and uses this knowledge in selecting the reference that will be presented to the reviewer next. In this way, the COVID-19 related papers are presented ordered from most to least relevant based on the input from the user. The goal of the software is to help scholars and practitioners to get an overview of the most relevant papers for their work as efficiently as possible, while being transparent in the process.

18.1 ASReview plugin

To help combat the COVID-19 crisis, the ASReview team developed a plugin that provides three datasets on COVID-19. These are automatically available in ASReview after installing the plugin, so reviewers can start reviewing the latest scientific literature on COVID-19 as soon as possible!

18.2 CORD-19 dataset

Two versions of the CORD-19 dataset (publications relating to COVID-19) are made available in ASReview:

- Full CORD-19 dataset .

- Custum made subset of the CORD-19 dataset with publications from December 2019 onwards.

The Cord19 database, developed by the Allen Institute for AI, with all publications on COVID-19 and other coronavirus research (e.g. SARS, MERS, etc.) from PubMed Central, the WHO COVID-19 database of publications, the preprint servers bioRxiv and medRxiv and papers contributed by specific publishers.

In addition to the full dataset, there is a subset available of studies published after December 1st, 2019 to search for relevant papers published during the COVID-19 crisis.

The datasets are updated in ASReview plugin shortly after the release by the Allen Institute for AI.

18.3 Pre-print dataset

A separate dataset of COVID-19 related preprints, containing metadata of preprints from over 15 preprints servers across disciplines, published since January 1, 2020. The preprint dataset is updated weekly by the maintainers (Nicholas Fraser and Bianca Kramer) and then automatically updated in ASReview as well. As this dataset is not readily available to researchers through regular search engines (e.g. PubMed), its inclusion in ASReview provided added value to researchers interested in COVID-19 research, especially if they want a quick way to screen preprints specifically.

18.4 Installation and usage

The COVID-19 plug-in requires ASReview 0.8 or higher. Install ASReview by following the instructions in *Installation*.

Install the extension with pip:

```
pip install asreview-covid19
```

The datasets are immediately available after starting ASReview.

```
asreview lab
```

The datasets are selectable in Step 2 of the project initialization. For more information on the usage of ASReview, have a look at the *Oracle Mode*.

18.5 License

The ASReview software and the plugin have an Apache 2.0 LICENSE. For the datasets, see the license of the CORD-19 dataset https://www.semanticscholar.org/cord19.

ASReview-visualization

ASReview-visualization is a plotting and visualization supplemental package for the ASReview software. It is a fast way to create a visual impression of the ASReview with different datasets, models and model parameters.

19.1 Installation

The easiest way to install the visualization package is to install from PyPI:

```
pip install asreview-visualization
```

After installation of the visualization package, `asreview` should automatically detect it. Test this by:

```
asreview --help
```

It should list the 'plot' modus.

19.2 Basic usage

State files that were created with the same ASReview settings can be put together/averaged by putting them in the same directory. State files with different settings/datasets should be put in different directories to compare them.

As an example consider the following directory structure, where we have two datasets, called `ace` and `ptsd`, each of which have 8 runs:

```
├── ace
│   ├── results_0.h5
│   ├── results_1.h5
│   ├── results_2.h5
│   ├── results_3.h5
│   ├── results_4.h5
│   ├── results_5.h5
```

```
│   │   ├── results_6.h5
│   │   └── results_7.h5
│   └── ptsd
│       ├── results_0.h5
│       ├── results_1.h5
│       ├── results_2.h5
│       ├── results_3.h5
│       ├── results_4.h5
│       ├── results_5.h5
│       ├── results_6.h5
│       └── results_7.h5
```

Then we can plot the results by:

```
asreview plot ace ptsd
```

By default, the values shown are expressed as percentages of the total number of papers. Use the `-a` or `--absolute-values` flags to have them expressed in absolute numbers:

```
asreview plot ace ptsd --absolute-values
```

Since version 0.15, you can plot project files (exported from asreview lab) as well. Use the following code:

```
asreview plot my_project_file.asreview
```

19.3 Plot types

There are currently four plot types implemented: *inclusion*, *discovery*, *limit*, *progression*. They can be individually selected with the `-t` or `--type` switch. Multiple plots can be made by using `,` as a separator:

```
asreview plot ace ptsd --type 'inclusion,discovery'
```

19.3.1 Inclusion

This figure shows the number/percentage of included papers found as a function of the number/percentage of papers reviewed. Initial included/excluded papers are subtracted so that the line always starts at (0,0).

The quicker the line goes to a 100%, the better the performance.

19.3.2 Discovery

This figure shows the distribution of the number of papers that have to be read before discovering each inclusion. Not every paper is equally hard to find.

The closer to the left, the better.

19.3.3 Limit

This figure shows how many papers need to be read with a given criterion. A criterion is expressed as "after reading y % of the papers, at most an average of z included papers have been not been seen by the reviewer, if he is using max

---
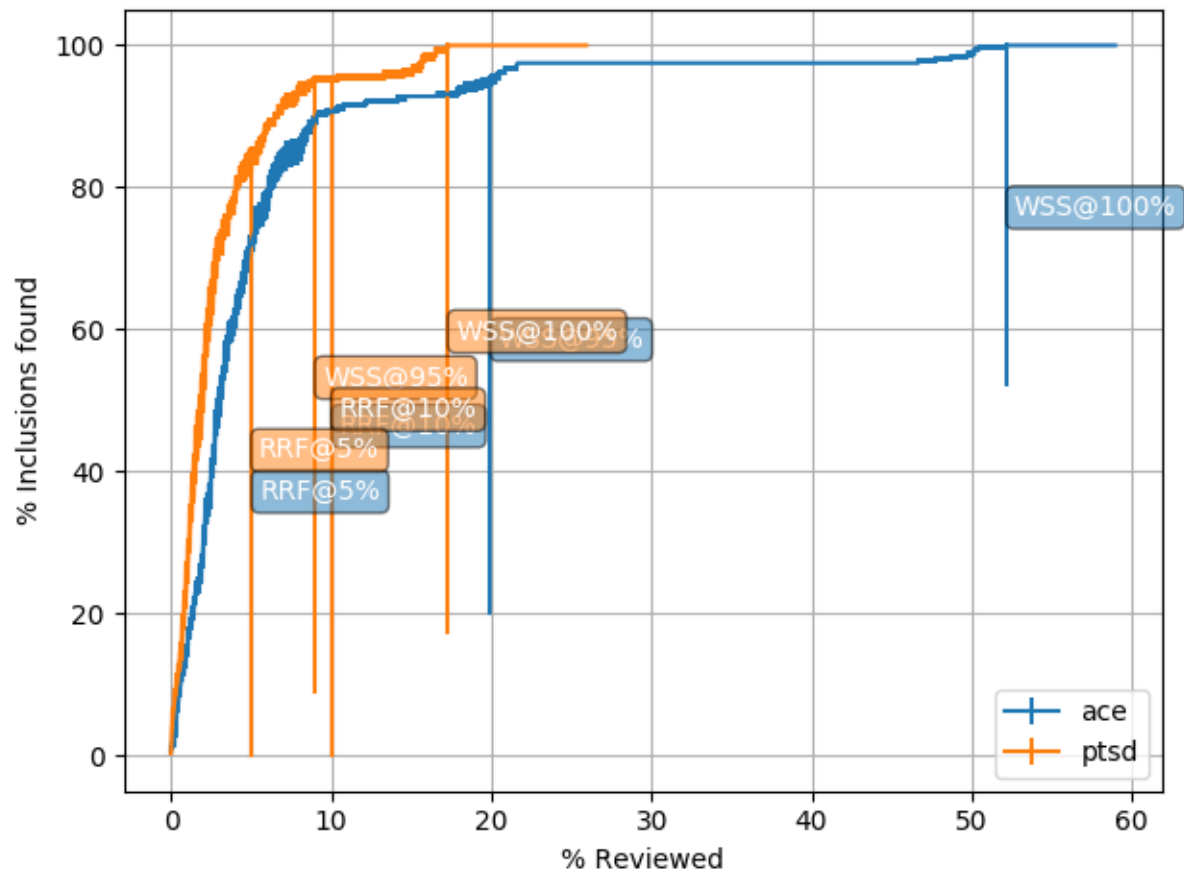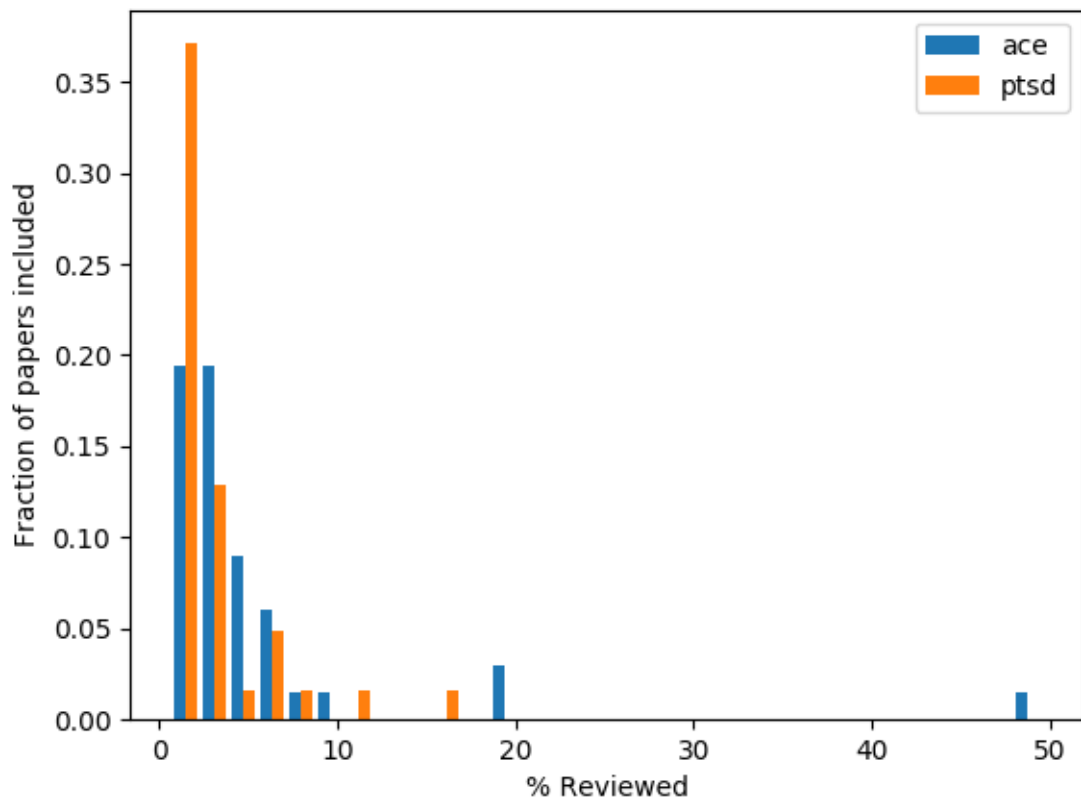
sampling.". Here, *y* is shown on the y-axis, while three values of *z* are plotted as three different lines with the same color. The three values for *z* are 0.1, 0.5 and 2.0.

The quicker the lines touch the black (y=x) line, the better.
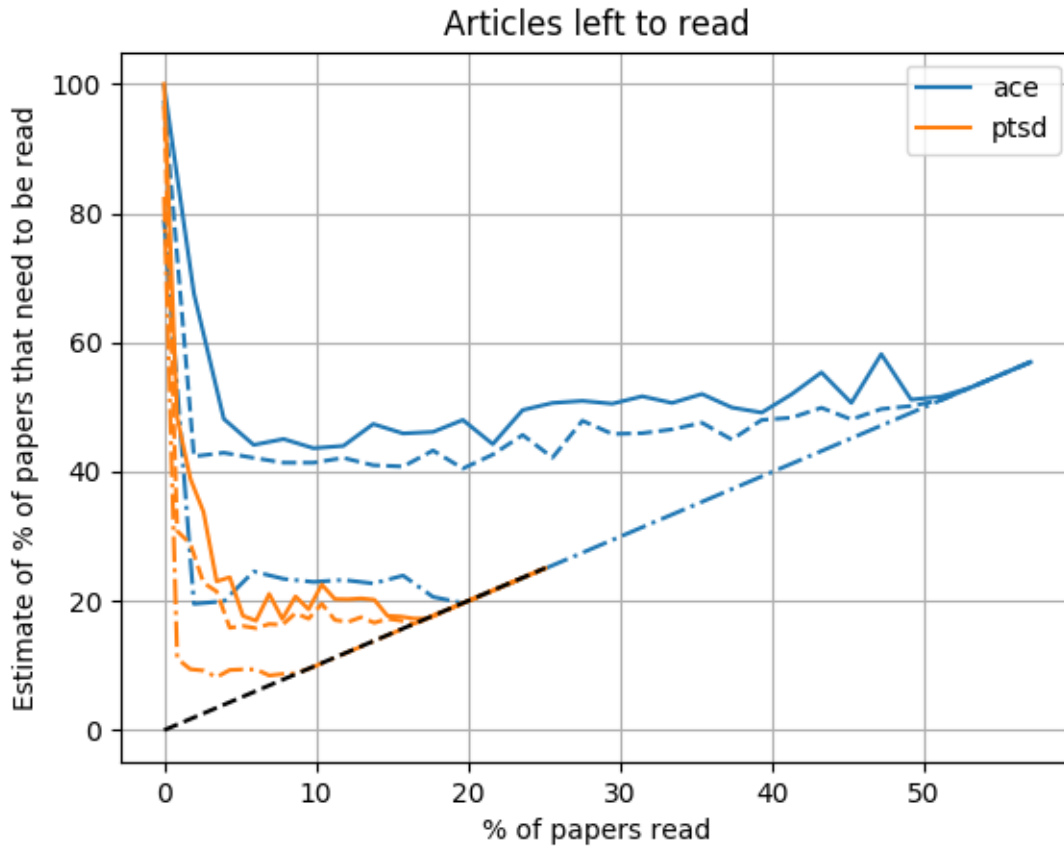

19.3.4 Progression

This figure shows the average inclusion rate as a function of time, number of papers read. The more concentrated on the left, the better. The thick line is the average of individual runs (thin lines). The visualization package will automatically detect which are directories and which are files. The curve is smoothed out by using a Gaussian smoothing algorithm.

19.4 API

To make use of the more advanced features, you can also use the visualization package as a library. The advantage is that you can make more reproducible plots where text, etc. is in the place *you* want it. Examples can be found in module `asreviewcontrib.visualization.quick`. Those are the scripts that are used for the command line interface.

```
from asreviewcontrib.visualization.plot import Plot

with Plot.from_paths(["PATH_1", "PATH_2"]) as plot:
```
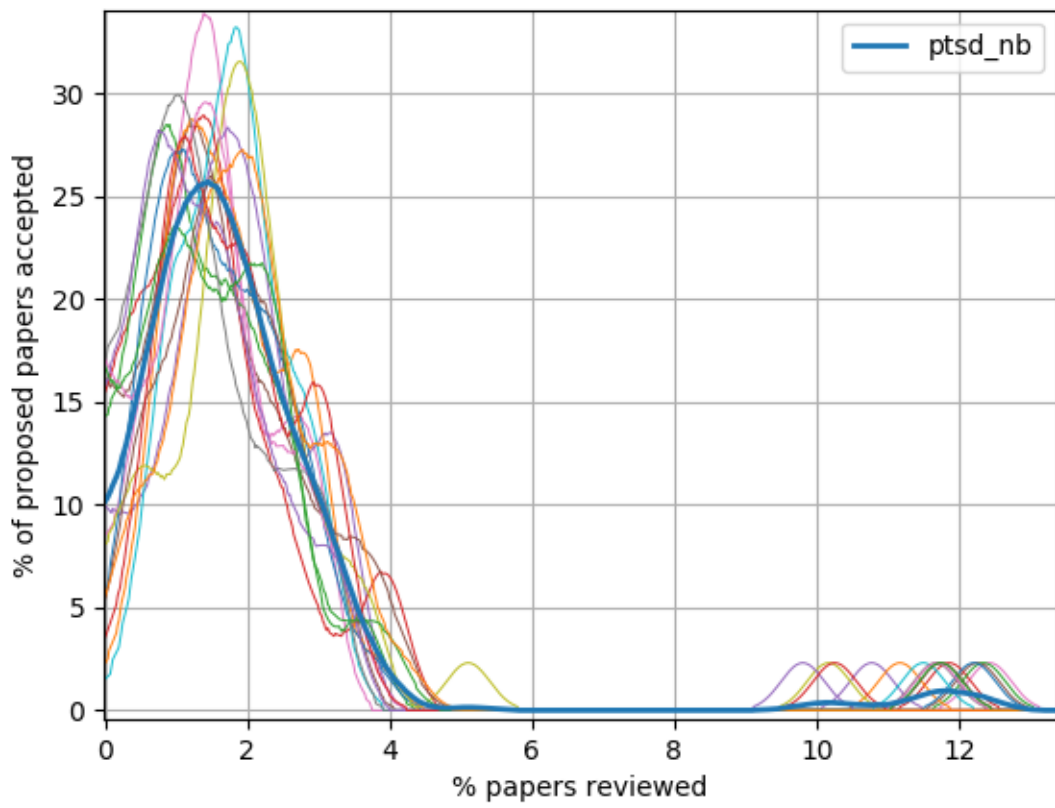
```
inc_plot = plot.new("inclusion")
inc_plot.set_grid()
inc_plot.set_xlim(0, 30)
inc_plot.set_ylim(0, 101)
inc_plot.set_legend()
inc_plot.show()
inc_plot.save("SOME_FILE.png")
```

Of course fill in `PATH_1` and `PATH_2` as the files you would like to plot.

If the customization is not sufficient, you can also directly manipulate the `self.ax` and `self.fig` attributes of the plotting class.

Overview

ASReview provides users an API to interact directly with the underlying ASReview machinery. This provides researchers an interface to study the behavior of algorithms and develop custom workflows. The following figure shows the available interfaces for interacting with the ASReview software:

- The REST API uses a Flask REST API to provide a method to let the React front-end communicate with the backend and algorithms. The REST API is not documented at the moment and should be considered 'internal use only'.

- The *Command Line* is an interface to open ASReview LAB, to run simulations, and more. See *Simulation* for example usage.

- The *API Reference* is a low level Python interface for ASReview. This interface requires detailed knowledge about the workings of the software. There is extensive documentation available on the functions, classes, modules. An outline for usage can be found in *Programming Interface (API)*.

ASReview LAB

REST API

Command Line

Python API
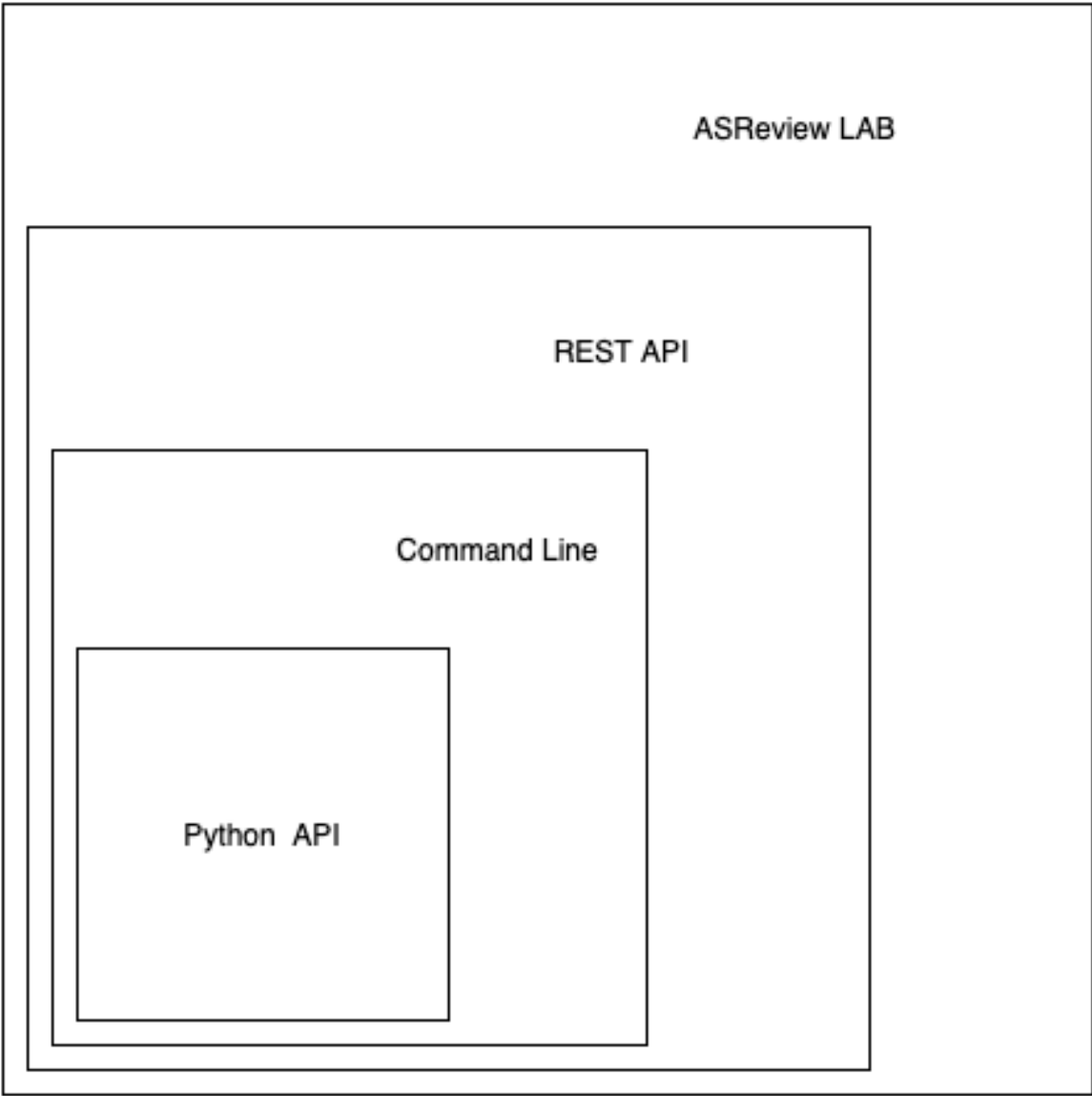
Command Line

ASReview provides a powerful command line interface for running tasks like simulations. For a list of available commands, type `asreview --help`.

21.1 Lab

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

```
asreview lab [options]
```

--ip `IP`
    The IP address the server will listen on.

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

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

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

--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.

--embedding `EMBEDDING_FP`
    File path of embedding matrix. Required for LSTM models.

--clean-project `CLEAN_PROJECT`
    Safe cleanup of temporary files in project.

--clean-all-projects `CLEAN_ALL_PROJECTS`
    Safe cleanup of temporary files in all projects.

--seed SEED
    Seed for the model (classifiers, balance strategies, feature extraction techniques, and query strategies). Use an integer between 0 and 2^32 - 1.

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

21.2 Simulate

asreview simulate measures the performance of the software on existing systematic reviews. The software shows how many papers you could have potentially skipped during the systematic review. You can use *your own labelled dataset*

```
asreview simulate [options] [dataset [dataset ...]]
```

or one of the *benchmark-datasets* (see index.csv for dataset IDs).

```
asreview simulate [options] benchmark: [dataset_id]
```

Examples:

```
asreview simulate YOUR_DATA.csv --state_file myreview.h5
```

```
asreview simulate benchmark:van_de_Schoot_2017 --state_file myreview.h5
```

dataset
    A dataset to simulate

-m, --model MODEL
    The prediction model for Active Learning. Default: nb. (See available options below: *Classifiers*)

-q, --query_strategy QUERY_STRATEGY
    The query strategy for Active Learning. Default: max. (See available options below: *Query strategies*)

-b, --balance_strategy BALANCE_STRATEGY
    Data rebalancing strategy. Helps against imbalanced datasets with few inclusions and many exclusions. Default: double. (See available options below: *Balance strategies*)

-e, --feature_extraction FEATURE_EXTRACTION
    Feature extraction method. Some combinations of feature extraction method and prediction model are not available. Default: tfidf. (See available options below: *Feature extraction*)

--embedding EMBEDDING_FP
    File path of embedding matrix. Required for LSTM models.

--config_file CONFIG_FILE
    Configuration file with model settingsand parameter values.

--seed SEED
    Seed for the model (classifiers, balance strategies, feature extraction techniques, and query strategies). Use an integer between 0 and 2^32 - 1.

--n_prior_included N_PRIOR_INCLUDED
    The number of prior included papers. Only used when prior_idx is not given. Default 1.

--n_prior_excluded N_PRIOR_EXCLUDED
    The number of prior excluded papers. Only used when prior_idx is not given. Default 1.

--prior_idx [PRIOR_IDX [PRIOR_IDX ...]]
> Prior indices by rownumber (0 is first rownumber).

--prior_record_id [PRIOR_RECORD_ID [PRIOR_RECORD_ID ...]]
> Prior indices by record_id.

New in version 0.15.

--included_dataset [INCLUDED_DATASET [INCLUDED_DATASET ...]]
> A dataset with papers that should be includedCan be used multiple times.

--excluded_dataset [EXCLUDED_DATASET [EXCLUDED_DATASET ...]]
> A dataset with papers that should be excludedCan be used multiple times.

--prior_dataset [PRIOR_DATASET [PRIOR_DATASET ...]]
> A dataset with papers from prior studies.

--state_file STATE_FILE, **-s** STATE_FILE
> Location to store the (active learning) state of the simulation. It is possible to output the state to a JSON file (extension .json) or HDF5 file (extension .h5).

--init_seed INIT_SEED
> Seed for setting the prior indices if the prior_idx option is not used. If the option prior_idx is used with one or more index, this option is ignored.

--n_instances N_INSTANCES
> Number of papers queried each query.Default 1.

--n_queries N_QUERIES
> The number of queries. By default, the program stops after all documents are reviewed or is interrupted by the user.

-n N_PAPERS, **--n_papers** N_PAPERS
> The number of papers to be reviewed. By default, the program stops after all documents are reviewed or is interrupted by the user.

--verbose VERBOSE, **-v** VERBOSE
> Verbosity

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

Note: Some classifiers (models) and feature extraction algorithms require additional dependecies. Use `pip install asreview[all]` to install all additional dependencies at once.

21.2.1 Feature Extraction

Name	Reference	Requires
tfidf	*asreview.models.feature_extraction.Tfidf*	
doc2vec	*asreview.models.feature_extraction.Doc2Vec*	gensim
embedding-idf	*asreview.models.feature_extraction.EmbeddingIdf*	
embedding-lstm	*asreview.models.feature_extraction.EmbeddingLSTM*	
sbert	*asreview.models.feature_extraction.SBERT*	sentence_transformers

21.2.2 Classifiers

Name	Reference	Requires
nb	*asreview.models.classifiers.NaiveBayesClassifier*	
svm	*asreview.models.classifiers.SVMClassifier*	
logistic	*asreview.models.classifiers.LogisticClassifier*	
rf	*asreview.models.classifiers.RandomForestClassifier*	
nn-2-layer	*asreview.models.classifiers.NN2LayerClassifier*	tensorflow
lstm-base	*asreview.models.classifiers.LSTMBaseClassifier*	tensorflow
lstm-pool	*asreview.models.classifiers.LSTMPoolClassifier*	tensorflow

21.2.3 Query Strategies

Name	Reference	Requires
max	*asreview.models.query.MaxQuery*	
random	*asreview.models.query.RandomQuery*	
uncertainty	*asreview.models.query.UncertaintyQuery*	
cluster	*asreview.models.query.ClusterQuery*	

21.2.4 Balance Strategies

Name	Reference	Requires
simple	*asreview.models.balance.SimpleBalance*	
double	*asreview.models.balance.DoubleBalance*	
triple	*asreview.models.balance.TripleBalance*	
undersample	*asreview.models.balance.UndersampleBalance*	

21.3 Simulate-batch

asreview simulate-batch provides the same interface as the **asreview simulate**, but adds an extra option (`--n_runs`) to run a batch of simulation runs with the same configuration.

```
asreview simulate-batch [options] [dataset [dataset ...]]
```

> **Warning:** The behavior of some arguments of **asreview simulate-batch** will differ slightly from **asreview simulate**.

dataset
    A dataset to simulate

--n_runs
    Number of simulation runs.

21.4 Algorithms

New in version 0.14.

asreview algorithms provides an overview of all available active learning model elements (classifiers, query strategies, balance strategies, and feature extraction algorithms) in ASReview.

```
asreview algorithms
```

Note: **asreview algorithms** included models added via extensions. See *Extensions* for more information on extending ASReview with new models via extensions.

Extensions

ASReview has support for extensions, which enable you to integrate your programs with the ASReview framework seamlessly. These extensions can extend the software with new classifiers, query strategies, balance strategies, and feature extraction techniques. It is also possible to extend ASReview with a completely new subcommand (like `lab` or `simulate`).

The extensibility of the framework is provided by the entrypoints of setuptools. You will need to create a package and install it (for example with pip). If you have no experience with creating packages, look at the visualization extension and modify it to suit your needs.

22.1 Usage

This section shows how to use an extension. In this example, the asreview-visualization extension is used. The extension extends ASReview to create basic plots from ASReview state files.

Install the extension with

```
pip install asreview-visualization
```

After installation, the subcommand `plot` is available in the command line. See `asreview -h` for this option.

```
$ asreview -h
usage: asreview [-h] [-V] [subcommand]

Automated Systematic Review (ASReview).

positional arguments:
  subcommand      The subcommand to launch. Available commands:

                  lab [asreview-0.13]
                      Graphical user interface for ASReview.

                  simulate [asreview-0.13]
```

```
                   Simulate the performance of ASReview.

               simulate-batch [asreview-0.13]
                   Parallel simulation for ASReview.

               plot [asreview-visualization-0.2.2]
                   Plotting functionality for logging files produced by ASReview.

optional arguments:
  -h, --help     show this help message and exit
  -V, --version  print the ASR version number and exit
```

With this extension installed, a plot can be made with an ASReview state file. The following example shows how a plot is made of the file example_run_1.h5.

```
asreview plot example_run_1.h5
```

22.2 Create subcommand

Extensions in ASReview are Python packages. Extension packages can extend the subcommands of asreview (see asreview -h) or add new algorithms.

The easiest way to create an extension is by defining a class that can be used as a new entry point for ASReview. This class should inherit from *asreview.entry_points.BaseEntryPoint*. Add the functionality to the class method execute.

```python
from asreview.entry_points import BaseEntryPoint

class ExampleEntryPoint(BaseEntryPoint):

    description = "Description of example extension"
    extension_name = "asreview-example"  # Name of the extension
    version = "1.0"  # Version of the extension in x.y(.z) format.

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

It is strongly recommended to define the attributes description, extension_name, and version.

The class method execute accepts a positional aruguument (argv in this example). First create the functionality you would like to be able to use in any directory. The argument argv are the command line arguments left after removing asreview and the entry point.

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

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

Create a `setup.py` in the root of the package, and set the keyword argument *entry_points* of `setup()` under `asreview.entry_points`, for example:

```
entry_points={
    "asreview.entry_points": [
        "example = asreviewcontrib.example.entrypoint:ExampleEntryPoint",
    ]
}
```

After installing this package. ASReview is extended with the `asreview example` subcommand.

If you are willing to share your work, the easiest way is to upload your package to GitHub and/or PyPi. Users can directly install the extension from these sources.

22.3 Add model

In the ASReview project, an active learning model consists of classifier, query strategy, balance strategy, or feature extraction technique. The easiest way to extend ASReview with a new classifier, query strategy, balance strategy, or feature extraction technique is by using the template Template for extending ASReview. Create a copy of the template and add the new algorithms. It is advised to use the following structure of the package:

```
├── README.md
├── asreviewcontrib
│   └── models
│       ├── classifiers
│       │   ├── __init__.py
│       │   └── example_model.py
│       ├── feature_extraction
│       │   ├── __init__.py
│       │   └── example_feature_extraction.py
│       ├── balance
│       │   ├── __init__.py
│       │   └── example_balance_strategies.py
│       └── query
│           ├── __init__.py
│           └── example_query_strategies.py
├── setup.py
└── tests
```
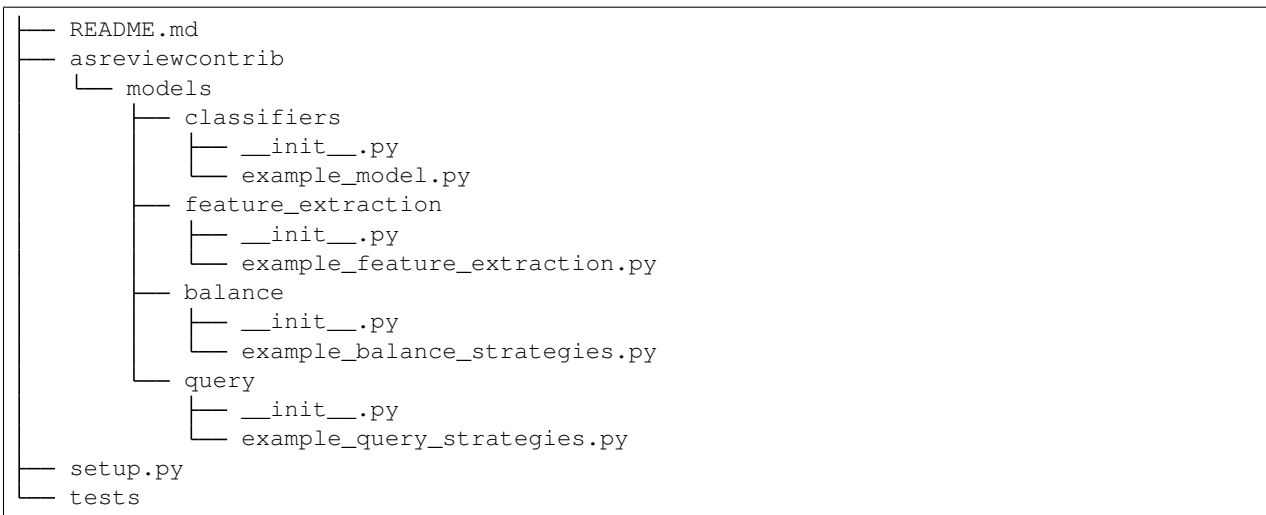
The next step is to add metadata to the setup.py file. Edit the `name` of the package and point the `entry_points` to the models.

```
entry_points={
    'asreview.models.classifiers': [
        'example = asreviewcontrib.models.classifiers.example_model:ExampleClassifier
↪',
    ],
    'asreview.models.feature_extraction': [
        # define feature_extraction algorithms
    ],
    'asreview.models.balance': [
        # define balance_strategy algorithms
    ],
    'asreview.models.query': [
        # define query_strategy algorithms
```

<div align="right">(continues on next page)</div>

```
    ]
},
```

This code registers the model with name `example`.

Install the package with pip:

```
pip install .
```

The new classifier is now available and can be used, for example, in the simulate command line.

```
asreview simulate example_data_file.csv -m example
```

API Reference

23.1 Low level API

class asreview.review.**BaseReview**(*as_data*, *model=None*, *query_model=None*, *balance_model=None*, *feature_model=None*, *n_papers=None*, *n_instances=1*, *n_queries=None*, *start_idx=[]*, *state_file=None*, *log_file=None*)

    Base class for Systematic Review.

    Parameters

- **as_data** (`asreview.ASReviewData`) – The data object which contains the text, labels, etc.

- **model** (`BaseModel`) – Initialized model to fit the data during active learning. See asreview.models.utils.py for possible models.

- **query_model** (`BaseQueryModel`) – Initialized model to query new instances for review, such as random sampling or max sampling. See asreview.query_strategies.utils.py for query models.

- **balance_model** (`BaseBalanceModel`) – Initialized model to redistribute the training data during the active learning process. They might either resample or undersample specific papers.

- **feature_model** (`BaseFeatureModel`) – Feature extraction model that converts texts and keywords to feature matrices.

- **n_papers** (`int`) – Number of papers to review during the active learning process, excluding the number of initial priors. To review all papers, set n_papers to None.

- **n_instances** (`int`) – Number of papers to query at each step in the active learning process.

- **n_queries** (`int`) – Number of steps/queries to perform. Set to None for no limit.

- **start_idx** (*numpy.ndarray*) – Start the simulation/review with these indices. They are assumed to be already labeled. Failing to do so might result bad behaviour.

- **state_file** (*str*) – Path to state file. Replaces log_file argument.

classify(*query_idx*, *inclusions*, *state*, *method=None*)

> Classify new papers and update the training indices.

> It automaticaly updates the state.

> > **Parameters**

> > > - **query_idx** (*list, numpy.ndarray*) – Indices to classify.

> > > - **inclusions** (*list, numpy.ndarray*) – Labels of the query_idx.

> > > - **state** (*BaseLogger*) – Logger to store the classification in.

> > > - **method** (*str*) – If not set to None, all inclusions have this query method.

log_probabilities(*state*)

> Store the modeling probabilities of the training indices and pool indices.

n_pool()

> Number of indices left in the pool.

> > **Returns** *int* – Number of indices left in the pool.

query(*n_instances*, *query_model=None*)

> Query records from pool.

> > **Parameters**

> > > - **n_instances** (*int*) – Batch size of the queries, i.e. number of records to be queried.

> > > - **query_model** (*BaseQueryModel*) – Query strategy model to use. If None, the query model of the reviewer is used.

> > **Returns** *numpy.ndarray* – Indices of records queried.

review(**args*, ***kwargs*)

> Do the systematic review, writing the results to the state file.

> > **Parameters**

> > > - **stop_after_class** (*bool*) – When to stop; if True stop after classification step, otherwise stop after training step.

> > > - **instant_save** (*bool*) – If True, save results after each single classification.

settings

> Get an ASReview settings object

statistics()

> Get statistics on the current state of the review.

> > **Returns** *dict* – A dictonary with statistics like n_included and last_inclusion.

train()

> Train the model.

class asreview.**ReviewSimulate**(*as_data*, **args*, *n_prior_included=0*, *n_prior_excluded=0*, *prior_idx=None*, *init_seed=None*, ***kwargs*)

> ASReview Simulation mode class.

> > **Parameters**

- **as_data** (`asreview.ASReviewData`) – The data object which contains the text, labels, etc.

- **model** (*BaseModel*) – Initialized model to fit the data during active learning. See asreview.models.utils.py for possible models.

- **query_model** (*BaseQueryModel*) – Initialized model to query new instances for review, such as random sampling or max sampling. See asreview.query_strategies.utils.py for query models.

- **balance_model** (*BaseBalanceModel*) – Initialized model to redistribute the training data during the active learning process. They might either resample or undersample specific papers.

- **feature_model** (*BaseFeatureModel*) – Feature extraction model that converts texts and keywords to feature matrices.

- **n_prior_included** (`int`) – Sample n prior included papers.

- **n_prior_excluded** (`int`) – Sample n prior excluded papers.

- **prior_idx** (`int`) – Prior indices by row number.

- **n_papers** (`int`) – Number of papers to review during the active learning process, excluding the number of initial priors. To review all papers, set n_papers to None.

- **n_instances** (`int`) – Number of papers to query at each step in the active learning process.

- **n_queries** (`int`) – Number of steps/queries to perform. Set to None for no limit.

- **start_idx** (`numpy.ndarray`) – Start the simulation/review with these indices. They are assumed to be already labeled. Failing to do so might result bad behaviour.

- **init_seed** (`int`) – Seed for setting the prior indices if the –prior_idx option is not used. If the option prior_idx is used with one or more index, this option is ignored.

- **state_file** (`str`) – Path to state file. Replaces log_file argument.

classify (*query_idx*, *inclusions*, *state*, *method=None*)
Classify new papers and update the training indices.

It automaticaly updates the state.

> **Parameters**
>
> - **query_idx** (`list,` `numpy.ndarray`) – Indices to classify.
>
> - **inclusions** (`list,` `numpy.ndarray`) – Labels of the query_idx.
>
> - **state** (*BaseLogger*) – Logger to store the classification in.
>
> - **method** (`str`) – If not set to None, all inclusions have this query method.

log_probabilities (*state*)
Store the modeling probabilities of the training indices and pool indices.

n_pool ()
Number of indices left in the pool.

> **Returns** *int* – Number of indices left in the pool.

query (*n_instances*, *query_model=None*)
Query records from pool.

> **Parameters**

- **n_instances** (*int*) – Batch size of the queries, i.e. number of records to be queried.

- **query_model** (*BaseQueryModel*) – Query strategy model to use. If None, the query model of the reviewer is used.

     Returns *numpy.ndarray* – Indices of records queried.

review(**args*, ***kwargs*)
    Do the systematic review, writing the results to the state file.

        Parameters

- **stop_after_class** (*bool*) – When to stop; if True stop after classification step, otherwise stop after training step.

- **instant_save** (*bool*) – If True, save results after each single classification.

settings
    Get an ASReview settings object

statistics()
    Get statistics on the current state of the review.

        Returns *dict* – A dictonary with statistics like n_included and last_inclusion.

train()
    Train the model.

23.2 Classifiers

class asreview.models.classifiers.**NaiveBayesClassifier**(*alpha=3.822*)
    Naive Bayes classifier

    Naive Bayes classifier. Only works in combination with the *asreview.models.feature_extraction.Tfidf* feature extraction model. Though relatively simplistic, seems to work quite well on a wide range of datasets.

    The naive Bayes classifier is an implementation based on the sklearn multinomial naive Bayes classifier.

        Parameters alpha (*float, default=3.822*) – Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

default_param
    Get the default parameters of the model.

        Returns *dict* – Dictionary with parameter: default value

fit(*X*, *y*)
    Fit the model to the data.

        Parameters

- **X** (*numpy.ndarray*) – Feature matrix to fit.

- **y** (*numpy.ndarray*) – Labels for supervised learning.

full_hyper_space()
    Get a hyperparameter space to use with hyperopt.

        Returns *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.

param
> Get the (assigned) parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: current value.

predict_proba(*X*)
> Get the inclusion probability for each sample.
>
> > **Parameters X** (*numpy.ndarray*) – Feature matrix to predict.
>
> > **Returns** *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

class asreview.models.classifiers.**RandomForestClassifier**(*n_estimators=100, max_features=10, class_weight=1.0, random_state=None*)

Random Forest classifier

The Random Forest classifier is an implementation based on the sklearn Random Forest classifier.

> **Parameters**
>
> - **n_estimators** (*int, default=100*) – The number of trees in the forest.
> - **max_features** (*int, default=10*) – Number of features in the model.
> - **class_weight** (*float, default=1.0*) – Class weight of the inclusions.
> - **random_state** (*int or RandomState, default=None*) – Controls both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node.

default_param
> Get the default parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: default value

fit(*X*, *y*)
> Fit the model to the data.
>
> > **Parameters**
> >
> > - **X** (*numpy.ndarray*) – Feature matrix to fit.
> > - **y** (*numpy.ndarray*) – Labels for supervised learning.

full_hyper_space()
> Get a hyperparameter space to use with hyperopt.
>
> > **Returns** *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.

param
> Get the (assigned) parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: current value.

predict_proba(*X*)
> Get the inclusion probability for each sample.
>
> > **Parameters X** (*numpy.ndarray*) – Feature matrix to predict.
>
> > **Returns** *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

---

class `asreview.models.classifiers.`**`SVMClassifier`**(*gamma='auto', class_weight=0.249, C=15.4, kernel='linear', random_state=None*)

> Support Vector Machine classifier
>
> The Support Vector Machine classifier is an implementation based on the sklearn Support Vector Machine classifier.
>
> > **Parameters**
> >
> > - **gamma** (*str*) – Gamma parameter of the SVM model.
> >
> > - **class_weight** (*float*) – class_weight of the inclusions.
> >
> > - **C** (*float*) – C parameter of the SVM model.
> >
> > - **kernel** (*str*) – SVM kernel type.
> >
> > - **random_state** (*int, RandomState*) – State of the RNG.
>
> **`default_param`**
> > Get the default parameters of the model.
> >
> > > **Returns** *dict* – Dictionary with parameter: default value
>
> **`fit`**(*X, y*)
> > Fit the model to the data.
> >
> > > **Parameters**
> > >
> > > - **X** (*numpy.ndarray*) – Feature matrix to fit.
> > >
> > > - **y** (*numpy.ndarray*) – Labels for supervised learning.
>
> **`full_hyper_space`**()
> > Get a hyperparameter space to use with hyperopt.
> >
> > > **Returns** *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.
>
> **`param`**
> > Get the (assigned) parameters of the model.
> >
> > > **Returns** *dict* – Dictionary with parameter: current value.
>
> **`predict_proba`**(*X*)
> > Get the inclusion probability for each sample.
> >
> > > **Parameters** **X** (*numpy.ndarray*) – Feature matrix to predict.
> > >
> > > **Returns** *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

class `asreview.models.classifiers.`**`LogisticClassifier`**(*C=1.0, class_weight=1.0, random_state=None, n_jobs=1*)

> Logistic regressions classifier
>
> The Logistic regressions classifier is an implementation based on the sklearn Logistic regressions classifier.
>
> > **Parameters**
> >
> > - **C** (*float*) – Parameter inverse to the regularization strength of the model.
> >
> > - **class_weight** (*float*) – Class weight of the inclusions.
> >
> > - **random_state** (*int, RandomState*) – Random state for the model.
> >
> > - **n_jobs** (*int*) – Number of CPU cores used.

default_param
>   Get the default parameters of the model.

>   **Returns**   *dict* – Dictionary with parameter: default value

fit(*X*, *y*)
>   Fit the model to the data.

>   **Parameters**

>   - **X** (*numpy.ndarray*) – Feature matrix to fit.

>   - **y** (*numpy.ndarray*) – Labels for supervised learning.

full_hyper_space()
>   Get a hyperparameter space to use with hyperopt.

>   **Returns**   *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.

param
>   Get the (assigned) parameters of the model.

>   **Returns**   *dict* – Dictionary with parameter: current value.

predict_proba(*X*)
>   Get the inclusion probability for each sample.

>   **Parameters**   **X** (*numpy.ndarray*) – Feature matrix to predict.

>   **Returns**   *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

class asreview.models.classifiers.**LSTMBaseClassifier**(*embedding_matrix=None, backwards=True, dropout=0.4, optimizer='rmsprop', lstm_out_width=20, learn_rate=1.0, dense_width=128, verbose=0, batch_size=32, epochs=35, shuffle=False, class_weight=30.0*)

LSTM base classifier.

LSTM model that consists of an embedding layer, LSTM layer with one output, dense layer, and a single sigmoid output node. Use the *asreview.models.feature_extraction.EmbeddingLSTM* feature extraction method. Currently not so well optimized and slow.

Note: This model requires tensorflow to be installed. Use pip install tensorflow or install all optional ASReview dependencies with pip install asreview[all]

>   **Parameters**

>   - **embedding_matrix** (*numpy.ndarray*) – Embedding matrix to use with LSTM model.

>   - **backwards** (*bool*) – Whether to have a forward or backward LSTM.

>   - **dropout** (*float*) – Value in [0, 1.0) that gives the dropout and recurrent dropout rate for the LSTM model.

>   - **optimizer** (*str*) – Optimizer to use.

- **lstm_out_width** (*int*) – Output width of the LSTM.

- **learn_rate** (*float*) – Learn rate multiplier of default learning rate.

- **dense_width** (*int*) – Size of the dense layer of the model.

- **verbose** (*int*) – Verbosity.

- **batch_size** (*int*) – Size of the batch size for the LSTM model.

- **epochs** (*int*) – Number of epochs to train the LSTM model.

- **shuffle** (*bool*) – Whether to shuffle the data before starting to train.

- **class_weight** (*float*) – Class weight for the included papers.

default_param
    Get the default parameters of the model.

    Returns *dict* – Dictionary with parameter: default value

fit(*X*, *y*)
    Fit the model to the data.

    Parameters

    - **X** (*numpy.ndarray*) – Feature matrix to fit.

    - **y** (*numpy.ndarray*) – Labels for supervised learning.

full_hyper_space()
    Get a hyperparameter space to use with hyperopt.

    Returns *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list
        of choices, store the choices there.

param
    Get the (assigned) parameters of the model.

    Returns *dict* – Dictionary with parameter: current value.

predict_proba(*X*)
    Get the inclusion probability for each sample.

    Parameters X (*numpy.ndarray*) – Feature matrix to predict.

    Returns *numpy.ndarray* – Array with the probabilities for each class, with two columns (class
        0, and class 1) and the number of samples rows.

class asreview.models.classifiers.**LSTMPoolClassifier**(*embedding_matrix=None,
                                                            backwards=True, dropout=0.4,
                                                            optimizer='rmsprop',
                                                            lstm_out_width=20,
                                                            lstm_pool_size=128,
                                                            learn_rate=1.0, ver-
                                                            bose=0, batch_size=32,
                                                            epochs=35, shuffle=False,
                                                            class_weight=30.0*)

LSTM pool classifier.

LSTM model that consists of an embedding layer, LSTM layer with many outputs, max pooling layer, and a
single sigmoid output node. Use the *asreview.models.feature_extraction.EmbeddingLSTM*
feature extraction method. Currently not so well optimized and slow.

Note: This model requires `tensorflow` to be installed. Use `pip install tensorflow` or install all optional ASReview dependencies with `pip install asreview[all]`

> **Parameters**
>
> - **embedding_matrix** (*[numpy.ndarray](#)*) – Embedding matrix to use with LSTM model.
>
> - **backwards** (*[bool](#)*) – Whether to have a forward or backward LSTM.
>
> - **dropout** (*[float](#)*) – Value in [0, 1.0) that gives the dropout and recurrent dropout rate for the LSTM model.
>
> - **optimizer** (*[str](#)*) – Optimizer to use.
>
> - **lstm_out_width** (*[int](#)*) – Output width of the LSTM.
>
> - **lstm_pool_size** (*[int](#)*) – Size of the pool, must be a divisor of max_sequence_length.
>
> - **learn_rate** (*[float](#)*) – Learn rate multiplier of default learning rate.
>
> - **verbose** (*[int](#)*) – Verbosity.
>
> - **batch_size** (*[int](#)*) – Size of the batch size for the LSTM model.
>
> - **epochs** (*[int](#)*) – Number of epochs to train the LSTM model.
>
> - **shuffle** (*[bool](#)*) – Whether to shuffle the data before starting to train.
>
> - **class_weight** (*[float](#)*) – Class weight for the included papers.

default_param
> Get the default parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: default value

fit (*X*, *y*)
> Fit the model to the data.
>
> > **Parameters**
> >
> > - **X** (*[numpy.ndarray](#)*) – Feature matrix to fit.
> >
> > - **y** (*[numpy.ndarray](#)*) – Labels for supervised learning.

full_hyper_space()
> Get a hyperparameter space to use with hyperopt.
>
> > **Returns** *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.

param
> Get the (assigned) parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: current value.

predict_proba (*X*)
> Get the inclusion probability for each sample.
>
> > **Parameters** **X** (*[numpy.ndarray](#)*) – Feature matrix to predict.
> >
> > **Returns** *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

---

class asreview.models.classifiers.**NN2LayerClassifier**(*dense_width=128*, *optimizer='rmsprop'*, *learn_rate=1.0*, *regularization=0.01*, *verbose=0*, *epochs=35*, *batch_size=32*, *shuffle=False*, *class_weight=30.0*)

Dense neural network classifier.

Neural network with two hidden, dense layers of the same size.

Recommended feature extraction model is *asreview.models.feature_extraction.Doc2Vec*.

Note: This model requires tensorflow to be installed. Use pip install tensorflow or install all optional ASReview dependencies with pip install asreview[all]

> **Warning:** Might crash on some systems with limited memory in combination with *asreview.models.* *feature_extraction.Tfidf*.

Parameters

- **dense_width** (*int*) – Size of the dense layers.

- **optimizer** (*str*) – Name of the Keras optimizer.

- **learn_rate** (*float*) – Learning rate multiplier of the default learning rate.

- **regularization** (*float*) – Strength of the regularization on the weights and biases.

- **verbose** (*int*) – Verbosity of the model mirroring the values for Keras.

- **epochs** (*int*) – Number of epochs to train the neural network.

- **batch_size** (*int*) – Batch size used for the neural network.

- **shuffle** (*bool*) – Whether to shuffle the training data prior to training.

- **class_weight** (*float*) – Class weights for inclusions (1's).

default_param
> Get the default parameters of the model.
>
> > **Returns** *dict* – Dictionary with parameter: default value

fit(*X*, *y*)
> Fit the model to the data.
>
> > **Parameters**
> >
> > - **X** (*numpy.ndarray*) – Feature matrix to fit.
> >
> > - **y** (*numpy.ndarray*) – Labels for supervised learning.

full_hyper_space()
> Get a hyperparameter space to use with hyperopt.
>
> > **Returns** *dict, dict* – Parameter space. Parameter choices; in case of hyperparameters with a list of choices, store the choices there.

param
> Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

predict_proba(*X*)

Get the inclusion probability for each sample.

> **Parameters** **X** (`numpy.ndarray`) – Feature matrix to predict.

> **Returns** *numpy.ndarray* – Array with the probabilities for each class, with two columns (class 0, and class 1) and the number of samples rows.

`asreview.models.classifiers.`**list_classifiers**()

List available classifiers.

> **Returns** *list* – Names of available classifiers in alphabetical order.

`asreview.models.classifiers.`**get_classifier**(*name*, **args*, *random_state=None*, ***kwargs*)

Get an instance of a model from a string.

> **Parameters**
>
> - **name** (`str`) – Name of the model.
> - ***args** – Arguments for the model.
> - ****kwargs** – Keyword arguments for the model.

> **Returns** *BaseFeatureExtraction* – Initialized instance of classifier.

`asreview.models.classifiers.`**get_classifier_class**(*name*)

Get class of model from string.

> **Parameters** **name** (`str`) – Name of the model, e.g. 'svm', 'nb' or 'lstm-pool'.

> **Returns** *BaseModel* – Class corresponding to the name.

23.3 Query

class `asreview.models.query.`**MaxQuery**

Maximum sampling query strategy.

Choose the most likely samples to be included according to the model.

default_param

Get the default parameters of the model.

> **Returns** *dict* – Dictionary with parameter: default value

param

Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

query(*X*, *classifier*, *pool_idx=None*, *n_instances=1*, *shared={}*)

Query method for strategies which use class probabilities.

class `asreview.models.query.`**MixedQuery**(*strategy_1='max'*, *strategy_2='random'*, *mix_ratio=0.95*, *random_state=None*, ***kwargs*)

Class for mixed query strategy.

The idea is to use two different query strategies at the same time with a ratio of one to the other. A mix of two query strategies is used. For example mixing max and random sampling with a mix ratio of 0.95 would mean that at each query 95% of the instances would be sampled with the max query strategy after which the remaining 5% would be sampled with the random query strategy. It would be called the *max_random* query strategy. Every combination of primitive query strategy is possible.

Parameters

- **strategy_1** (*str*) – Name of the first query strategy.

- **strategy_2** (*str*) – Name of the second query strategy.

- **mix_ratio** (*float*) – Portion of queries done by the first strategy. So a mix_ratio of 0.95 means that 95% of the time query strategy 1 is used and 5% of the time query strategy 2.

- ****kwargs** (*dict*) – Keyword arguments for the two strategy. To specify which of the strategies the argument is for, prepend with the name of the query strategy and an underscore, e.g. 'max' for maximal sampling.

default_param

Get the default parameters of the model.

Returns *dict* – Dictionary with parameter: default value

name

str(object=") -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

param

Get the (assigned) parameters of the model.

Returns *dict* – Dictionary with parameter: current value.

query (*X*, *classifier*, *pool_idx=None*, *n_instances=1*, *shared={}*)

Query new instances.

Parameters

- **X** (*numpy.ndarray*) – Feature matrix to choose samples from.

- **classifier** (*SKLearnModel*) – Trained classifier to compute probabilities if they are necessary.

- **pool_idx** (*numpy.ndarray*) – Indices of samples that are still in the pool.

- **n_instances** (*int*) – Number of instances to query.

- **shared** (*dict*) – Dictionary for exchange between query strategies and others. It is mainly used to store the current class probabilities, and the source of the queries; which query strategy has produced which index.

class asreview.models.query.**UncertaintyQuery**

Maximum uncertainty query strategy.

Choose the most uncertain samples according to the model (i.e. closest to 0.5 probability). Doesn't work very well in the case of LSTM's, since the probabilities are rather arbitrary.

default_param

Get the default parameters of the model.

Returns *dict* – Dictionary with parameter: default value

param

Get the (assigned) parameters of the model.

Returns *dict* – Dictionary with parameter: current value.

query (*X*, *classifier*, *pool_idx=None*, *n_instances=1*, *shared={}*)
> Query method for strategies which use class probabilities.

class asreview.models.query.**RandomQuery** (*random_state=None*)
> Random sampling query strategy.

Randomly select samples with no regard to model assigned probabilities.

> **Warning:** Selecting this option means your review is not going to be accelerated by ASReview.

default_param
> Get the default parameters of the model.

> > **Returns** *dict* – Dictionary with parameter: default value

param
> Get the (assigned) parameters of the model.

> > **Returns** *dict* – Dictionary with parameter: current value.

query (*X*, *classifier*, *pool_idx=None*, *n_instances=1*, *shared={}*)
> Query method for strategies which do not use class probabilities

class asreview.models.query.**ClusterQuery** (*cluster_size=350*, *update_interval=200*, *random_state=None*)
> Query strategy using clustering algorithms.

Use clustering after feature extraction on the dataset. Then the highest probabilities within random clusters are sampled.

> **Parameters**

> > • **cluster_size** ([*int*](#)) – Size of the clusters to be made. If the size of the clusters is smaller than the size of the pool, fall back to max sampling.

> > • **update_interval** ([*int*](#)) – Update the clustering every x instances.

> > • **random_state** (*int, RandomState*) – State/seed of the RNG.

default_param
> Get the default parameters of the model.

> > **Returns** *dict* – Dictionary with parameter: default value

param
> Get the (assigned) parameters of the model.

> > **Returns** *dict* – Dictionary with parameter: current value.

query (*X*, *classifier*, *pool_idx=None*, *n_instances=1*, *shared={}*)
> Query method for strategies which use class probabilities.

asreview.models.query.**list_query_strategies** ()
> List available query strategies.

This excludes all possible mixed query strategies.

> **Returns** *list* – Names of available query strategies in alphabetical order.

asreview.models.query.**get_query_model** (*name*, **args*, *random_state=None*, ***kwargs*)
> Get an instance of the query strategy.

> **Parameters**

- **name** (*str*) – Name of the query strategy.

- ***args** – Arguments for the model.

- ****kwargs** – Keyword arguments for the model.

Returns *asreview.query.base.BaseQueryModel* – Initialized instance of query strategy.

asreview.models.query.**get_query_class**(*name*)

Get class of query strategy from its name.

Parameters name (*str*) – Name of the query strategy, e.g. 'max', 'uncertainty', 'random. A special mixed query strategy is als possible. The mix is denoted by an underscore: 'max_random' or 'max_uncertainty'.

Returns *class* – Class corresponding to the name name.

23.4 Balance

class asreview.models.balance.**SimpleBalance**

No balancing.

Use all training data.

default_param

Get the default parameters of the model.

Returns *dict* – Dictionary with parameter: default value

param

Get the (assigned) parameters of the model.

Returns *dict* – Dictionary with parameter: current value.

sample(*X*, *y*, *train_idx*, *shared*)

Function that does not resample the training set.

Parameters

- **X** (*numpy.ndarray*) – Complete matrix of all samples.

- **y** (*numpy.ndarray*) – Classified results of all samples.

- **extra_vars** (*dict:*) – Extra variables that can be passed around between functions.

Returns

- *numpy.ndarray* – Training samples.

- *numpy.ndarray* – Classification of training samples.

class asreview.models.balance.**DoubleBalance**(*a=2.155*, *alpha=0.94*, *b=0.789*, *beta=1.0*, *random_state=None*)

Dynamic Resampling balance strategy.

Class to get the two way rebalancing function and arguments. It super samples ones depending on the number of 0's and total number of samples in the training data.

Parameters

- **a** (*float*) – Governs the weight of the 1's. Higher values mean linearly more 1's in your training sample.

- **alpha** (*float*) – Governs the scaling the weight of the 1's, as a function of the ratio of ones to zeros. A positive value means that the lower the ratio of zeros to ones, the higher the weight of the ones.

- **b** (*float*) – Governs how strongly we want to sample depending on the total number of samples. A value of 1 means no dependence on the total number of samples, while lower values mean increasingly stronger dependence on the number of samples.

- **beta** (*float*) – Governs the scaling of the weight of the zeros depending on the number of samples. Higher values means that larger samples are more strongly penalizing zeros.

default_param
    Get the default parameters of the model.

> **Returns** *dict* – Dictionary with parameter: default value

param
    Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

sample(*X*, *y*, *train_idx*, *shared*)
    Resample the training data.

> **Parameters**
>
> - **X** (*numpy.ndarray*) – Complete feature matrix.
>
> - **y** (*numpy.ndarray*) – Labels for all papers.
>
> - **train_idx** (*numpy.ndarray*) – Training indices, that is all papers that have been reviewed.
>
> - **shared** (*dict*) – Dictionary to share data between balancing models and other models.
>
> **Returns** *numpy.ndarray,numpy.ndarray* – X_train, y_train: the resampled matrix, labels.

class asreview.models.balance.**TripleBalance**(*a=2.155*, *alpha=0.94*, *b=0.789*, *beta=1.0*, *c=0.835*, *gamma=2.0*, *shuffle=True*, *random_state=None*)
    Triple balance strategy.

This divides the training data into three sets: included papers, excluded papers found with random sampling and papers found with max sampling. They are balanced according to formulas depending on the percentage of papers read in the dataset, the number of papers with random/max sampling etc. Works best for stochastic training algorithms. Reduces to both full sampling and undersampling with corresponding parameters.

> **Parameters**
>
> - **a** (*float*) – Governs the weight of the 1's. Higher values mean linearly more 1's in your training sample.
>
> - **alpha** (*float*) – Governs the scaling the weight of the 1's, as a function of the ratio of ones to zeros. A positive value means that the lower the ratio of zeros to ones, the higher the weight of the ones.
>
> - **b** (*float*) – Governs how strongly we want to sample depending on the total number of samples. A value of 1 means no dependence on the total number of samples, while lower values mean increasingly stronger dependence on the number of samples.
>
> - **beta** (*float*) – Governs the scaling of the weight of the zeros depending on the number of samples. Higher values means that larger samples are more strongly penalizing zeros.
>
> - **c** (*float*) – Value between one and zero that governs the weight of samples done with maximal sampling. Higher values mean higher weight.

- **gamma** (*float*) – Governs the scaling of the weight of the max samples as a function of the % of papers read. Higher values mean stronger scaling.

default_param
    Get the default parameters of the model.

        Returns *dict* – Dictionary with parameter: default value

param
    Get the (assigned) parameters of the model.

        Returns *dict* – Dictionary with parameter: current value.

sample (*X*, *y*, *train_idx*, *shared*)
    Resample the training data.

        Parameters

- **X** (*numpy.ndarray*) – Complete feature matrix.

- **y** (*numpy.ndarray*) – Labels for all papers.

- **train_idx** (*numpy.ndarray*) – Training indices, that is all papers that have been reviewed.

- **shared** (*dict*) – Dictionary to share data between balancing models and other models.

        Returns *numpy.ndarray,numpy.ndarray* – X_train, y_train: the resampled matrix, labels.

class asreview.models.balance.**UndersampleBalance** (*ratio=1.0*, *random_state=None*)
    Balancing class that undersamples the data with a given ratio.

    This undersamples the data, leaving out excluded papers so that the included and excluded papers are in some particular ratio (closer to one).

    Parameters **ratio** (*double*) – Undersampling ratio of the zero's. If for example we set a ratio of 0.25, we would sample only a quarter of the zeros and all the ones.

default_param
    Get the default parameters of the model.

        Returns *dict* – Dictionary with parameter: default value

param
    Get the (assigned) parameters of the model.

        Returns *dict* – Dictionary with parameter: current value.

sample (*X*, *y*, *train_idx*, *shared*)
    Resample the training data.

        Parameters

- **X** (*numpy.ndarray*) – Complete feature matrix.

- **y** (*numpy.ndarray*) – Labels for all papers.

- **train_idx** (*numpy.ndarray*) – Training indices, that is all papers that have been reviewed.

- **shared** (*dict*) – Dictionary to share data between balancing models and other models.

        Returns *numpy.ndarray,numpy.ndarray* – X_train, y_train: the resampled matrix, labels.

asreview.models.balance.**list_balance_strategies**()
    List available balancing strategies.

Returns *list* – Names of available balance strategies in alphabetical order.

asreview.models.balance.**get_balance_model**(*name*, **args*, *random_state=None*, ***kwargs*)
    Get an instance of a balance model from a string.

> **Parameters**
>
> - **name** (`str`) – Name of the balance model.
>
> - ***args** – Arguments for the balance model.
>
> - ****kwargs** – Keyword arguments for the balance model.
>
> **Returns** *BaseFeatureExtraction* – Initialized instance of features extraction algorithm.

asreview.models.balance.**get_balance_class**(*name*)
    Get class of balance model from string.

> **Parameters** **name** (`str`) – Name of the model, e.g. 'simple', 'double' or 'undersample'.
>
> **Returns** *BaseBalanceModel* – Class corresponding to the name.

23.5 Feature extraction

class asreview.models.feature_extraction.**Tfidf**(**args*, *ngram_max=1*, *stop_words='english'*, ***kwargs*)
    Class to apply TF-IDF to texts.

Use the standard TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction from SKLearn. Gives a sparse matrix as output. Works well in combination with asreview.models.NBModel and other fast training models (given that the features vectors are relatively wide).

> **Parameters**
>
> - **ngram_max** (`int`) – Can use up to ngrams up to ngram_max. For example in the case of ngram_max=2, monograms and bigrams could be used.
>
> - **stop_words** (`str`) – When set to 'english', use stopwords. If set to None or 'none', do not use stop words.

default_param
    Get the default parameters of the model.

> **Returns** *dict* – Dictionary with parameter: default value

fit(*texts*)
    Fit the model to the texts.

It is not always necessary to implement this if there's not real fitting being done.

> **Parameters** **texts** (`numpy.ndarray`) – Texts to be fitted.

fit_transform(*texts*, *titles=None*, *abstracts=None*, *keywords=None*)
    Fit and transform a list of texts.

> **Parameters** **texts** (`numpy.ndarray`) – A sequence of texts to be transformed. They are not yet tokenized.
>
> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

param
    Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

---

transform(*texts*)
> Transform a list of texts.

> > **Parameters texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.

> > **Returns** *numpy.ndarray* – Feature matrix representing the texts.

class asreview.models.feature_extraction.**Doc2Vec**(**args*, *vector_size=40*, *epochs=33*, *min_count=1*, *n_jobs=1*, *window=7*, *dm_concat=0*, *dm=2*, *dbow_words=0*, ***kwargs*)

Base class for doc2vec feature extraction.

Feature extraction method provided by the gensim package. It takes relatively long to create a feature matrix with this method. However, this only has to be done once per simulation/review. The upside of this method is the dimension- reduction that generally takes place, which makes the modelling quicker.

Note: This feature extraction algorithm requires gensim to be installed. Use pip install gensim or install all optional ASReview dependencies with pip install asreview[all]

> **Parameters**

> > - **vector_size** (*int*) – Output size of the vector.

> > - **epochs** (*int*) – Number of epochs to train the doc2vec model.

> > - **min_count** (*int*) – Minimum number of occurences for a word in the corpus for it to be included in the model.

> > - **n_jobs** (*int*) – Number of threads to train the model with.

> > - **window** (*int*) – Maximum distance over which word vectors influence each other.

> > - **dm_concat** (*int*) – Whether to concatenate word vectors or not. See paper for more detail.

> > - **dm** (*int*) – Model to use. 0: Use distribute bag of words (DBOW). 1: Use distributed memory (DM). 2: Use both of the above with half the vector size and concatenate them.

> > - **dbow_words** (*int*) – Whether to train the word vectors using the skipgram method.

default_param
> Get the default parameters of the model.

> > **Returns** *dict* – Dictionary with parameter: default value

fit(*texts*)
> Fit the model to the texts.

> It is not always necessary to implement this if there's not real fitting being done.

> > **Parameters texts** (*numpy.ndarray*) – Texts to be fitted.

fit_transform(*texts*, *titles=None*, *abstracts=None*, *keywords=None*)
> Fit and transform a list of texts.

> > **Parameters texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.

> > **Returns** *numpy.ndarray* – Feature matrix representing the texts.

param
> Get the (assigned) parameters of the model.
>
>> **Returns** *dict* – Dictionary with parameter: current value.

transform(*texts*)
> Transform a list of texts.
>
>> **Parameters texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.
>>
>> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

class asreview.models.feature_extraction.**EmbeddingIdf**(**args*, *embedding_fp=None*, *random_state=None*, ***kwargs*)

Class for Embedding-Idf model.

This model averages the weighted word vectors of all the words in the text, in order to get a single feature vector for each text. The weights are provided by the inverse document frequencies.

Note: This feature extraction algorithm requires tensorflow to be installed. Use pip install tensorflow or install all optional ASReview dependencies with pip install asreview[all]

> **Parameters embedding_fp** (*str*) – Path to embedding.

default_param
> Get the default parameters of the model.
>
>> **Returns** *dict* – Dictionary with parameter: default value

fit(*texts*)
> Fit the model to the texts.
>
> It is not always necessary to implement this if there's not real fitting being done.
>
>> **Parameters texts** (*numpy.ndarray*) – Texts to be fitted.

fit_transform(*texts*, *titles=None*, *abstracts=None*, *keywords=None*)
> Fit and transform a list of texts.
>
>> **Parameters texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.
>>
>> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

param
> Get the (assigned) parameters of the model.
>
>> **Returns** *dict* – Dictionary with parameter: current value.

transform(*texts*)
> Transform a list of texts.
>
>> **Parameters texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.
>>
>> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

class asreview.models.feature_extraction.**EmbeddingLSTM**(**args*, *loop_sequence=1*, *num_words=20000*, *max_sequence_length=1000*, *padding='post'*, *truncating='post'*, *n_jobs=1*, ***kwargs*)

Class to create embedding matrices for LSTM models.

Feature extraction method for *asreview.models.classifiers.LSTMBaseClassifier* and *asreview.models.classifiers.LSTMPoolClassifier* models.

Note: This feature extraction algorithm requires tensorflow to be installed. Use pip install tensorflow or install all optional ASReview dependencies with pip install asreview[all]

Parameters

- **loop_sequence** (*bool*) – Instead of zeros at the start/end of sequence loop it.

- **num_words** (*int*) – Maximum number of unique words to be processed.

- **max_sequence_length** (*int*) – Maximum length of the sequence. Shorter get struncated. Longer sequences get either padded with zeros or looped.

- **padding** (*str*) – Which side should be padded [pre/post].

- **truncating** – Which side should be truncated [pre/post].

- **n_jobs** – Number of processors used in reading the embedding matrix.

default_param
Get the default parameters of the model.

> **Returns** *dict* – Dictionary with parameter: default value

fit(*texts*)
Fit the model to the texts.

It is not always necessary to implement this if there's not real fitting being done.

> **Parameters** **texts** (*numpy.ndarray*) – Texts to be fitted.

fit_transform(*texts*, *titles=None*, *abstracts=None*, *keywords=None*)
Fit and transform a list of texts.

> **Parameters** **texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.

> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

param
Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

transform(*texts*)
Transform a list of texts.

> **Parameters** **texts** (*numpy.ndarray*) – A sequence of texts to be transformed. They are not yet tokenized.

> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

class `asreview.models.feature_extraction.`**SBERT**(*split_ta=0*, *use_keywords=0*)
Sentence BERT class for feature extraction.

Feature extraction method based on Sentence BERT. Implementation based on the sentence_transformers package. It is relatively slow.

Note: This feature extraction algorithm requires `sentence_transformers` to be installed. Use `pip install sentence_transformers` or install all optional ASReview dependencies with `pip install asreview[all]`

---

default_param
Get the default parameters of the model.

> **Returns** *dict* – Dictionary with parameter: default value

fit(*texts*)
Fit the model to the texts.

It is not always necessary to implement this if there's not real fitting being done.

> **Parameters** **texts** (`numpy.ndarray`) – Texts to be fitted.

fit_transform(*texts*, *titles=None*, *abstracts=None*, *keywords=None*)
Fit and transform a list of texts.

> **Parameters** **texts** (`numpy.ndarray`) – A sequence of texts to be transformed. They are not yet tokenized.

> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

param
Get the (assigned) parameters of the model.

> **Returns** *dict* – Dictionary with parameter: current value.

transform(*texts*)
Transform a list of texts.

> **Parameters** **texts** (`numpy.ndarray`) – A sequence of texts to be transformed. They are not yet tokenized.

> **Returns** *numpy.ndarray* – Feature matrix representing the texts.

`asreview.models.feature_extraction.`**list_feature_extraction**()
List available feature extraction methods.

> **Returns** *list* – Names of available feature extraction methods in alphabetical order.

`asreview.models.feature_extraction.`**get_feature_model**(*name*, **args*, *random_state=None*, ***kwargs*)
Get an instance of a feature extraction model from a string.

> **Parameters**
>
> > • **name** (`str`) – Name of the feature extraction model.
> >
> > • ***args** – Arguments for the feature extraction model.
> >
> > • ****kwargs** – Keyword arguments for thefeature extraction model.
>
> **Returns** *BaseFeatureExtraction* – Initialized instance of feature extraction algorithm.

`asreview.models.feature_extraction.`**get_feature_class**(*name*)
Get class of feature extraction from string.

Parameters name (*str*) – Name of the feature model, e.g. 'doc2vec', 'tfidf' or 'embedding-lstm'.

Returns *BaseFeatureExtraction* – Class corresponding to the name.

23.6 Data

class asreview.**ASReviewData**(*df=None*, *data_name='empty'*, *data_type='standard'*, *column_spec=None*)
Data object to the dataset with texts, labels, DOIs etc.

> **Parameters**
>
> - **df** (*pandas.DataFrame*) – Dataframe containing the data for the ASReview data object.
> - **data_name** (*str*) – Give a name to the data object.
> - **data_type** (*str*) – What kind of data the dataframe contains.
> - **column_spec** (*dict*) – Specification for which column corresponds to which standard specification. Key is the standard specification, key is which column it is actually in.

append(*as_data*)
Append another ASReviewData object.

It puts the training data at the end.

> **Parameters as_data** (*ASReviewData*) – Dataset to append.

classmethod from_file(*fp*, *read_fn=None*, *data_name=None*, *data_type=None*)
Create instance from csv/ris/excel file.

It works in two ways; either manual control where the conversion functions are supplied or automatic, where it searches in the entry points for the right conversion functions.

> **Parameters**
>
> - **fp** (*str, pathlib.Path*) – Read the data from this file.
> - **read_fn** (*callable*) – Function to read the file. It should return a standardized dataframe.
> - **data_name** (*str*) – Name of the data.
> - **data_type** (*str*) – What kind of data it is. Special names: 'included', 'excluded', 'prior'.

get(*name*)
Get column with name.

hash()
Compute a hash from the dataset.

> **Returns** *str* – SHA1 hash, computed from the titles/abstracts of the dataframe.

prior_data_idx
Get prior_included, prior_excluded from dataset.

prior_labels(*state*, *by_index=True*)
Get the labels that are marked as 'initial'.

state: BaseState Open state that contains the label information.

by_index: bool If True, return internal indexing. If False, return record_ids for indexing.

> **Returns** *numpy.ndarray* – Array of indices that have the 'initial' property.

record(*i*, *by_index=True*)
> Create a record from an index.
>
> > **Parameters**
> >
> > - **i** (`int, iterable`) – Index of the record, or list of indices.
> >
> > - **by_index** (`bool`) – If True, take the i-th value as used internally by the review. If False, take the record with record_id==i.
> >
> > **Returns** *PaperRecord* – The corresponding record if i was an integer, or a list of records if i was an iterable.

slice(*idx*, *by_index=True*)
> Create a slice from itself.
>
> Useful if some parts should be kept/thrown away.
>
> > **Parameters idx** (`list, numpy.ndarray`) – Record ids that should be kept.
> >
> > **Returns** *ASReviewData* – Slice of itself.

to_csv(*fp*, *sep=', '*, *labels=None*, *ranking=None*)
> Export to csv.
>
> > **Parameters**
> >
> > - **fp** (`str, NoneType`) – Filepath or None for buffer.
> >
> > - **sep** (`str`) – Seperator of the file.
> >
> > - **labels** (`list, numpy.ndarray`) – Current labels will be overwritten by these labels (including unlabelled). No effect if labels is None.
> >
> > - **ranking** (`list`) – Reorder the dataframe according to these (internal) indices. Default ordering if ranking is None.
> >
> > **Returns** *pandas.DataFrame* – Dataframe of all available record data.

to_dataframe(*labels=None*, *ranking=None*)
> Create new dataframe with updated label (order).
>
> > **Parameters**
> >
> > - **labels** (`list, numpy.ndarray`) – Current labels will be overwritten by these labels (including unlabelled). No effect if labels is None.
> >
> > - **ranking** (`list`) – Reorder the dataframe according to these record_ids. Default ordering if ranking is None.
> >
> > **Returns** *pandas.DataFrame* – Dataframe of all available record data.

to_excel(*fp*, *labels=None*, *ranking=None*)
> Export to Excel xlsx file.
>
> > **Parameters**
> >
> > - **fp** (`str, NoneType`) – Filepath or None for buffer.
> >
> > - **labels** (`list, numpy.ndarray`) – Current labels will be overwritten by these labels (including unlabelled). No effect if labels is None.
> >
> > - **ranking** (`list`) – Reorder the dataframe according to these (internal) indices. Default ordering if ranking is None.

Returns *pandas.DataFrame* – Dataframe of all available record data.

to_file(*fp*, *labels=None*, *ranking=None*)

Export data object to file.

RIS, CSV and Excel are supported file formats at the moment.

> **Parameters**
>
> - **fp** (*str*) – Filepath to export to.
>
> - **labels** (*list, numpy.ndarray*) – Labels to be inserted into the dataframe before export.
>
> - **ranking** (*list, numpy.ndarray*) – Optionally, dataframe rows can be reordered.

23.7 Utils

asreview.**load_embedding**(*fp*, *word_index=None*, *n_jobs=None*)

Load embedding matrix from file.

The embedding matrix needs to be stored in the FastText format.

> **Parameters**
>
> - **fp** (*str*) – File path of the trained embedding vectors.
>
> - **word_index** (*dict*) – Sample word embeddings.
>
> - **n_jobs** (*int*) – Number of processes to parse the embedding (+1 process for reading).
>
> - **verbose** (*int*) – The verbosity. Default 1.

Returns *dict* – The embedding weights stored in a dict with the word as key and the weights as values.

asreview.**sample_embedding**(*embedding*, *word_index*)

Sample embedding matrix

> **Parameters**
>
> - **embedding** (*dict*) – A dictionary with the words and embedding vectors.
>
> - **word_index** (*dict*) – A word_index like the output of Keras Tokenizer.word_index.
>
> - **verbose** (*int*) – The verbosity. Default 1.

Returns *(np.ndarray, list)* – The embedding weights strored in a two dimensional numpy array and a list with the corresponding words.

23.8 State

asreview.state.**open_state**(*fp*, **args*, *read_only=False*, ***kwargs*)

Open a state from a file.

> **Parameters**
>
> - **fp** (*str*) – File to open.
>
> - **read_only** (*bool*) – Whether to open the file in read_only mode.

> **Returns** *Basestate* – Depending on the extension the appropriate state is chosen: - [.h5, .hdf5, .he5]
> -> HDF5state. - None -> Dictstate (doesn't store anything permanently). - Anything else ->
> JSONstate.

class asreview.state.**BaseState**(*state_fp*, *read_only=False*)

add_classification(*idx*, *labels*, *methods*, *query_i*)
  Add training indices and their labels.

  > **Parameters**
  >
  > - **indices** (*list, numpy.ndarray*) – A list of indices used for training.
  >
  > - **labels** (*list*) – A list of labels corresponding with the training indices.
  >
  > - **i** (*int*) – The query number.

add_proba(*pool_idx*, *train_idx*, *proba*, *query_i*)
  Add inverse pool indices and their labels.

  > **Parameters**
  >
  > - **indices** (*list, numpy.ndarray*) – A list of indices used for unlabeled pool.
  >
  > - **pred** (*numpy.ndarray*) – Array of prediction probabilities for unlabeled pool.
  >
  > - **i** (*int*) – The query number.

close()
  Close the files opened by the state.

  Also sets the end time if not in read-only mode.

delete_last_query()
  Delete the last query from the state object.

get(*variable*, *query_i=None*, *default=None*, *idx=None*)
  Get data from the state object.

  This is universal accessor method of the State classes. It can be used to get a variable from one specific
  query. In theory, it should get the whole data set if query_i=None, but this is not currently implemented in
  any of the States.

  > **Parameters**
  >
  > - **variable** (*str*) – Name of the variable/data to get. Options are: label_idx, inclusions,
  >   label_methods, labels, final_labels, proba , train_idx, pool_idx.
  >
  > - **query_i** (*int*) – Query number, should be between 0 and self.n_queries().
  >
  > - **idx** (*int, numpy.ndarray, list*) – Indices to get in the returned array.

get_current_queries()
  Get the current queries made by the model.

  This is useful to get back exactly to the state it was in before shutting down a review.

  > **Returns** *dict* – The last known queries according to the state file.

get_feature_matrix(*data_hash*)
  Get feature matrix out of the state.

  > **Parameters** **data_hash** (*str*) – Hash of as_data object from which the matrix is derived.

  > **Returns** *np.ndarray, sklearn.sparse.csr_matrix* – Feature matrix as computed by the feature ex-
  > traction model.

initialize_structure()
  Create empty internal structure for state

is_empty()
  Check if state has no results.

  Returns *bool* – True if empty.

n_queries()
  Number of queries saved in the state.

  Returns *int* – Number of queries.

pred_proba
  Get last predicted probabilities.

restore(*fp*)
  Restore or create state from a state file.

  If the state file doesn't exist, creates and empty state that is ready for storage.

  Parameters fp (*str*) – Path to file to restore/create.

save()
  Save state to file.

  Parameters fp (*str*) – The file path to export the results to.

set_current_queries(*current_queries*)
  Set the current queries made by the model.

  Parameters current_queries (*dict*) – The last known queries, with {query_idx: query_method}.

set_final_labels(*y*)
  Add/set final labels to state.

  If final_labels does not exist yet, add it.

  Parameters y (*numpy.ndarray*) – One dimensional integer numpy array with final inclusion labels.

set_labels(*y*)
  Add/set labels to state

  If the labels do not exist, add it to the state.

  Parameters y (*numpy.ndarray*) – One dimensional integer numpy array with inclusion labels.

settings
  Get settings from state

startup_vals()
  Get variables for reviewer to continue review.

  Returns

  - *numpy.ndarray* – Current labels of dataset.

  - *numpy.ndarray* – Current training indices.

  - *dict* – Dictionary containing the sources of the labels.

  - *query_i* – Currenty query number (starting from 0).

to_dict()
> Convert state to dictionary.
>
> > **Returns** *dict* – Dictionary with all relevant variables.

class asreview.state.**HDF5State**(*state_fp*, *read_only=False*)
> Class for storing the review state with HDF5 storage.

> **add_classification**(*idx*, *labels*, *methods*, *query_i*)
> > Add training indices and their labels.
> >
> > > **Parameters**
> > >
> > > * **indices** (*list,* *numpy.ndarray*) – A list of indices used for training.
> > >
> > > * **labels** (*list*) – A list of labels corresponding with the training indices.
> > >
> > > * **i** (*int*) – The query number.

> **add_proba**(*pool_idx*, *train_idx*, *proba*, *query_i*)
> > Add inverse pool indices and their labels.
> >
> > > **Parameters**
> > >
> > > * **indices** (*list,* *numpy.ndarray*) – A list of indices used for unlabeled pool.
> > >
> > > * **pred** (*numpy.ndarray*) – Array of prediction probabilities for unlabeled pool.
> > >
> > > * **i** (*int*) – The query number.

> **close**()
> > Close the files opened by the state.
> >
> > Also sets the end time if not in read-only mode.

> **delete_last_query**()
> > Delete the last query from the state object.

> **get**(*variable*, *query_i=None*, *idx=None*)
> > Get data from the state object.
> >
> > This is universal accessor method of the State classes. It can be used to get a variable from one specific query. In theory, it should get the whole data set if query_i=None, but this is not currently implemented in any of the States.
> >
> > > **Parameters**
> > >
> > > * **variable** (*str*) – Name of the variable/data to get. Options are: label_idx, inclusions, label_methods, labels, final_labels, proba , train_idx, pool_idx.
> > >
> > > * **query_i** (*int*) – Query number, should be between 0 and self.n_queries().
> > >
> > > * **idx** (*int,* *numpy.ndarray,* *list*) – Indices to get in the returned array.

> **get_current_queries**()
> > Get the current queries made by the model.
> >
> > This is useful to get back exactly to the state it was in before shutting down a review.
> >
> > > **Returns** *dict* – The last known queries according to the state file.

> **get_feature_matrix**(*data_hash*)
> > Get feature matrix out of the state.
> >
> > > **Parameters** **data_hash** (*str*) – Hash of as_data object from which the matrix is derived.

> **Returns** *np.ndarray, sklearn.sparse.csr_matrix* – Feature matrix as computed by the feature extraction model.

initialize_structure()
> Create empty internal structure for state

is_empty()
> Check if state has no results.

> > **Returns** *bool* – True if empty.

n_queries()
> Number of queries saved in the state.

> > **Returns** *int* – Number of queries.

pred_proba
> Get last predicted probabilities.

restore(*fp*)
> Restore or create state from a state file.

> If the state file doesn't exist, creates and empty state that is ready for storage.

> > **Parameters** **fp** (`str`) – Path to file to restore/create.

save()
> Save state to file.

> > **Parameters** **fp** (`str`) – The file path to export the results to.

set_current_queries(*current_queries*)
> Set the current queries made by the model.

> > **Parameters** **current_queries** (`dict`) – The last known queries, with {query_idx: query_method}.

set_final_labels(*y*)
> Add/set final labels to state.

> If final_labels does not exist yet, add it.

> > **Parameters** **y** (`numpy.ndarray`) – One dimensional integer numpy array with final inclusion labels.

set_labels(*y*)
> Add/set labels to state

> If the labels do not exist, add it to the state.

> > **Parameters** **y** (`numpy.ndarray`) – One dimensional integer numpy array with inclusion labels.

settings
> Get settings from state

startup_vals()
> Get variables for reviewer to continue review.

> > **Returns**

> > - *numpy.ndarray* – Current labels of dataset.
> > - *numpy.ndarray* – Current training indices.
> > - *dict* – Dictionary containing the sources of the labels.

- *query_i* – Currently query number (starting from 0).

to_dict()
> Convert state to dictionary.
>
> > **Returns** *dict* – Dictionary with all relevant variables.

class asreview.state.**JSONState**(*state_fp*, *read_only=False*)
> Class for storing the state of a Systematic Review using JSON files.

> **add_classification**(*idx*, *labels*, *methods*, *query_i*)
> > Add training indices and their labels.
> >
> > > **Parameters**
> > >
> > > - **indices** (*list, numpy.ndarray*) – A list of indices used for training.
> > >
> > > - **labels** (*list*) – A list of labels corresponding with the training indices.
> > >
> > > - **i** (*int*) – The query number.

> **add_proba**(*pool_idx*, *train_idx*, *proba*, *query_i*)
> > Add inverse pool indices and their labels.
> >
> > > **Parameters**
> > >
> > > - **indices** (*list, numpy.ndarray*) – A list of indices used for unlabeled pool.
> > >
> > > - **pred** (*numpy.ndarray*) – Array of prediction probabilities for unlabeled pool.
> > >
> > > - **i** (*int*) – The query number.

> **close**()
> > Close the files opened by the state.
> >
> > Also sets the end time if not in read-only mode.

> **delete_last_query**()
> > Delete the last query from the state object.

> **get**(*variable*, *query_i=None*, *idx=None*)
> > Get data from the state object.
> >
> > This is universal accessor method of the State classes. It can be used to get a variable from one specific query. In theory, it should get the whole data set if query_i=None, but this is not currently implemented in any of the States.
> >
> > > **Parameters**
> > >
> > > - **variable** (*str*) – Name of the variable/data to get. Options are: label_idx, inclusions, label_methods, labels, final_labels, proba , train_idx, pool_idx.
> > >
> > > - **query_i** (*int*) – Query number, should be between 0 and self.n_queries().
> > >
> > > - **idx** (*int, numpy.ndarray, list*) – Indices to get in the returned array.

> **get_current_queries**()
> > Get the current queries made by the model.
> >
> > This is useful to get back exactly to the state it was in before shutting down a review.
> >
> > > **Returns** *dict* – The last known queries according to the state file.

> **get_feature_matrix**(*data_hash*)
> > Get feature matrix out of the state.
> >
> > > **Parameters** **data_hash** (*str*) – Hash of as_data object from which the matrix is derived.

> **Returns** *np.ndarray, sklearn.sparse.csr_matrix* – Feature matrix as computed by the feature extraction model.

initialize_structure()
> Create empty internal structure for state

is_empty()
> Check if state has no results.
>
> > **Returns** *bool* – True if empty.

n_queries()
> Number of queries saved in the state.
>
> > **Returns** *int* – Number of queries.

pred_proba
> Get last predicted probabilities.

restore(*fp*)
> Restore or create state from a state file.
>
> If the state file doesn't exist, creates and empty state that is ready for storage.
>
> > **Parameters** **fp** (*str*) – Path to file to restore/create.

save()
> Save state to file.
>
> > **Parameters** **fp** (*str*) – The file path to export the results to.

set_current_queries(*current_queries*)
> Set the current queries made by the model.
>
> > **Parameters** **current_queries** (*dict*) – The last known queries, with {query_idx: query_method}.

set_final_labels(*y*)
> Add/set final labels to state.
>
> If final_labels does not exist yet, add it.
>
> > **Parameters** **y** (*numpy.ndarray*) – One dimensional integer numpy array with final inclusion labels.

set_labels(*y*)
> Add/set labels to state
>
> If the labels do not exist, add it to the state.
>
> > **Parameters** **y** (*numpy.ndarray*) – One dimensional integer numpy array with inclusion labels.

settings
> Get settings from state

startup_vals()
> Get variables for reviewer to continue review.
>
> > **Returns**
> >
> > - *numpy.ndarray* – Current labels of dataset.
> > - *numpy.ndarray* – Current training indices.
> > - *dict* – Dictionary containing the sources of the labels.

> • *query_i* – Currenty query number (starting from 0).

to_dict()
> Convert state to dictionary.

> > **Returns** *dict* – Dictionary with all relevant variables.

class asreview.state.**DictState**(*state_fp*, **_*, ***__*)
> Class for storing the state of a review with no permanent storage.

> **add_classification**(*idx*, *labels*, *methods*, *query_i*)
> > Add training indices and their labels.

> > **Parameters**

> > > • **indices** (*list, numpy.ndarray*) – A list of indices used for training.

> > > • **labels** (*list*) – A list of labels corresponding with the training indices.

> > > • **i** (*int*) – The query number.

> **add_proba**(*pool_idx*, *train_idx*, *proba*, *query_i*)
> > Add inverse pool indices and their labels.

> > **Parameters**

> > > • **indices** (*list, numpy.ndarray*) – A list of indices used for unlabeled pool.

> > > • **pred** (*numpy.ndarray*) – Array of prediction probabilities for unlabeled pool.

> > > • **i** (*int*) – The query number.

> **close**()
> > Close the files opened by the state.

> > Also sets the end time if not in read-only mode.

> **delete_last_query**()
> > Delete the last query from the state object.

> **get**(*variable*, *query_i=None*, *idx=None*)
> > Get data from the state object.

> > This is universal accessor method of the State classes. It can be used to get a variable from one specific query. In theory, it should get the whole data set if query_i=None, but this is not currently implemented in any of the States.

> > **Parameters**

> > > • **variable** (*str*) – Name of the variable/data to get. Options are: label_idx, inclusions, label_methods, labels, final_labels, proba , train_idx, pool_idx.

> > > • **query_i** (*int*) – Query number, should be between 0 and self.n_queries().

> > > • **idx** (*int, numpy.ndarray, list*) – Indices to get in the returned array.

> **get_current_queries**()
> > Get the current queries made by the model.

> > This is useful to get back exactly to the state it was in before shutting down a review.

> > **Returns** *dict* – The last known queries according to the state file.

> **get_feature_matrix**(*data_hash*)
> > Get feature matrix out of the state.

> > **Parameters data_hash** (*str*) – Hash of as_data object from which the matrix is derived.

> **Returns** *np.ndarray, sklearn.sparse.csr_matrix* – Feature matrix as computed by the feature extraction model.

initialize_structure()
> Create empty internal structure for state

is_empty()
> Check if state has no results.
>
> > **Returns** *bool* – True if empty.

n_queries()
> Number of queries saved in the state.
>
> > **Returns** *int* – Number of queries.

pred_proba
> Get last predicted probabilities.

restore(**_, **__*)
> Restore or create state from a state file.
>
> If the state file doesn't exist, creates and empty state that is ready for storage.
>
> > **Parameters** **fp** (*str*) – Path to file to restore/create.

save()
> Save state to file.
>
> > **Parameters** **fp** (*str*) – The file path to export the results to.

set_current_queries(*current_queries*)
> Set the current queries made by the model.
>
> > **Parameters** **current_queries** (*dict*) – The last known queries, with {query_idx: query_method}.

set_final_labels(*y*)
> Add/set final labels to state.
>
> If final_labels does not exist yet, add it.
>
> > **Parameters** **y** (*numpy.ndarray*) – One dimensional integer numpy array with final inclusion labels.

set_labels(*y*)
> Add/set labels to state
>
> If the labels do not exist, add it to the state.
>
> > **Parameters** **y** (*numpy.ndarray*) – One dimensional integer numpy array with inclusion labels.

settings
> Get settings from state

startup_vals()
> Get variables for reviewer to continue review.
>
> > **Returns**
> >
> > - *numpy.ndarray* – Current labels of dataset.
> > - *numpy.ndarray* – Current training indices.
> > - *dict* – Dictionary containing the sources of the labels.

• *query_i* – Currently query number (starting from 0).

to_dict()
    Convert state to dictionary.

        Returns *dict* – Dictionary with all relevant variables.

23.9 Analysis

class `asreview.analysis.`**Analysis**(*states*, *key=None*)
    Analysis object to do statistical analysis on state files.

    avg_time_to_discovery(*result_format='number'*)
        Estimate the Time to Discovery (TD) for each paper.

        Get the best/last estimate on how long it takes to find a paper.

            Parameters result_format (`str`) – Desired output format: "number", "fraction" or "percentage".

            Returns *dict* – For each inclusion, key=paper_id, value=avg time.

    close()
        Close states.

    classmethod from_dir(*data_dir*, *prefix=''*, *key=None*)
        Create an Analysis object from a directory.

        Parameters

            • **data_dir** (`str`) – Directory to read the state files from.

            • **prefix** (`str`) – Only assume files starting with this prefix are state files. Ignore all other files.

            • **key** (`str`) – Name for the analysis object.

    classmethod from_file(*data_fp*, *key=None*)
        Create an Analysis object from a file.

        Parameters

            • **data_fp** (`str`) – Path to state file to analyse.

            • **key** (`str`) – Name for analysis object.

    classmethod from_path(*data_path*, *prefix=''*, *key=None*)
        Create an Analysis object from either a file or a directory.

    inclusions_found(*result_format='fraction'*, *final_labels=False*, ***kwargs*)
        Get the number of inclusions at each point in time.

        Caching is used to prevent multiple calls being expensive.

        Parameters

            • **result_format** (`str`) – The format % or # of the returned values.

            • **final_labels** (`bool`) – If true, use the final_labels instead of labels for analysis.

        Returns *tuple* – Three numpy arrays with x, y, error_bar.

limits (*prob_allow_miss=[0.1], result_format='percentage'*)
> For each query, compute the number of papers for a criterium.
>
> A criterium is the average number of papers missed. For example, with 0.1, the criterium is that after reading x papers, there is (about) a 10% chance that one paper is not included. Another example, with 2.0, there are on average 2 papers missed after reading x papers. The value for x is returned for each query and probability by the function.
>
> > **Parameters prob_allow_miss** (*list, float*) – Sets the criterium for how many papers can be missed.
> >
> > **Returns** *dict* – One entry, "x_range" with the number of papers read. List, "limits" of results for each probability and at # papers read.

rrf (*val=10*, *x_format='percentage'*, ***kwargs*)
> Get the RRF (Relevant References Found).
>
> > **Parameters**
> >
> > - **val** – At which recall, between 0 and 100.
> >
> > - **x_format** – Format for position of RRF value in graph.
> >
> > **Returns** *tuple* – Tuple consisting of RRF value, x_positions, y_positions of RRF bar.

wss (*val=100*, *x_format='percentage'*, ***kwargs*)
> Get the WSS (Work Saved Sampled) value.
>
> > **Parameters**
> >
> > - **val** – At which recall, between 0 and 100.
> >
> > - **x_format** – Format for position of WSS value in graph.
> >
> > **Returns** *tuple* – Tuple consisting of WSS value, x_positions, y_positions of WSS bar.

23.10 Extensions

class asreview.entry_points.**BaseEntryPoint**
> Base class for defining entry points.
>
> **classmethod execute** (*argv*)
> > Perform the functionality of the entry point.
> >
> > > **Parameters argv** (*list*) – Argument list, with the entry point and program removed. For example, if *asreview plot X* is executed, then argv == ['X'].
>
> **format** (*entry_name='?'*)
> > Create a short formatted description of the entry point.
> >
> > > **Parameters entry_name** (*str*) – Name of the entry point. For example 'plot' in *asreview plot X*

Active learning for Systematic Reviews

The rapidly evolving field of artificial intelligence (AI) has allowed the development of AI-aided pipelines that assist in finding relevant texts for such search tasks[1]. A well-established approach to increase the efficiency of title and abstract screening is determining prioritization[2, 3] with active learning[4], which is very effective for systematic reviewing[5-15]. It works as follows: Just like with a classical pipeline, you start with the set of all unlabeled records (e.g., meta-data containing titles and abstracts of scientific papers) retrieved from a search (pool). This is followed by constructing a training set, which, in the example of systematic reviewing, consists of labeled data provided by the annotator (researcher) (e.g., some relevant and irrelevant abstracts). Then, the active learning cycle starts. Active learning denotes the scenario in which the reviewer is labeling references that are presented by a machine learning model. The machine learning model learns from the reviewers' decision and uses this knowledge in selecting the next reference that will be presented to the reviewer. The annotated dataset starts out small and iteratively grows in size:

1. The first step is feature extraction. That is, an algorithm cannot make predictions from the records as they are; their textual content needs to be represented more abstractly. The algorithm can then determine the important features that are necessary to classify a record, thereby drastically decreasing the search space;

2. A specific classification algorithm (i.e, machine learning model) is then chosen;

3. The chosen classifier is trained on the labeled records and estimates relevance scores for all unlabeled records. The model chooses a record to show to the user for the next iteration of the cycle;

4. The annotator screens this record and provides a label, relevant or irrelevant. The newly labeled record is moved to the training data set and it's back to step 1.

The interaction with the human can be used to train a model with a minimum number of labeling tasks and the trained model is the output which is then used to classify new data, also called Human-in-the-Loop machine learning[16]. In the general sense the key idea behind active learning is that, if you allow the model to decide for itself which data it wants to learn from, its performance and accuracy may improve and it requires fewer training instances to do so.

Moreover, the dataset's informativeness is increased by having the reviewer annotate those references that are more informative to the model (**uncertainty-based sampling**).

The application of active learning to systematic reviewing is called Researcher-In-The-Loop (RITL)[17] with three unique components:

(I) The primary output of the process is a selection of the pool with only relevant papers;

(II) All data points in the relevant selection should have been seen by a human at the end of the process (**certainty-based sampling**);

(III) The process requires a strong, systematic way of working. As such, it entails several explicit and reproducible steps, as outlined in the PRISMA guidelines for systematic reviewing. This procedure ensures (more or less) that all likely relevant publications are found in a standardized way based on pre-defined eligibility criteria, extracting data from eligible studies, and synthesizing the results.

In the active learning cycle, the model incrementally improves its predictions on the remaining unlabeled records, but hopefully all relevant records are identified as early in the process as possible. The reviewer decides to stop at some point during the process to conserve resources or when all records have been labeled. In the latter case, no time was saved and therefore the main question is to decide when to stop: i.e. to determine the point at which the cost of labeling more papers by the reviewer is greater than the cost of the errors made by the current model[18]. Finding 100% of the relevant papers appears to be almost impossible, even for human annotators[19]. Therefore, we typically aim to find 95% of the inclusions. However, in the situation of an unlabeled dataset, you don't know how many relevant papers there are left to be found. So researchers might either stop too early and potentially miss many relevant papers, or stop too late, causing unnecessary further reading[20]. That is, one can decide to stop reviewing after a certain amount of non-relevant papers have been found in succession[21], but this is up to the user to decide.

1. Harrison, H., et al., Software tools to support title and abstract screening for systematic reviews in healthcare: an evaluation. BMC Medical Research Methodology, 2020. 20(1): p. 7.

2. Cohen, A.M., K. Ambert, and M. McDonagh, Cross-Topic Learning for Work Prioritization in Systematic Review Creation and Update. J Am Med Inform Assoc, 2009. 16(5): p. 690-704.

3. Shemilt, I., et al., Pinpointing Needles in Giant Haystacks: Use of Text Mining to Reduce Impractical Screening Workload in Extremely Large Scoping Reviews. Res. Synth. Methods, 2014. 5(1): p. 31-49.

4. Settles, B., Active Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2012. 6(1): p. 1-114.

5. Yu, Z. and T. Menzies, FAST2: An intelligent assistant for finding relevant papers. Expert Systems with Applications, 2019. 120: p. 57-71.

6. Yu, Z., N.A. Kraft, and T. Menzies, Finding Better Active Learners for Faster Literature Reviews. Empir. Softw. Eng., 2018. 23(6): p. 3161-3186.

7. Miwa, M., et al., Reducing Systematic Review Workload through Certainty-Based Screening. J Biomed Inform, 2014. 51: p. 242-253.

8. Cormack, G.V. and M.R. Grossman, Engineering quality and reliability in technology-assisted review. Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, 2016: p. 75-84.

9. Cormack, G.V. and M.R. Grossman, Autonomy and Reliability of Continuous Active Learning for Technology-Assisted Review. 2015.

10. Wallace, B.C., C.E. Brodley, and T.A. Trikalinos, Active learning for biomedical citation screening. Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, 2010: p. 173-182.

11. Wallace, B.C., et al., Deploying an interactive machine learning system in an evidence-based practice center: abstrackr. Proceedings of the ACM International Health Informatics Symposium (IHI), 2012: p. 819-824.

12. Wallace, B.C., et al., Semi-Automated Screening of Biomedical Citations for Systematic Reviews. BMC Bioinform, 2010. 11(1): p. 55-55.

13. Gates, A., et al., Performance and Usability of Machine Learning for Screening in Systematic Reviews: A Comparative Evaluation of Three Tools. Systematic Reviews, 2019. 8(1): p. 278-278.

14. Gates, A., C. Johnson, and L. Hartling, Technology-assisted title and abstract screening for systematic reviews: a retrospective evaluation of the Abstrackr machine learning tool. Systematic Reviews, 2018. 7(1): p. 45.

15. Singh, G., J. Thomas, and J. Shawe-Taylor, Improving Active Learning in Systematic Reviews. 2018.

16. Holzinger, A., Interactive Machine Learning for Health Informatics: When Do We Need the Human-in-the-Loop? Brain Inf., 2016. 3(2): p. 119-131.

17. Van de Schoot, R. and J. De Bruin Researcher-in-the-loop for systematic reviewing of text databases. 2020. DOI: 10.5281/zenodo.4013207.

18. Cohen, A.M., Performance of Support-Vector-Machine-Based Classification on 15 Systematic Review Topics Evaluated with the WSS@95 Measure. J Am Med Inform Assoc, 2011. 18(1): p. 104-104.

19. Wang, Z., et al., Error rates of human reviewers during abstract screening in systematic reviews. PloS one, 2020. 15(1): p. e0227742.

20. Yu, Z., N. Kraft, and T. Menzies, Finding better active learners for faster literature reviews. Empirical Software Engineering, 2018.

21. Ros, R., E. Bjarnason, and P. Runeson. A machine learning approach for semi-automated search and selection in literature studies. in Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. 2017.

22. Webster, A.J. and R. Kemp, Estimating omissions from searches. The American Statistician, 2013. 67(2): p. 82-89.

23. Stelfox, H.T., et al., Capture-mark-recapture to estimate the number of missed articles for systematic reviews in surgery. The American Journal of Surgery, 2013. 206(3): p. 439-440.

24. Kastner, M., et al., The capture–mark–recapture technique can be used as a stopping rule when searching in systematic reviews. Journal of clinical epidemiology, 2009. 62(2): p. 149-157.

Simulation results

To provide insight into how much screening effort ASReview could potentially save, seven ASReview models were simulated on six existing systematic review datasets (Ferdinands, 2020). In short, for all six datasets ASReview could have saved at least 60% of screening effort (e.g. WSS@95% was > 40%). For some datasets, ASReview was even able to detect 95% of relevant publications after screening only 5% of relevant publications.

25.1 Datasets

To assess the generalizability of the models across research contexts, the models were simulated on data from varying research contexts. Data were collected from the fields of medicine (Cohen et al. 2006; Appenzeller-Herzog et al. 2019), virology (Kwok et al. 2020), software engineering (Yu, Kraft, and Menzies 2018), behavioural public administration (Nagtegaal et al. 2019) and psychology (van de Schoot et al. 2017, 2018). Datasets are available in the ASReview systematic review datasets repository.

The data were preprocessed from their source into a dataset containing title and abstract of the publications obtained in the initial search. Candidate studies with missing abstracts and duplicate instances were removed from the data. All datasets consisted of thousands of candidate studies, of which only a fraction was deemed relevant to the systematic review. For the Virus and the Nudging dataset, the inclusion rate was about 5 per cent. For the remaining six datasets, inclusion rates were centred around 1-2 per cent.

Table 1 - Statistics on the systematic review datasets

Dataset	Candidate publications	Relevant publications	Inclusion rate (%)
Nudging	1847	100	5.41
PTSD	5031	38	0.76
Software	8896	104	1.17
Ace	2235	41	1.83
Virus	2304	114	4.95
Wilson	2333	23	0.99

25.2 Models

To assess the effect of different ASReview models, seven models were evaluated differing in terms of the classification technique (Naive Bayes, Linear Regression, Support Vector Machine, and Random Forest) and the feature extraction strategy they adopt (TF-IDF and Doc2vec). The Naive Bayes + TF-IDF model is the current default in ASReview. Note that the ASReview GUI currently only offers freedom of choice for a classification technique. However, the models presented here are not exhaustive as the ASReview backend implements various other configurations.

25.3 Evaluation

Model performance was assessed by two different measures, Work Saved over Sampling (WSS), and Relevant References Found (RRF). WSS indicates the reduction in publications needed to be screened, at a given level of recall (Cohen et al. 2006). Typically measured at a recall level of 0.95, WSS@95 yields an estimate of the amount of work that can be saved at the cost of failing to identify 5% of relevant publications. In the current study, WSS is computed at 0.95 recall. RRF statistics are computed at 10%, representing the proportion of relevant publications that are found after screening 10% of all publications.

Furthermore, model performance was visualized by plotting recall curves. Plotting recall as a function of the proportion of screened publications offers insight in model performance throughout the entire screening process (Cormack and Grossman 2014; Yu, Kraft, and Menzies 2018). The x-axis represents the proportion of screened publications, the y-axis represents the proportion of relevant publications found. The curves give information in two directions. On the one hand, they display the proportion of relevant publications found for any point during the screening process (on the x-axis), RRF. On the other hand, they present the proportion of publications that need to be screened to achieve a certain level of recall (on the y-axis), 1-WSS. The quicker the curve reaches towards the top of the y-axis, the better the performance.

For every simulation, the RRF@10 and WSS@95, are reported as means over 15 trials. To indicate the spread of performance within simulations, the means are accompanied by an estimated standard error of the mean (hat s). To compare overall performance across datasets, median performance is reported for every dataset, accompanied by the Median Absolute Deviation (MAD), indicating variability between models within a certain dataset. Recall curves are plot for every simulation, representing the average recall over 15 trials (pm) the standard error of the mean.

25.4 Results

The figures below show the recall curves of simulations for all model-dataset combinations. These curves plot recall as a function of the proportion of publications screened. The curves represent the average recall over 15 trials where the error margin represents the standard error of the mean in the direction of the y-axis. The x-axis is cut off at 40% since all for simulations, the models reached 95% recall after screening 40% of the publications. The dashed horizontal lines indicate the RRF@10 values, the dashed vertical lines the WSS@95 values. The dashed grey diagonal line corresponds to the expected recall curve when publications are screened in random order.

25.4.1 Recall curves

LRTB: Nudging dataset, PTSD dataset, Software dataset, Ace dataset, Virus dataset, Wilson dataset.
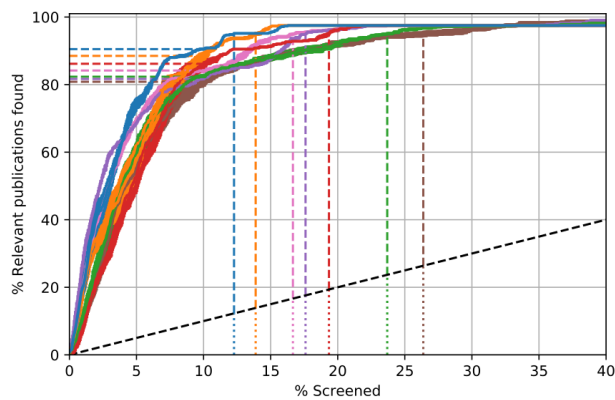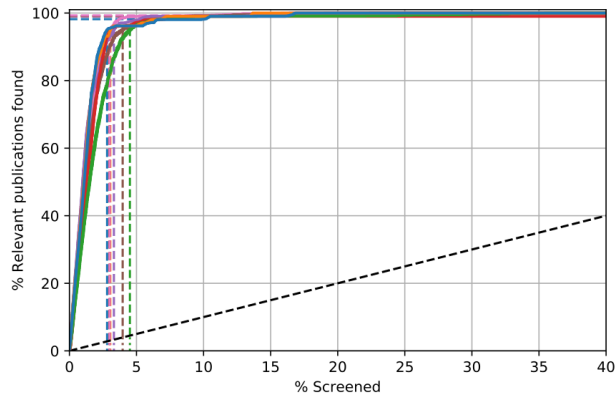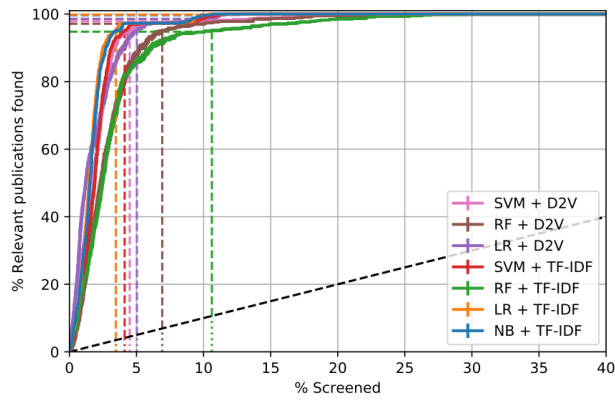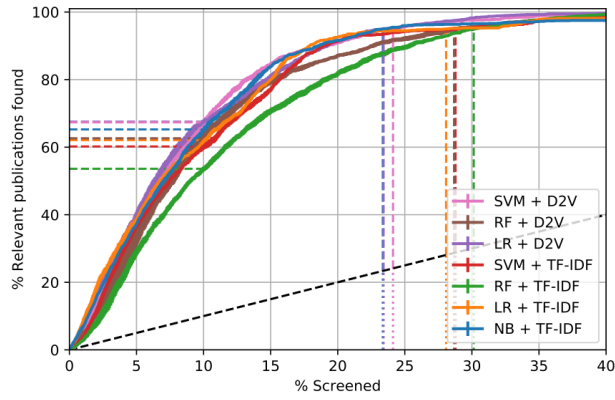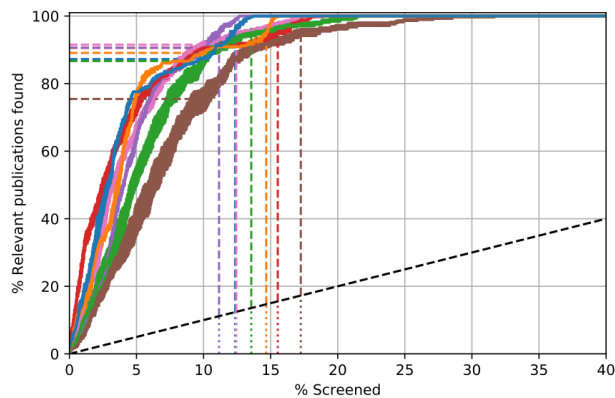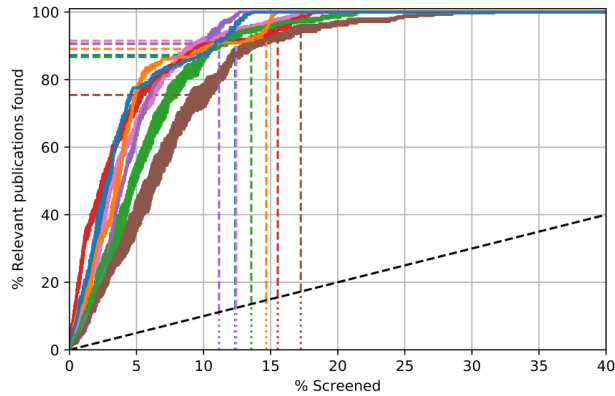
25.4.2 Between models comparison

For all datasets, the models were able to detect the relevant publications much faster compared to when screening publications at random order as the recall curves exceed the expected recall when screening at random order by far. While all models perform quite well, the NB + TF-IDF shows high performance on all measures across all datasets, whereas the RF + TF-IDF model never performed best on any of the measures across all datasets. Neither TF-IDF nor D2V feature extraction showed superior performance when simulated on certain datasets nor when combined with certain classification techniques.

25.4.3 Between datasets comparison

Firstly, models showed much higher performance for some datasets than for others. While performance on the PTSD and the Software dataset was quite high, the performance was much lower across models for the Nudging and Virus datasets. Secondly, the performance variability between models differed across datasets. within the PTSD, Software and Virus datasets, model performance is less spread out than within the Nudging, Ace and Wilson dataset. Thirdly, the curves for the Ace (Figure 2c) and Wilson (Figure 2e) datasets show a larger standard error of the mean compared to other the other datasets. For these datasets, model performance seemed to be more dependent on the initial training dataset compared to others.

25.4.4 WSS and RRF tables

Table 2 - WSS@95 values (mean, standard error) for all model-dataset combinations, and median (MAD) for all datasets

	Nudging	PTSD	Software	Ace	Virus	Wilson
SVM + TF-IDF	66.2 (2.90)	91.0 (0.41)	92.0 (0.10)	75.8 (1.95)	69.7 (0.81)	79.9 (2.09)
NB + TF-IDF	71.7 (1.37)	91.7 (0.27)	92.3 (0.08)	82.9 (0.99)	71.2 (0.62)	83.4 (0.89)
RF + TF-IDF	64.9 (2.50)	84.5 (3.38)	90.5 (0.34)	71.3 (4.03)	63.9 (3.54)	81.6 (3.35)
LR + TF-IDF	66.9 (4.01)	91.7 (0.18)	92.0 (0.10)	81.1 (1.31)	70.3 (0.65)	80.5 (0.65)
SVM + D2V	70.9 (1.68)	90.6 (0.73)	92.0 (0.21)	78.3 (1.92)	70.7 (1.76)	82.7 (1.44)
RF + D2V	66.3 (3.25)	88.2 (3.23)	91.0 (0.55)	68.6 (7.11)	67.2 (3.44)	77.9 (3.43)
LR + D2V	71.6 (1.66)	90.1 (0.63)	91.7 (0.13)	77.4 (1.03)	70.4 (1.34)	84.0 (0.77)
median (MAD)	66.9 (3.05)	90.6 (1.53)	92.0 (0.47)	77.4 (5.51)	70.3 (0.90)	81.6 (2.48)

Table 3 - RRF@10 values (mean, standard error) for all model-dataset combinations, and median (MAD) for all datasets

	Nudging	PTSD	Software	Ace	Virus	Wilson
SVM + TF-IDF	60.2 (3.12)	98.6 (1.40)	99.0 (0.00)	86.2 (5.25)	73.4 (1.62)	90.6 (1.17)
NB + TF-IDF	65.3 (2.61)	99.6 (0.95)	98.2 (0.34)	90.5 (1.40)	73.9 (1.70)	87.3 (2.55)
RF + TF-IDF	53.6 (2.71)	94.8 (1.60)	99.0 (0.00)	82.3 (2.75)	62.1 (3.19)	86.7 (5.82)
LR + TF-IDF	62.1 (2.59)	99.8 (0.70)	99.0 (0.00)	88.5 (5.16)	73.7 (1.48)	89.1 (2.30)
SVM + D2V	67.3 (3.00)	97.8 (1.12)	99.3 (0.44)	84.2 (2.78)	73.6 (2.54)	91.5 (4.16)
RF + D2V	62.6 (5.47)	97.1 (1.90)	99.2 (0.34)	80.8 (5.72)	67.3 (3.19)	75.5 (14.35)
LR + D2V	67.5 (2.59)	98.6 (1.40)	99.0 (0.00)	81.7 (1.81)	70.6 (2.21)	90.6 (5.00)
median (MAD)	62.6 (3.89)	98.6 (1.60)	99.0 (0.00)	84.2 (3.71)	73.4 (0.70)	89.1 (2.70)

25.5 Conclusion

Overall, the findings confirm the great potential of active learning models in reducing workload for systematic reviewers. All models were able to detect 95% of the relevant publications after screening less than 40% of the total number of publications, indicating that active learning models can save more than half of the workload in the screening process. The results shed new light on the performance of different classification techniques, indicating that the Naive Bayes classification technique is superior to the widely used Support Vector Machine. As model performance differs vastly across datasets, this study raises the question of what causes models to yield more workload savings for some systematic review datasets than for others. To facilitate the applicability of active learning models in systematic review practice, it is essential to identify how dataset characteristics relate to model performance.

25.6 References

[1] Cohen AM, Hersh WR, Peterson K, Yen P-Y. Reducing Workload in Systematic Review Preparation Using Automated Citation Classification. J Am Med Inform Assoc 2006;13:206–19. https://doi.org/10.1197/jamia.M1929.

[2] Appenzeller-Herzog C, Mathes T, Heeres MLS, Weiss KH, Houwen RHJ, Ewald H. Comparative effectiveness of common therapies for Wilson disease: A systematic review and meta-analysis of controlled studies. Liver Int 2019;39:2136–52. https://doi.org/10.1111/liv.14179.

[3] Kwok KTT, Nieuwenhuijse DF, Phan MVT, Koopmans MPG. Virus Metagenomics in Farm Animals: A Systematic Review. Viruses 2020;12:107. https://doi.org/10.3390/v12010107.

[4] Yu Z, Kraft NA, Menzies T. Finding better active learners for faster literature reviews. Empir Softw Eng 2018;23:3161–86. https://doi.org/10.1007/s10664-017-9587-0.

[5] Nagtegaal R, Tummers L, Noordegraaf M, Bekkers V. Nudging healthcare professionals towards evidence-based medicine: A systematic scoping review. J Behav Public Adm 2019;2. https://doi.org/10.30636/jbpa.22.71.

[6] van de Schoot R, Sijbrandij M, Winter SD, Depaoli S, Vermunt JK. The GRoLTS-Checklist: Guidelines for reporting on latent trajectory studies. Struct Equ Model Multidiscip J 2017;24:451–67. https://doi.org/10/gdpcw9.

[7] Cormack GV, Grossman MR. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery. In:. Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, Gold Coast, Queensland, Australia: Association for Computing Machinery; 2014, pp. 153–62. https://doi.org/10.1145/2600428.2609601.

Simulation

26.1 Why run a simulation?

Doing simulations can be a great way to assess how well ASReview performs for your particular purposes. The user can run simulations on previously labeled datasets to see how much time the user could have saved using ASReview.

26.2 Doing the simulation

The ASReview simulation mode works through the dataset exactly like an ASReview user would, using the inclusions and exclusions in the dataset to learn in the active learning cycle. In this way, the entire screening process is replicated.

You can use the simulation mode that is provided with the ASReview package. It can be accessed directly from the command line, for example:

```
asreview simulate MY_DATASET.csv --state_file myreview.h5
```

This performs a simulation of a default active learning model, where `MY_DATASET.csv` is the path to the dataset you wish to simulate on and where `myreview.h5` is the file where the results will be stored.

More detail on specific model and simulation settings can be found in the Simulation options section. For how to prepare your data, see *Prepare your Data*.

26.3 Analyzing your results

The extensions asreview-statistics and asreview-visualization are useful tools to analyze results. Install them directly from PyPi:

```
pip install asreview-statistics asreview-visualization
```

Detailed information can be found on their respective GitHub pages. The following commands should give you at least a basic exploratory idea of the performance of your review:

```
asreview stat YOUR_DATASET.csv
asreview stat myreview.h5
asreview stat DIR_WITH_MULTIPLE_SIMULATIONS

asreview plot myreview.h5
asreview plot DIR_WITH_MULTIPLE_SIMULATIONS
```

For an example of results of a simulation study, see *Simulation results*.

26.4 Simulation options

ASReview provides an extensive simulation interface via the command line. An overview of the options are found on the *ASReview command line interface for simulation* page. This section highlights some of the more often used options here. When no additional arguments are specified in the `asreview simulate` command, default settings are used.

To make your simulations reproducible you can use the `--seed` and `--init_seed` options. 'init_seed' controls the starting set of papers to train the model on, while the 'seed' controls the seed of the random number generation that is used after initialization.

By default, the model initializes with one relevant and one irrelevant record. You can set the number of priors by *–n_prior_included* and *–n_prior_excluded*. However, if you want to initialize your model with a specific set of starting papers, you can use `--prior_idx` to select the indices of the papers you want to start the simulation with.

The `--n_instances` argument controls the number of records that have to be labeled before the model is retrained again and is set at 1 by default. If you want to reduce the number of training iterations, for example to limit the size of your state file and the time to simulate, you can increase `--n_instances`.

You can select a classifier with the `-m` flag, which is set to be Naive Bayes by default. Names for implemented classifiers are listed on the :ref :*classifiers-table* table.

Implemented query strategies are listed on the *Query Strategies* table and can be set with the `-q` option.

For feature extraction, supply the `-e` flag. Default is TF-IDF, more details on the table for *Feature Extraction*.

The last element that can be changed is the *Balance Strategies*, and is changed with the `-b` flag. Default is double balance.

Programming Interface (API)

For more control over the workings of the ASReview software, an API is provided. For example, it is possible to define a new model or a sampling strategy and use it with ASReview.

27.1 Simulation Mode

This example shows how to use the API in simulation mode:

```python
import asreview
from asreview.models import LSTMBaseModel
from asreview.query_strategies import MaxQueryModel
from asreview.balance_strategies import SimpleBalanceModel
from asreview.feature_extraction import EmbeddingLSTM

# Load data
as_data = asreview.ASReviewData.from_file(DATA_FILE)

train_model = LSTMBaseModel()
query_model = MaxQueryModel()
balance_model = SimpleBalanceModel()
feature_model = EmbeddingLSTM()

# Load the embedding matrix, only necessary for LSTM models
train_model.embedding_matrix = feature_model.get_embedding_matrix(
        as_data.texts, EMBEDDING_FILE)

# Start the review process
reviewer = asreview.ReviewSimulate(
    as_data,
    model=train_model,
    query_model=query_model,
    balance_model=balance_model,
    feature_model=feature_model,
```

```
    n_instances=10,
)
reviewer.review()
```

CHAPTER 28

Indices and tables

- genindex
- modindex
- search

Index

Symbols

U

W