# soweego

*Release 1.0*

**Marco Fossati, Massimo Frasson, Edoardo Lenzi, Andrea Tupini**

**Feb 23, 2020**

# CONTENTS

*soweego* is a pipeline that connects Wikidata to large-scale third-party catalogs.

*soweego* is the only system that makes *statisticians, epidemiologists, historians,* and *computer scientists* agree. Why? Because it performs *record linkage, data matching,* and *entity resolution* at the same time. Too easy, they all seem to be synonyms!

Oh, *soweego* also embeds Machine Learning and advocates for Linked Data.

CHAPTER

# ONE

# OFFICIAL PROJECT PAGE

*soweego* is made possible thanks to the Wikimedia Foundation:
https://meta.wikimedia.org/wiki/Grants:Project/Hjfocs/soweego

CHAPTER

TWO

# HIGHLIGHTS

- Run the whole *pipeline*, or
- use the *command line*;
- `import` large catalogs into a SQL database;
- `gather` live Wikidata datasets;
- `connect` them to target catalogs via *rule-based* and *supervised* linkers;
- `upload` links to Wikidata and Mix'n'match;
- `synchronize` Wikidata to imported catalogs;
- `enrich` Wikidata items with relevant statements.

CHAPTER

THREE

# GET READY

Install Docker and Compose, then enter *soweego*:

```
$ git clone https://github.com/Wikidata/soweego.git
$ cd soweego
$ ./docker/run.sh
Building soweego
...

root@70c9b4894a30:/app/soweego#
```

Now it's too late to get out!

CHAPTER

# FOUR

# RUN THE PIPELINE

Piece of cake:

```
:/app/soweego# python -m soweego run CATALOG
```

Pick `CATALOG` from `discogs`, `imdb`, or `musicbrainz`.

These steps are executed by default:

1. import the target catalog into a local database;
2. link Wikidata to the target with a supervised linker;
3. synchronize Wikidata to the target.

Results are in `/app/shared/results`.

CHAPTER

FIVE

# USE THE COMMAND LINE

You can launch every single *soweego* action with CLI commands:

```
:/app/soweego# python -m soweego
Usage: soweego [OPTIONS] COMMAND [ARGS]...

  Link Wikidata to large catalogs.

Options:
  -l, --log-level <TEXT CHOICE>...
                            Module name followed by one of [DEBUG, INFO,
                            WARNING, ERROR, CRITICAL]. Multiple pairs
                            allowed.
  --help                    Show this message and exit.

Commands:
  importer  Import target catalog dumps into a SQL database.
  ingester  Take soweego output into Wikidata items.
  linker    Link Wikidata items to target catalog identifiers.
  run       Launch the whole pipeline.
  sync      Sync Wikidata to target catalogs.
```

Just two things to remember:

1. you can always get `--help`;
2. each command may have sub-commands.

Find all details in the *CLI Documentation*.

CHAPTER

SIX

# HOW-TOS

## 6.1 Run the pipeline

*soweego* is a pipeline of **Python modules** by design. Each module can be used alone or combined with others at will.

In this page, you will grasp the typical workflow:

1. *import* the dumps of a given target catalog into a SQL database
2. *link* the imported catalog to Wikidata
3. *sync* Wikidata to the imported catalog

### 6.1.1 Get set

1. Install Docker
2. install MariaDB
3. create a credentials JSON file like this:

```
{
    "DB_ENGINE": "mysql+pymysql",
    "HOST": "${DB_IP_ADDRESS}",
    "USER": "${DB_USER}",
    "PASSWORD": "${DB_PASSWORD}",
    "TEST_DB": "soweego",
    "PROD_DB": "${DB_NAME}",
    "WIKIDATA_API_USER": "${WIKI_USER_NAME}",
    "WIKIDATA_API_PASSWORD": "${WIKI_PASSWORD}"
}
```

`WIKIDATA_API_USER` and `WIKIDATA_API_PASSWORD` are optional: set them to run authenticated requests against the Wikidata Web API. If you have a Wikidata bot account, processing will speed up.

*soweego*'s favourite food is disk space, so make sure you have enough: **20 GB** should sate its appetite.

### 6.1.2 Go

```
$ git clone https://github.com/Wikidata/soweego.git
$ cd soweego
$ ./docker/pipeline.sh -c ${CREDENTIALS_FILE} -s ${OUTPUT_FOLDER} ${CATALOG}
```

`${OUTPUT_FOLDER}` is a path to a folder on your local filesystem: this is where all *soweego* output goes. Pick `${CATALOG}` from `discogs`, `imdb`, or `musicbrainz`.

#### `pipeline.sh`

This script does not only run *soweego*, but also takes care of some side tasks:

- backs up the output folder in a tar ball
- keeps at most 3 backups
- empties the output folder
- pulls the latest *soweego* master branch. **N.B.:** this will **erase any pending edits** in the local git repository

| Flag | Default | Description |
|---|---|---|
| `--importer` / `--no-importer` | enabled | enable / disable the importer |
| `--linker` / `--no-linker` | enabled | enable / disable the linker |
| `--validator` / `--no-validator` | enabled | enable / disable the validator |
| `--upload` / `--no-upload` | disabled | enable / disable the upload of results to Wikidata |

### 6.1.3 Under the hood

The actual pipeline is implemented in `soweego/pipeline.py`, so you can also launch it with

```
python -m soweego run
```

See *The command line* and *Pipeline* for more details.

### 6.1.4 Cron jobs

*soweego* periodically runs pipelines for each supported catalog via cron jobs. You can find `crontab`-ready scripts in the `scripts/cron` folder. Feel free to reuse them! Just remember to set the appropriate paths.

## 6.2 Import a new catalog

Five steps:

1. set up the *Development environment*
2. declare the SQLAlchemy Object Relational Mapper (*ORM*)
3. implement the catalog *Extractor*
4. *Set up the CLI to import your catalog*
5. *Run the importer*

---

**Note:** you will encounter some variables while reading this page

- set `${PROJECT_ROOT}` to the root directory where *soweego* lives
- set `${CATALOG}` to the name of the catalog you want to import, like `IMDb`
- set `${ENTITY}` to what the catalog is about, like `Musician` or `Book`

- the other ones should be self-explanatory

### 6.2.1 ORM

1. create a Python file in:

   ```
   ${PROJECT_ROOT}/soweego/importer/models/${CATALOG}_entity.py
   ```

2. paste the code snippet below
3. set the `${...}` variables accordingly
4. **optional:** define catalog-specific attributes
   - see `TODO` in the code snippet
   - just remember that attribute names **must be different** from *`BaseEntity`* ones, otherwise you would override them
   - don't forget their documentation!

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""`${CATALOG} <${CATALOG_HOME_URL}>`_
`SQLAlchemy <https://www.sqlalchemy.org/>`_ ORM entities."""

__author__ = '${YOUR_NAME_HERE}'
__email__ = '${YOUR_EMAIL_HERE}'
__version__ = '1.0'
__license__ = 'GPL-3.0'
__copyright__ = 'Copyleft ${YEAR}, ${YOUR_NAME_HERE}'

from sqlalchemy import Column, String

from soweego.importer.models.base_entity import BaseEntity

${ENTITY}_TABLE = '${CATALOG}_${ENTITY}'

class ${CATALOG}${ENTITY}Entity(BaseEntity):
    """A ${CATALOG} ${ENTITY}.
    It comes from the ${CATALOG_DUMP_FILE} dataset.
    See the `download page <${CATALOG_DOWNLOAD_URL}>`_.

    **Attributes:**

    - **birth_place** (string(255)) - a birth place

    """

    __tablename__ = ${ENTITY}_TABLE
    __mapper_args__ = {
        'polymorphic_identity': __tablename__,
        'concrete': True
    }

    # TODO Optional: define catalog-specific attributes here
```

(continues on next page)

(continued from previous page)

```
    # For instance:
    birth_place = Column(String(255))
```

### 6.2.2 Extractor

1. create a Python file in:

   ```
   ${PROJECT_ROOT}/soweego/importer/${CATALOG}_dump_extractor.py
   ```

2. paste the code snippet below
3. set the `${...}` variables accordingly
4. implement *BaseDumpExtractor* methods:
   - *extract_and_populate()* should extract instances of your `${CATALOG}${ENTITY}Entity` from relevant catalog dumps and store them in a database. The `extract` step is up to you. For the `populate` step, see *Populate the SQL database*
   - *get_dump_download_urls()* should compute the latest list of URLs to download catalog dumps. Tipically, there will be only one, but you never know
5. still tortured by doubts? Check out *DiscogsDumpExtractor*, *IMDbDumpExtractor*, or `MusicBrainzDumpExtractor`. You are now doubtless

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""`${CATALOG} <${CATALOG_HOME_URL}>`_
`SQLAlchemy <https://www.sqlalchemy.org/>`_ ORM entities."""

__author__ = '${YOUR_NAME_HERE}'
__email__ = '${YOUR_EMAIL_HERE}'
__version__ = '1.0'
__license__ = 'GPL-3.0'
__copyright__ = 'Copyleft ${YEAR}, ${YOUR_NAME_HERE}'

from soweego.importer.base_dump_extractor import BaseDumpExtractor


class ${CATALOG}DumpExtractor(BaseDumpExtractor):
    """Download ${CATALOG} dumps, extract data, and
    populate a database instance.
    """

    def extract_and_populate(
            self, dump_file_paths: List[str], resolve: bool
    ) -> None:
        # TODO implement!

    def get_dump_download_urls(self) -> Optional[List[str]]:
        # TODO implement!
```

#### Populate the SQL database

```
from sqlalchemy.exc import SQLAlchemyError

from soweego.commons.db_manager import DBManager
from soweego.importer.base_dump_extractor import BaseDumpExtractor


class ${CATALOG}DumpExtractor(BaseDumpExtractor):

    def extract_and_populate(
            self, dump_file_paths: List[str], resolve: bool
    ) -> None:

        # The `extract` step should build a list of entities
        # For instance:
        entities = _extract_from(dump_file_paths)

        # 1. Get a `DBManager` instance
        db_manager = DBManager()

        # 2. Drop & recreate database tables
        db_manager.drop(${CATALOG}${ENTITY})
        db_manager.create(${CATALOG}${ENTITY})

        # 3. Create a session, AKA a database transaction
        session = db_manager.new_session()

        try:
            # 4. Add a list of entities to the session
            session.bulk_save_objects(entities)

            # 5. Commit the session
            session.commit()

        except SQLAlchemyError as error:
            # 6. Handle transaction errors
            # For instance: (are you serious? Don't do this)
            print(f'There was an error: {error}')

            session.rollback()

        finally:
            session.close()
```

### 6.2.3 Set up the CLI to import your catalog

1. add your catalog keys in

   ```
   ${PROJECT_ROOT}/soweego/commons/keys.py
   ```

```
# Supported catalogs
MUSICBRAINZ = 'musicbrainz'
...
${CATALOG} = '${CATALOG}'
```

(continues on next page)

(continued from previous page)

```
# Supported entities
# People
ACTOR = 'actor'
...
${ENTITY} = '${ENTITY}'
```

2. include your extractor in the `DUMP_EXTRACTOR` dictionary of

   `${PROJECT_ROOT}/soweego/importer/importer.py`

```
DUMP_EXTRACTOR = {
    keys.MUSICBRAINZ: MusicBrainzDumpExtractor,
    ...
    keys.${CATALOG}: ${CATALOG}DumpExtractor
}
```

3. add the Wikidata class QID corresponding to your entity in

   `${PROJECT_ROOT}/soweego/wikidata/vocabulary.py`

```
# Class QID of supported entities
# People
ACTOR_QID = 'Q33999'
...
${ENTITY}_QID = '${QID}'
```

4. include your catalog mapping in the `TARGET_CATALOGS` dictionary of

   `${PROJECT_ROOT}/soweego/commons/constants.py`

```
keys.MUSICBRAINZ: {
        keys.MUSICIAN: {
            keys.CLASS_QID: vocabulary.MUSICIAN_QID,
            keys.MAIN_ENTITY: MusicBrainzArtistEntity,
            keys.LINK_ENTITY: MusicBrainzArtistLinkEntity,
            keys.NLP_ENTITY: None,
            keys.RELATIONSHIP_ENTITY: MusicBrainzReleaseGroupArtistRelationship,
            keys.WORK_TYPE: keys.MUSICAL_WORK,
        },
        ...
},
keys.${CATALOG}: {
        keys.${ENTITY}: {
            keys.CLASS_QID: vocabulary.${ENTITY}_QID,
            keys.MAIN_ENTITY: ${CATALOG}${ENTITY}Entity,
            keys.LINK_ENTITY: None,
            keys.NLP_ENTITY: None,
            keys.RELATIONSHIP_ENTITY: None,
            keys.WORK_TYPE: None,
        },
},
```

### 6.2.4 Run the importer

```
:/app/soweego# python -m soweego importer import ${CATALOG}
```

## 6.3 Development and production environments

**Note:** as a *soweego* user or contributor, you will typically use the *Development environment*. The *Production environment* is tailored for Cloud VPS project members.

### 6.3.1 First of all

Install Docker, then clone soweego:

```
$ git clone https://github.com/Wikidata/soweego.git
$ cd soweego
```

### 6.3.2 Development environment

Use it to run or play around *soweego* on your local machine. And to contribute new features, of course!

This environment ships with a MariaDB database instance and a BASH shell. It is ready to run *The command line*. Feel free to hack *soweego* while the environment is running: your code is synced.

#### Get set

Just install Docker Compose.

#### Go

```
$ ./docker/run.sh
Building soweego
...

root@70c9b4894a30:/app/soweego#
```

You are now in a BASH shell with a fully working *soweego* instance. Check if everything went fine with a shot of

```
python -m soweego
```

#### `run.sh` options

| Option | Expected value | Default value | Description |
|---|---|---|---|
| `-s` | a directory path | `/tmp/soweego_shared` | Directory shared between the *soweego* Docker container and your local filesystem |

#### Access the local database instance

As easy as:

```
$ docker exec -it soweego_db_1 /bin/bash
root@0f51e7c512df:/# mysql -uroot -hlocalhost -pdba soweego
MariaDB [soweego]>
```

### 6.3.3 Production environment

*soweego* lives in a Wikimedia Cloud VPS project, and this is the environment deployed there. Please contact the project administrators if you want to help with the VPS machine.

You can also use it to run *soweego* on a machine that already has a working database (typically a server).

This environment ships with a BASH shell ready to run *The command line*. Feel free to hack *soweego* while the environment is running: your code is synced.

#### Get set

Just create a credentials JSON file like this:

```
{
    "DB_ENGINE": "mysql+pymysql",
    "HOST": "${DB_IP_ADDRESS}",
    "USER": "${DB_USER}",
    "PASSWORD": "${DB_PASSWORD}",
    "TEST_DB": "soweego",
    "PROD_DB": "${DB_NAME}"
}
```

Don't forget to set the `${...}` variables!

#### Go

```
$ ./docker/prod.sh -c ${YOUR_CREDENTIALS_FILE}
Sending build context to Docker daemon
...

root@62c602c23fa9:/app/soweego#
```

You are now in a BASH shell with a fully working *soweego* instance. Check if everything went fine with a shot of

```
python -m soweego
```

### `prod.sh` options

| Option | Expected value | Default value | Description |
|---|---|---|---|
| `-s` | a directory path | `/tmp/soweego_shared` | Directory shared between the *soweego* Docker container and your local filesystem |
| `-c` | a file path | `${PROJECT_ROOT}/soweego/importer/resources/credentials.json` | Credentials file |

CHAPTER

SEVEN

# CLI DOCUMENTATION

## 7.1 The command line

**Note:** start your exploration journey of the command line interface (*CLI*) with

```
$ python -m soweego
```

As a reminder, make sure you are inside *soweego*:

```
$ cd soweego && ./docker/run.sh
Building soweego

...

root@70c9b4894a30:/app/soweego#
```

### 7.1.1 Importer

```
python -m soweego importer
Usage: soweego importer [OPTIONS] COMMAND [ARGS]...

  Import target catalog dumps into a SQL database.

Options:
  --help  Show this message and exit.

Commands:
  check_urls  Check for rotten URLs of an imported catalog.
  import      Download, extract, and import a supported catalog.
```

#### check_urls

Check for rotten URLs of an imported catalog.

```
check_urls [OPTIONS] [discogs|imdb|musicbrainz]
```

#### Arguments

**CATALOG**
Required argument

#### import

Download, extract, and import a supported catalog.

```
import [OPTIONS] [discogs|imdb|musicbrainz]
```

#### Options

**--url-check**
Check for rotten URLs while importing. Default: no. WARNING: this will dramatically increase the import time.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

#### Arguments

**CATALOG**
Required argument

### 7.1.2 Ingester

```
python -m soweego ingester
Usage: soweego ingester [OPTIONS] COMMAND [ARGS]...

  Take soweego output into Wikidata items.

Options:
  --help  Show this message and exit.

Commands:
  delete       Delete invalid identifiers.
  deprecate    Deprecate invalid identifiers.
  identifiers  Add identifiers.
  mnm          Upload matches to the Mix'n'match tool.
  people       Add statements to Wikidata people.
  works        Add statements to Wikidata works.
```

#### delete

Delete invalid identifiers.

INVALID_IDENTIFIERS must be a JSON file. Format: { catalog_identifier: [ list of QIDs ] }

```
delete [OPTIONS] [musicbrainz|imdb|discogs|twitter] [writer|producer|actor|dir
       ector|band|audiovisual_work|musical_work|musician] INVALID_IDENTIFIERS
```

### Options

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

**INVALID_IDENTIFIERS**
Required argument

## deprecate

Deprecate invalid identifiers.

INVALID_IDENTIFIERS must be a JSON file. Format: { catalog_identifier: [ list of QIDs ] }

```
deprecate [OPTIONS] [musicbrainz|imdb|discogs|twitter] [writer|producer|actor|
          director|band|audiovisual_work|musical_work|musician]
          INVALID_IDENTIFIERS
```

### Options

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

**INVALID_IDENTIFIERS**
Required argument

## identifiers

Add identifiers.

IDENTIFIERS must be a JSON file. Format: { QID: catalog_identifier }

If the identifier already exists, just add a reference.

Example:

$ echo '{ "Q446627": "266995" }' > rhell.json

$ python -m soweego ingester identifiers discogs musician rhell.json

Result:

claim (Richard Hell, Discogs artist ID, 266995)

**reference (based on heuristic, artificial intelligence),** (retrieved, today)

```
identifiers [OPTIONS] [musicbrainz|imdb|discogs|twitter] [writer|producer|acto
            r|director|band|audiovisual_work|musical_work|musician]
            IDENTIFIERS
```

### Options

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

**IDENTIFIERS**
Required argument

### mnm

Upload matches to the Mix'n'match tool.

CONFIDENCE_RANGE must be a pair of floats that indicate the minimum and maximum confidence scores.

MATCHES must be a CSV file path. Format: QID, catalog_identifier, confidence_score

The CSV file can be compressed.

Example:

echo Q446627,266995,0.666 > rhell.csv

python -m soweego ingest mnm discogs musician 0.3 0.7 rhell.csv

Result: see 'Latest catalogs' at https://tools.wmflabs.org/mix-n-match/

```
mnm [OPTIONS] [musicbrainz|imdb|discogs|twitter] [writer|producer|actor|direct
    or|band|audiovisual_work|musical_work|musician] CONFIDENCE_RANGE...
    MATCHES
```

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

**CONFIDENCE_RANGE**
Required argument(s)

**MATCHES**
Required argument

### people

Add statements to Wikidata people.

STATEMENTS must be a CSV file. Format: person_QID, PID, value

If the claim already exists, just add a reference.

Example:

$ echo Q312387,P463,Q483407 > joey.csv

$ python -m soweego ingester people joey.csv

Result:

claim (Joey Ramone, member of, Ramones)

**reference (based on heuristic, artificial intelligence),** (retrieved, today)

```
people [OPTIONS] STATEMENTS
```

### Options

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

### Arguments

**STATEMENTS**
Required argument

### works

Add statements to Wikidata works.

STATEMENTS must be a CSV file. Format: work_QID, PID, person_QID, person_target_ID

If the claim already exists, just add a reference.

Example:

$ echo Q4354548,P175,Q5969,139984 > cmon.csv

$ python -m soweego ingester works discogs cmon.csv

Result:

claim (C'mon Everybody, performer, Eddie Cochran)

**reference (based on heuristic, artificial intelligence),** (Discogs artist ID, 139984), (retrieved, today)

```
works [OPTIONS] [musicbrainz|imdb|discogs|twitter] STATEMENTS
```

#### Options

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

#### Arguments

**CATALOG**
Required argument

**STATEMENTS**
Required argument

### 7.1.3 Linker

```
python -m soweego linker
Usage: soweego linker [OPTIONS] COMMAND [ARGS]...

  Link Wikidata items to target catalog identifiers.

Options:
  --help  Show this message and exit.

Commands:
  baseline  Run a rule-based linker.
  evaluate  Evaluate the performance of a supervised linker.
  extract   Extract Wikidata links from a target catalog dump.
  link      Run a supervised linker.
  train     Train a supervised linker.
```

#### baseline

Run a rule-based linker.

Available rules:

'perfect' = perfect match on names

'links' = similar match on link tokens

'names' = similar match on name tokens

Run all of them by default.

```
baseline [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|
         band|audiovisual_work|musical_work|musician]
```

#### Options

**-r, --rule** <rule>
Activate a specific rule or all of them. Default: all.

**Options** perfect|links|names|all

**-u, --upload**
Upload links to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

**--dates, --no-dates**
Check if dates match, when applicable. Default: yes.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

### evaluate

Evaluate the performance of a supervised linker.

By default, run 5-fold cross-validation and return averaged performance scores.

```
evaluate [OPTIONS] [naive_bayes|logistic_regression|support_vector_machines|li
         near_support_vector_machines|random_forest|single_layer_perceptron|mu
         lti_layer_perceptron|voting_classifier|gated_classifier|stacked_class
         ifier|nb|lr|svm|lsvm|rf|slp|mlp|vc|gc|sc] [discogs|imdb|musicbrainz] [
         writer|producer|actor|director|band|audiovisual_work|musical_work|mus
         ician]
```

### Options

**-k, --k-folds** <k_folds>
Number of folds, default: 5.

**-s, --single**
Compute a single evaluation over all k folds, instead of k evaluations.

**-n, --nested**
Compute a nested cross-validation with hyperparameters tuning via grid search. WARNING: this will take a lot of time.

**-m, --metric** <metric>
Performance metric for nested cross-validation. Use with '–nested'. Default: f1.

**Options** precision|recall|f1

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CLASSIFIER**
Required argument

**CATALOG**
Required argument

**ENTITY**
Required argument

### extract

Extract Wikidata links from a target catalog dump.

```
extract [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|b
        and|audiovisual_work|musical_work|musician]
```

### Options

**-u, --upload**
Upload links to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

### link

Run a supervised linker.

Build the classification set relevant to the given catalog and entity, then generate links between Wikidata items and catalog identifiers.

Output a gzipped CSV file, format: QID,catalog_ID,confidence_score

You can pass the '-u' flag to upload the output to Wikidata.

A trained model must exist for the given classifier, catalog, entity. To do so, use:

$ python -m soweego linker train

```
link [OPTIONS] [naive_bayes|logistic_regression|support_vector_machines|linear
     _support_vector_machines|random_forest|single_layer_perceptron|multi_laye
     r_perceptron|voting_classifier|gated_classifier|stacked_classifier|nb|lr|
```

(continues on next page)

(continued from previous page)

```
    svm|lsvm|rf|slp|mlp|vc|gc|sc] [discogs|imdb|musicbrainz] [writer|producer
    |actor|director|band|audiovisual_work|musical_work|musician]
```

### Options

**-t, --threshold** <threshold>
Probability score threshold, default: 0.5.

**-n, --name-rule**
Activate post-classification rule on full names: links with different full names will be filtered.

**-u, --upload**
Upload links to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CLASSIFIER**
Required argument

**CATALOG**
Required argument

**ENTITY**
Required argument

### train

Train a supervised linker.

Build the training set relevant to the given catalog and entity, then train a model with the given classification algorithm.

```
train [OPTIONS] [naive_bayes|logistic_regression|support_vector_machines|linea
      r_support_vector_machines|random_forest|single_layer_perceptron|multi_la
      yer_perceptron|voting_classifier|gated_classifier|stacked_classifier|nb|
      lr|svm|lsvm|rf|slp|mlp|vc|gc|sc] [discogs|imdb|musicbrainz] [writer|prod
      ucer|actor|director|band|audiovisual_work|musical_work|musician]
```

### Options

**-t, --tune**
Run grid search for hyperparameters tuning.

**-k, --k-folds** <k_folds>
Number of folds for hyperparameters tuning. Use with '–tune'. Default: 5.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

#### Arguments

**CLASSIFIER**
Required argument

**CATALOG**
Required argument

**ENTITY**
Required argument

### 7.1.4 Pipeline

```
python -m soweego run
```

#### run

Launch the whole pipeline.

```
run [OPTIONS] [discogs|imdb|musicbrainz]
```

#### Options

**--validator, --no-validator**
Sync Wikidata to the target catalog. Default: no.

**--importer, --no-importer**
Import the target catalog dump into the database. Default: yes.

**--linker, --no-linker**
Link Wikidata items to target catalog identifiers. Default: yes.

**--upload, --no-upload**
Upload results to Wikidata. Default: yes.

#### Arguments

**CATALOG**
Required argument

### 7.1.5 Validator AKA Sync

```
python -m soweego sync
Usage: soweego sync [OPTIONS] COMMAND [ARGS]...

  Sync Wikidata to target catalogs.

Options:
  --help  Show this message and exit.

Commands:
```

(continues on next page)

(continued from previous page)

```
  bio    Validate identifiers against biographical data.
  ids    Check if identifiers are still alive.
  links  Validate identifiers against links.
  works  Generate statements about works by people.
```

### bio

Validate identifiers against biographical data.

Look for birth/death dates, birth/death places, gender.

Dump 2 output files:

1. target identifiers to be deprecated. Format: (JSON) {identifier: [list of QIDs]}
2. statements to be added. Format: (CSV) QID,metadata_PID,value

You can pass the '-u' flag to upload the output to Wikidata.

```
bio [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|band|
    audiovisual_work|musical_work|musician]
```

### Options

**-u, --upload**
Upload the output to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**--dump-wikidata**
Dump biographical data gathered from Wikidata to a JSON file.

**--dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

### ids

Check if identifiers are still alive.

Dump a JSON file of dead ones. Format: { identifier: [ list of QIDs ] }

Dead identifiers should get a deprecated rank in Wikidata: you can pass the '-d' flag to do so.

```
ids [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|band|
    audiovisual_work|musical_work|musician]
```

### Options

**-d, --deprecate**
Deprecate dead identifiers: this changes their rank in Wikidata.

**-s, --sandbox**
Perform all deprecations on the Wikidata sandbox item Q4115189.

**--dump-wikidata**
Dump identifiers gathered from Wikidata to a JSON file.

**--dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

### links

Validate identifiers against links.

Dump 3 output files:

1. target identifiers to be deprecated. Format: (JSON) {identifier: [list of QIDs]}
2. third-party identifiers to be added. Format: (CSV) QID,identifier_PID,identifier
3. URLs to be added. Format: (CSV) QID,P973,URL

You can pass the '-u' flag to upload the output to Wikidata.

```
links [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|ban
      d|audiovisual_work|musical_work|musician]
```

### Options

**-u, --upload**
Upload the output to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**--dump-wikidata**
Dump URLs gathered from Wikidata to a JSON file.

**--dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

### works

Generate statements about works by people.

Dump a CSV file of statements. Format: work_QID,PID,person_QID,person_catalog_ID

You can pass the '-u' flag to upload the statements to Wikidata.

```
works [OPTIONS] [discogs|imdb|musicbrainz] [writer|producer|actor|director|ban
      d|audiovisual_work|musical_work|musician]
```

### Options

**-u, --upload**
Upload statements to Wikidata.

**-s, --sandbox**
Perform all edits on the Wikidata sandbox item Q4115189.

**-d, --dir-io** <dir_io>
Input/output directory, default: /app/shared/.

### Arguments

**CATALOG**
Required argument

**ENTITY**
Required argument

CHAPTER

# EIGHT

# API DOCUMENTATION

## 8.1 importer

Import target catalog dumps into a SQL database.

### 8.1.1 base_dump_extractor

Base class for catalog dumps extraction.

**class** soweego.importer.base_dump_extractor.**BaseDumpExtractor**

Method definitions to download catalog dumps, extract data, and populate a database instance.

**extract_and_populate**(*dump_file_paths*, *resolve*)

Extract relevant data and populate SQLAlchemy ORM entities accordingly. Entities will be then persisted to a database instance.

**Parameters**

- **dump_file_paths** (List[str]) – paths to downloaded catalog dumps
- **resolve** (bool) – whether to resolve URLs found in catalog dumps or not

**Return type** None

**get_dump_download_urls**()

Get the dump download URLs of a target catalog. Useful if there is a way to compute the latest URLs.

**Return type** Optional[List[str]]

**Returns** the latest dumps URL

### 8.1.2 discogs_dump_extractor

Discogs dump extractor.

**class** soweego.importer.discogs_dump_extractor.**DiscogsDumpExtractor**

Download Discogs dumps, extract data, and populate a database instance.

**extract_and_populate**(*dump_file_paths*, *resolve*)

Extract relevant data from the *artists* (people) and *masters* (works) Discogs dumps, preprocess them, populate SQLAlchemy ORM entities, and persist them to a database instance.

See *discogs_entity* for the ORM definitions.

**Parameters**

- **dump_file_paths** (List[str]) – paths to downloaded catalog dumps
- **resolve** (bool) – whether to resolve URLs found in catalog dumps or not

**Return type** None

**get_dump_download_urls**()
Get the dump download URLs of a target catalog. Useful if there is a way to compute the latest URLs.

**Return type** Optional[List[str]]

**Returns** the latest dumps URL

### 8.1.3 imdb_dump_extractor

IMDb dump extractor.

**class** soweego.importer.imdb_dump_extractor.**IMDbDumpExtractor**
Download IMDb dumps, extract data, and populate a database instance.

**extract_and_populate**(*dump_file_paths*, *resolve*)
Extract relevant data from the *name* (people) and *title* (works) IMDb dumps, preprocess them, populate SQLAlchemy ORM entities, and persist them to a database instance.

See *imdb_entity* for the ORM definitions.

**Parameters**

- **dump_file_paths** (List[str]) – paths to downloaded catalog dumps
- **resolve** (bool) – whether to resolve URLs found in catalog dumps or not

**Return type** None

**get_dump_download_urls**()
Get the dump download URLs of a target catalog. Useful if there is a way to compute the latest URLs.

**Return type** List[str]

**Returns** the latest dumps URL

### 8.1.4 musicbrainz_dump_extractor

MusicBrainz dump extractor.

**class** soweego.importer.musicbrainz_dump_extractor.**MusicBrainzDumpExtractor**
Download MusicBrainz dumps, extract data, and populate a database instance.

**extract_and_populate**(*dump_file_paths*, *resolve*)
Extract relevant data from the *artist* (people) and *release group* (works) MusicBrainz dumps, preprocess them, populate SQLAlchemy ORM entities, and persist them to a database instance.

See *musicbrainz_entity* for the ORM definitions.

**Parameters**

- **dump_file_paths** (List[str]) – paths to downloaded catalog dumps
- **resolve** (bool) – whether to resolve URLs found in catalog dumps or not

**get_dump_download_urls**()
Get the dump download URLs of a target catalog. Useful if there is a way to compute the latest URLs.

**Return type** List[str]

**Returns** the latest dumps URL

### 8.1.5 importer

Download, extract, and import a supported catalog.

**class** soweego.importer.importer.**Importer**
Handle a catalog dump: check its freshness and dispatch the appropriate extractor.

**refresh_dump**(*output_folder*, *extractor*, *resolve*)

**Eventually download the latest dump, and call the** corresponding extractor.

**Parameters**

- **output_folder** (str) – a path where the downloaded dumps will be stored
- **extractor** (*BaseDumpExtractor*) – *BaseDumpExtractor* implementation to process the dump
- **resolve** (bool) – whether to resolve URLs found in catalog dumps or not

## 8.2 models

SQLAlchemy Object Relational Mapper (*ORM*) declarations, implemented as a set of classes.

All class attributes are Column objects representing columns of a SQL database table. Data types are detailed in the *Attributes* section of each class.

### 8.2.1 base_entity

Base SQLAlchemy ORM entities.

**class** soweego.importer.models.base_entity.**BaseEntity**(***kwargs*)
Minimal ORM structure for a target catalog entry. Each ORM entity should inherit this class.

**Attributes:**

- **internal_id** (integer) - an internal primary key
- **catalog_id** (string(50)) - a target catalog identifier
- **name** (text) - a full name (person), or full title (work)
- **name_tokens** (text) - a **name** tokenized through tokenize()
- **born** (date) - a birth (person), or publication (work) date
- **born_precision** (integer) - a birth (person), or publication (work) date precision
- **died** (date) - a death date. Only applies to a person
- **died_precision** (integer) - a death date precision. Only applies to a person

**class** soweego.importer.models.base_entity.**BaseRelationship**(*from_catalog_id*, *to_catalog_id*)
Minimal ORM structure for a target catalog relationship between entries. Each ORM relationship entity should implement this interface.

You can build a relationship for different purposes: typically, to connect works with people, or groups with individuals.

**Attributes:**

- **from_catalog_id** (string(50)) - a target catalog identifier
- **to_catalog_id** (string(50)) - a target catalog identifier

### 8.2.2 base_link_entity

Base SQLAlchemy ORM entity for URLs.

**class** soweego.importer.models.base_link_entity.**BaseLinkEntity**(***kwargs*)

Minimal ORM structure for a target catalog link/URL. Each ORM link entity should inherit this class.

**Attributes:**

- **internal_id** (integer) - an internal primary key
- **catalog_id** (string(50)) - a target catalog identifier
- **url** (text) - a full URL
- **is_wiki** (boolean) - whether a URL is a Wiki link or not
- **url_tokens** (text) - a **url** tokenized through `tokenize()`

### 8.2.3 base_nlp_entity

Base SQLAlchemy ORM entity for textual data that will undergo some natural language processing (*NLP*).

**class** soweego.importer.models.base_nlp_entity.**BaseNlpEntity**(***kwargs*)

Minimal ORM structure for a target catalog piece of text. Each ORM NLP entity should inherit this class.

**Attributes:**

- **internal_id** (integer) - an internal primary key
- **catalog_id** (string(50)) - a target catalog identifier
- **description** (text) - a text describing the main catalog entry
- **description_tokens** (text) - a **description** tokenized through `tokenize()`

### 8.2.4 discogs_entity

Discogs SQLAlchemy ORM entities.

**class** soweego.importer.models.discogs_entity.**DiscogsArtistEntity**(***kwargs*)

A Discogs *artist*: either a musician or a band. It comes from the `_artists.xml.gz` dataset. See the download page.

All ORM entities describing Discogs people should inherit this class.

**Attributes:**

- **real_name** (text) - a name in real life
- **data_quality** (string(20)) - an indicator of data quality

**class** soweego.importer.models.discogs_entity.**DiscogsGroupEntity**(***kwargs*)
A Discogs *group*, namely a band.

**class** soweego.importer.models.discogs_entity.**DiscogsGroupLinkEntity**(***kwargs*)
A Discogs band Web link (URL).

**class** soweego.importer.models.discogs_entity.**DiscogsGroupNlpEntity**(***kwargs*)
A Discogs band textual description.

**class** soweego.importer.models.discogs_entity.**DiscogsMasterArtistRelationship**(*from_catalog_id*, *to_catalog_id*)
A relationship between a Discogs musical work and the Discogs musician or band who made it.

**class** soweego.importer.models.discogs_entity.**DiscogsMasterEntity**(***kwargs*)
A Discogs *master*: a musical work, which can have multiple *releases*. It comes from the `_masters.xml.gz` dataset. See the download page.

**Attributes:**

- **main_release_id** (string(50)) - a Discogs identifier of the main release for this musical work
- **genres** (text) - a string list of musical genres

**class** soweego.importer.models.discogs_entity.**DiscogsMusicianEntity**(***kwargs*)
A Discogs musician.

**class** soweego.importer.models.discogs_entity.**DiscogsMusicianLinkEntity**(***kwargs*)
A Discogs musician Web link (URL).

**class** soweego.importer.models.discogs_entity.**DiscogsMusicianNlpEntity**(***kwargs*)
A Discogs musician textual description.

### 8.2.5 imdb_entity

IMDb SQLAlchemy ORM entities, based on the datasets specifications.

**class** soweego.importer.models.imdb_entity.**IMDbActorEntity**(***kwargs*)
An IMDb actor.

**class** soweego.importer.models.imdb_entity.**IMDbDirectorEntity**(***kwargs*)
An IMDb director.

**class** soweego.importer.models.imdb_entity.**IMDbMusicianEntity**(***kwargs*)
An IMDb musician.

**class** soweego.importer.models.imdb_entity.**IMDbNameEntity**(***kwargs*)
An IMDb *name*: a person like an actor, director, producer, etc. It comes from the `name.basics.tsv.gz` dataset. See the download page

All ORM entities describing IMDb people should inherit this class.

**Attributes**:

- **gender** (string(10)) - a gender
- **occupations** (string(255)) - a string list of Wikidata occupation QIDs

**class** soweego.importer.models.imdb_entity.**IMDbProducerEntity**(***kwargs*)
An IMDb producer.

**class** soweego.importer.models.imdb_entity.**IMDbTitleEntity**(***kwargs*)
An IMDb *title*: an audiovisual work like a movie, short, TV series episode, etc. It comes from the `title.basics.tsv.gz` dataset. See the download page

All ORM entities describing IMDb works should inherit this class.

**Attributes:**

- **title_type** (string(100)) - an audiovisual work type, like *movie* or *short*
- **primary_title** (text) - the most popular title
- **original_title** (text) - a title in the original language
- **is_adult** (boolean) - whether the audiovisual work is for adults or not
- **runtime_minutes** (integer) - a runtime in minutes
- **genres** (string(255)) - a string list of audiovisual genres

**class** soweego.importer.models.imdb_entity.**IMDbTitleNameRelationship**(*from_catalog_id*, *to_catalog_id*)

A relationship between an IMDb audiovisual work and an IMDb person who took part in it.

**class** soweego.importer.models.imdb_entity.**IMDbWriterEntity**(***kwargs*)

An IMDb writer.

### 8.2.6 musicbrainz_entity

MusicBrainz SQLAlchemy ORM entities, based on the database specifications.

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzArtistBandRelationship**(*from_catal*
*to_catalog_*

A membership between a MusicBrainz artist and a MusicBrainz band.

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzArtistEntity**(***kwargs*)

A MusicBrainz *artist*, namely a musician.

**Attributes:**

- **gender** (string(10)) - a gender
- **birth_place** (string(255)) - a birth place
- **death_place** (string(255)) - a death place

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzArtistLinkEntity**(***kwargs*)

A MusicBrainz musician Web link (URL).

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzBandEntity**(***kwargs*)

A MusicBrainz band.

**Attributes:**

- **birth_place** (string(255)) - a place where the band was formed
- **death_place** (string(255)) - a place where the band was disbanded

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzBandLinkEntity**(***kwargs*)

A MusicBrainz band Web link (URL).

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzReleaseGroupArtistRelationship**

A relationship between a MusicBrainz musical work and the MusicBrainz musician or band who made it.

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzReleaseGroupEntity**(***kwargs*)

A MusicBrainz *release group*: a musical work, which is a group of *releases*.

**class** soweego.importer.models.musicbrainz_entity.**MusicBrainzReleaseGroupLinkEntity**(***kwargs*)

A MusicBrainz musical work Web link (URL).

### 8.2.7 mix_n_match

Mix'n'match SQLAlchemy ORM entities for catalogs that need curation.

They follow the `catalog` and `entry` tables of the `s51434__mixnmatch_p` database located in ToolsDB under the Wikimedia Toolforge infrastructure. See how to connect.

**class** soweego.importer.models.mix_n_match.**MnMCatalog**(***kwargs*)
A Mix'n'match catalog.

**class** soweego.importer.models.mix_n_match.**MnMEntry**(***kwargs*)
A Mix'n'match entry.

## 8.3 ingester

Take soweego output into Wikidata items.

### 8.3.1 wikidata_bot

A Wikidata bot that adds, deletes, or deprecates referenced statements. Here are typical output examples.

***add_identifiers()***

*Claim:* Joey Ramone, Discogs artist ID, 264375
*Reference:* stated in, Discogs), (retrieved, TIMESTAMP

***add_people_statements()***

*Claim:* Joey Ramone, member of, Ramones
*Reference:* stated in, Discogs), (retrieved, TIMESTAMP

***add_works_statements()***

*Claim:* Leave Home, performer, Ramones
*Reference:* stated in, Discogs), (Discogs artist ID, 264375), (retrieved, TIMESTAMP

***delete_or_deprecate_identifiers()*** deletes or deprecates identifier statements.

soweego.ingester.wikidata_bot.**add_identifiers**(*identifiers*, *catalog*, *entity*, *sandbox*)
Add identifier statements to existing Wikidata items.

**Parameters**

- **identifiers** (dict) – a `{QID: catalog_identifier}` dictionary
- **catalog** (str) – `{'discogs', 'imdb', 'musicbrainz', 'twitter'}`. A supported catalog
- **entity** (str) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **sandbox** (bool) – whether to perform edits on the Wikidata sandbox item

**Return type** None

soweego.ingester.wikidata_bot.**add_people_statements**(*statements*, *sandbox*)
Add statements to existing Wikidata people.

Statements typically come from validation criteria 2 or 3 as per *soweego.validator.checks.links()* and *soweego.validator.checks.bio()*.

**Parameters**

- **statements** (Iterable[+T_co]) – iterable of (subject, predicate, value) triples
- **sandbox** (bool) – whether to perform edits on the Wikidata sandbox item

**Return type** None

soweego.ingester.wikidata_bot.**add_works_statements**(*statements*, *catalog*, *sandbox*)

Add statements to existing Wikidata works.

Statements typically come from *soweego.validator.enrichment.generate_statements()*.

**Parameters**

- **statements** (Iterable[+T_co]) – iterable of (work QID, predicate, person QID, person target ID) tuples
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz', 'twitter'}. A supported catalog
- **sandbox** (bool) – whether to perform edits on the Wikidata sandbox item

**Return type** None

soweego.ingester.wikidata_bot.**delete_or_deprecate_identifiers**(*action*, *catalog*, *entity*, *invalid*, *sandbox*)

Delete or deprecate invalid identifier statements from existing Wikidata items.

Deletion candidates come from validation criterion 1 as per *soweego.validator.checks.dead_ids()*.

Deprecation candidates come from validation criteria 2 or 3 as per *soweego.validator.checks.links()* and *soweego.validator.checks.bio()*.

**Parameters**

- **action** (str) – {'delete', 'deprecate'}
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz', 'twitter'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity
- **invalid** (dict) – a {invalid_catalog_identifier: [list of QIDs]} dictionary
- **sandbox** (bool) – whether to perform edits on the Wikidata sandbox item

**Return type** None

### 8.3.2 mix_n_match_client

A client that uploads non-confident links to the Mix'n'match tool for curation.

It inserts data in the catalog and entry tables of the s51434__mixnmatch_p database located in ToolsDB under the Wikimedia Toolforge infrastructure. See how to connect.

soweego.ingester.mix_n_match_client.**activate_catalog**(*catalog_id*, *catalog*, *entity*)
Activate a catalog.

**Parameters**

- **catalog_id** (int) – the catalog *id* field of the *catalog* table in the *s51434__mixnmatch_p* Toolforge database
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz', 'twitter'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity

**Return type** None

soweego.ingester.mix_n_match_client.**add_catalog**(*catalog*, *entity*)
Add or update a catalog.

**Parameters**

- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz', 'twitter'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity

**Return type** int

**Returns** the catalog *id* field of the *catalog* table in the *s51434__mixnmatch_p* Toolforge database

soweego.ingester.mix_n_match_client.**add_matches**(*file_path*, *catalog_id*, *catalog*, *entity*, *confidence_range*)
Add or update matches to an existing catalog. Curated matches found in the catalog are kept as is.

**Parameters**

- **file_path** (str) – path to a file with matches
- **catalog_id** (int) – the catalog *id* field of the *catalog* table in the *s51434__mixnmatch_p* Toolforge database
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz', 'twitter'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity
- **confidence_range** (Tuple[float, float]) – a pair of floats indicating the minimum and maximum confidence scores of matches that will be added/updated.

**Return type** None

## 8.4 linker

This is soweego's core, where Wikidata items get linked to target catalog identifiers.

### 8.4.1 workflow

Record linkage workflow. It is a pipeline composed of the following main steps:

1. build the Wikidata (*build_wikidata()*) and target (*build_target()*) datasets
2. preprocess both (*preprocess_wikidata()* and *preprocess_target()*)
3. extract features by comparing pairs of Wikidata and target values (*extract_features()*)

soweego.linker.workflow.**build_target**(*goal*, *catalog*, *entity*, *identifiers*)

Build a target catalog dataset for training or classification purposes: workflow step 1.

Data is gathered by querying the `s51434__mixnmatch_large_catalogs_p` database. This is where the `importer` inserts processed catalog dumps.

The database is located in ToolsDB under the Wikimedia Toolforge infrastructure. See how to connect.

**Parameters**

- **goal** (`str`) – `{'training', 'classification'}`. Whether to build a dataset for training or classification
- **catalog** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **entity** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **identifiers** (`Set[str]`) – a set of catalog IDs to gather data for

**Return type** `Iterator[DataFrame]`

**Returns** the generator yielding `pandas.DataFrame` chunks

soweego.linker.workflow.**build_wikidata**(*goal*, *catalog*, *entity*, *dir_io*)

Build a Wikidata dataset for training or classification purposes: workflow step 1.

Data is gathered from the SPARQL endpoint and the Web API.

**How it works:**

1. gather relevant Wikidata items that *hold* (for *training*) or *lack* (for *classification*) identifiers of the given catalog
2. gather relevant item data
3. dump the dataset to a gzipped JSON Lines file
4. read the dataset into a generator of `pandas.DataFrame` chunks for memory-efficient processing

**Parameters**

- **goal** (`str`) – `{'training', 'classification'}`. Whether to build a dataset for training or classification
- **catalog** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **entity** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **dir_io** (`str`) – input/output directory where working files will be read/written

**Return type** JsonReader

**Returns** the generator yielding pandas.DataFrame chunks

soweego.linker.workflow.**extract_features**(*candidate_pairs*, *wikidata*, *target*, *path_io*)

Extract feature vectors by comparing pairs of *(Wikidata, target catalog)* records.

**Main features:**

- exact match on full names and URLs
- match on tokenized names, URLs, and genres
- Levenshtein distance on name tokens
- string kernel similarity on name tokens
- weighted intersection on name tokens
- match on dates by maximum shared precision
- cosine similarity on textual descriptions
- match on occupation QIDs

See features for more details.

This function uses multithreaded parallel processing.

**Parameters**

- **candidate_pairs** (MultiIndex) – an index of *(QID, target ID)* pairs that should undergo comparison
- **wikidata** (DataFrame) – a preprocessed Wikidata dataset (typically a chunk)
- **target** (DataFrame) – a preprocessed target catalog dataset (typically a chunk)
- **path_io** (str) – input/output path to an extracted feature file

**Return type** DataFrame

**Returns** the feature vectors dataset

soweego.linker.workflow.**preprocess_target**(*goal*, *target_reader*)

Preprocess a target catalog dataset: workflow step 2.

This function consumes pandas.DataFrame chunks and should be pipelined after *build_target()*.

**Preprocessing actions:**

1. drop unneeded columns holding target DB primary keys
2. rename non-null catalog ID columns & drop others
3. drop columns with null values only
4. pair dates with their precision and drop precision columns when applicable
5. aggregate denormalized data on target ID
6. *(shared with preprocess_wikidata() )* normalize columns with names, occupations, dates, when applicable

**Parameters**

- **goal** (str) – {'training', 'classification'}. Whether the dataset is for training or classification

- **target_reader** (Iterator[DataFrame]) – a dataset reader as returned by *build_target()*

**Return type** DataFrame

**Returns** the generator yielding preprocessed pandas.DataFrame chunks

soweego.linker.workflow.**preprocess_wikidata**(*goal*, *wikidata_reader*)

Preprocess a Wikidata dataset: workflow step 2.

This function consumes pandas.DataFrame chunks and should be pipelined after *build_wikidata()*.

**Preprocessing actions:**

1. set QIDs as pandas.core.indexes.base.Index of the chunk
2. drop columns with null values only
3. *(training)* ensure one target ID per QID
4. tokenize names, URLs, genres, when applicable
5. *(shared with preprocess_target() )* normalize columns with names, occupations, dates, when applicable

**Parameters**

- **goal** (str) – {'training', 'classification'}. Whether the dataset is for training or classification
- **wikidata_reader** (JsonReader) – a dataset reader as returned by *build_wikidata()*

**Return type** Iterator[DataFrame]

**Returns** the generator yielding preprocessed pandas.DataFrame chunks

### 8.4.2 blocking

Custom blocking technique for the Record Linkage Toolkit, where blocking stands for *record pairs indexing*.

In a nutshell, blocking means finding *candidate pairs* suitable for comparison: this is essential to avoid blind comparison of all records, thus reducing the overall complexity of the task. In a supervised learning scenario, this translates into finding relevant training and classification *samples*.

Given a Wikidata pandas.Series (dataset column), this technique finds samples through full-text search in natural language mode against the target catalog database.

Target catalog identifiers of the output pandas.MultiIndex are also passed to *build_target()* for building the actual target dataset.

soweego.linker.blocking.**find_samples**(*goal*, *catalog*, *wikidata_column*, *chunk_number*, *target_db_entity*, *dir_io*)

Build a blocking index by looking up target catalog identifiers given a Wikidata dataset column. A meaningful column should hold strings.

Under the hood, run full-text search in *natural language mode* against the target catalog database.

This function uses multithreaded parallel processing.

**Parameters**

- **goal** (str) – {'training', 'classification'}. Whether the samples are for training or classification

- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz'}. A supported catalog
- **wikidata_column** (Series) – a Wikidata dataset column holding values suitable for full-text search against the target database
- **chunk_number** (int) – which Wikidata chunk will undergo blocking. Typically returned by calling enumerate() over *preprocess_wikidata()*
- **target_db_entity** (*~DB_ENTITY*) – an ORM entity (AKA table) of the target catalog database that full-text search should aim at
- **dir_io** (str) – input/output directory where index chunks will be read/written

**Return type** MultiIndex

**Returns** the blocking index holding candidate pairs

### 8.4.3 features

A set of custom features suitable for the Record Linkage Toolkit, where feature extraction stands for *record pairs comparison*.

**Input:** pairs of list objects coming from Wikidata and target catalog pandas.DataFrame columns as per *preprocess_wikidata()* and *preprocess_target()* output.

**Output:** a *feature vector* pandas.Series.

All classes in this module share the following constructor parameters:

- **left_on** (str) - a Wikidata column label
- **right_on** (str) - a target catalog column label
- **missing_value** - (optional) a score to fill null values
- **label** - (optional) a label for the output feature Series

Specific parameters are documented in the *__init__* method of each class.

All classes in this module implement recordlinkage.base.BaseCompareFeature, and can be added to the feature extractor object recordlinkage.Compare.

Usage:

```
>>> import recordlinkage as rl
>>> from soweego.linker import features
>>> extractor = rl.Compare()
>>> source_column, target_column = 'birth_name', 'fullname'
>>> feature = features.ExactMatch(source_column, target_column)
>>> extractor.add(feature)
```

**class** soweego.linker.features.**ExactMatch**(*left_on*, *right_on*, *match_value=1.0*, *non_match_value=0.0*, *missing_value=0.0*, *label=None*)

Compare pairs of lists through exact match on each pair of elements.

**__init__**(*left_on*, *right_on*, *match_value=1.0*, *non_match_value=0.0*, *missing_value=0.0*, *label=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label

- **right_on** (str) – a target catalog DataFrame column label
- **match_value** (float) – (optional) a score when element pairs match
- **non_match_value** (float) – (optional) a score when element pairs do not match
- **missing_value** (float) – (optional) a score to fill null values
- **label** (Optional[str]) – (optional) a label for the output feature Series

**class** soweego.linker.features.**SimilarStrings**(*left_on*, *right_on*, *algorithm='levenshtein'*, *threshold=None*, *missing_value=0.0*, *analyzer=None*, *ngram_range=(2, 2)*, *label=None*)

Compare pairs of lists holding **strings** through similarity measures on each pair of elements.

**__init__**(*left_on*, *right_on*, *algorithm='levenshtein'*, *threshold=None*, *missing_value=0.0*, *analyzer=None*, *ngram_range=(2, 2)*, *label=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label
- **right_on** (str) – a target catalog DataFrame column label
- **algorithm** (str) – (optional) {'cosine', 'levenshtein'}. A string similarity algorithm, either the cosine similarity or the Levenshtein distance respectively
- **threshold** (Optional[float]) – (optional) a threshold to filter features with a lower or equal score
- **missing_value** (float) – (optional) a score to fill null values
- **analyzer** (Optional[str]) – (optional, only applies when *algorithm='cosine'*) {'soweego', 'word', 'char', 'char_wb'}. A text analyzer to preprocess input. It is passed to the *analyzer* parameter of sklearn.feature_extraction.text.CountVectorizer.
  - 'soweego' is soweego.commons.text_utils.tokenize()
  - {'word', 'char', 'char_wb'} are *scikit* built-ins. See here for more details
  - None is str.split(), and means input is already preprocessed
- **ngram_range** (Tuple[int, int]) – (optional, only applies when *algorithm='cosine'* and *analyzer* is not *'soweego'*). Lower and upper boundary for n-gram extraction, passed to CountVectorizer
- **label** (Optional[str]) – (optional) a label for the output feature Series

**class** soweego.linker.features.**SimilarDates**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

Compare pairs of lists holding **dates** through match by maximum shared precision.

**__init__**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label
- **right_on** (str) – a target catalog DataFrame column label
- **missing_value** (float) – (optional) a score to fill null values
- **label** (Optional[str]) – (optional) a label for the output feature Series

**class** soweego.linker.features.**SharedTokens**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

Compare pairs of lists holding **string tokens** through weighted intersection.

**__init__**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label
- **right_on** (str) – a target catalog DataFrame column label
- **missing_value** (float) – (optional) a score to fill null values
- **label** (Optional[str]) – (optional) a label for the output feature Series

**class** soweego.linker.features.**SharedOccupations**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

Compare pairs of lists holding **occupation QIDs** *(ontology classes)* through expansion of the class hierarchy, plus intersection of values.

**__init__**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label
- **right_on** (str) – a target catalog DataFrame column label
- **missing_value** (float) – (optional) a score to fill null values
- **label** (Optional[str]) – (optional) a label for the output feature Series

**class** soweego.linker.features.**SharedTokensPlus**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*, *stop_words=None*)

Compare pairs of lists holding **string tokens** through weighted intersection.

This feature is similar to *SharedTokens*, but has extra functionality:

- handles arbitrary stop words
- accepts nested list of tokens
- output score is the percentage of tokens in the smallest set which are shared among both sets

**__init__**(*left_on*, *right_on*, *missing_value=0.0*, *label=None*, *stop_words=None*)

**Parameters**

- **left_on** (str) – a Wikidata DataFrame column label
- **right_on** (str) – a target catalog DataFrame column label
- **missing_value** (float) – (optional) a score to fill null values
- **label** (Optional[str]) – (optional) a label for the output feature Series
- **stop_words** (Optional[Set[~T]]) – (optional) a set of stop words to be filtered from input pairs

### 8.4.4 classifiers

A set of custom supervised classifiers suitable for the Record Linkage Toolkit. It includes neural networks and support-vector machines.

All classes implement recordlinkage.base.BaseClassifier: typically, you will use its fit(), predict(), and prob() methods.

**class** soweego.linker.classifiers.**GatedEnsembleClassifier**(*num_features*, ***kwargs*)

Ensemble of classifiers, whose predictions are joined by using a further meta-learner, which decides the final output based on the prediction of the base classifiers.

This classifier uses `mlens.ensemble.SuperLearner` to implement the *gating* functionality.

The parameters, and their default values, are:

- **meta_layer: Name of the classifier to use as a *meta layer*. By** default this is *single_layer_perceptron*
- **folds: The number of folds to use for cross validation when** generating the training set for the **meta_layer**. The default value for this is *2*.

  For a better explanation of this parameter, see:

  *Polley, Eric C. and van der Laan, Mark J., "Super Learner In Prediction" (May 2010). U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 266* https://biostats.bepress.com/ucbbiostat/paper266/

**class** soweego.linker.classifiers.**MultiLayerPerceptron**(*num_features*, ***kwargs*)

A multi-layer perceptron classifier.

This class implements a keras.Sequential model with the following default architecture:

- Dense layer 1, with `128` output dimension and `relu` activation function
- BatchNormalization layer
- Dense layer 2, with `32` output dimension and `relu` activation function
- BatchNormalization layer
- Dense layer 3, with `1` output dimension and `sigmoid` activation function
- `adadelta` optimizer
- `binary_crossentropy` loss function
- `accuracy` metric for evaluation

If you want to override the default parameters, you can pass the following keyword arguments to the constructor:

- **activations** - a triple with values for *(dense layer 1, dense layer 2, dense layer 3)*. See available activations
- **optimizer** - see optimizers
- **loss** - see available loss functions
- **metrics** - see available metrics

**class** soweego.linker.classifiers.**RandomForest**(**args*, ***kwargs*)

A Random Forest classifier.

This class implements `sklearn.ensemble.RandomForestClassifier`, and receives the same parameters.

It fits multiple decision trees on sub-samples (aka, parts) of the dataset and averages the result to get more accuracy and reduce over-fitting.

The default parameters are:

- **n_estimators**: 500
- **criterion**: entropy
- **max_features**: None
- **bootstrap**: True

**prob**(*feature_vectors*)
Classify record pairs and include the probability score of being a match.

**Parameters** **feature_vectors** (DataFrame) – a DataFrame computed via record pairs comparison. This should be recordlinkage.Compare.compute() output. See *extract_features()* for more details

**Return type** Series

**Returns** the classification results

**class** soweego.linker.classifiers.**SVCClassifier**(**args*, ***kwargs*)
A support-vector machine classifier.

This class implements sklearn.svm.SVC, which is based on the libsvm library.

This classifier differs from recordlinkage.classifiers.SVMClassifier, which implements sklearn.svm.LinearSVC, based on the liblinear library.

Main highlights:

- output probability scores
- can use non-linear kernels
- higher training time (quadratic to the number of samples)

**prob**(*feature_vectors*)
Classify record pairs and include the probability score of being a match.

**Parameters** **feature_vectors** (DataFrame) – a DataFrame computed via record pairs comparison. This should be recordlinkage.Compare.compute() output. See *extract_features()* for more details

**Return type** Series

**Returns** the classification results

**class** soweego.linker.classifiers.**SingleLayerPerceptron**(*num_features*, ***kwargs*)
A single-layer perceptron classifier.

This class implements a keras.Sequential model with the following default architecture:

- single Dense layer
- sigmoid activation function
- adam optimizer
- binary_crossentropy loss function
- accuracy metric for evaluation

If you want to override the default parameters, you can pass the following keyword arguments to the constructor:

- **activation** - see available activations
- **optimizer** - see optimizers
- **loss** - see available loss functions
- **metrics** - see available metrics

**class** soweego.linker.classifiers.**StackedEnsembleClassifier**(*num_features*, ***kwargs*)
Ensemble of stacked classifiers, meaning that classifiers are arranged in layers with the next layer getting as input the output of the last layer. The predictions of the final layer are merged with a meta-learner (the same happens

for ~:class:*soweego.linker.GatedEnsembleClassifier*), which decides the final output based on the prediction of the base classifiers.

This classifier uses `mlens.ensemble.SuperLearner` to implement the *stacking* functionality.

The parameters, and their default values, are:

- **meta_layer: Name of the classifier to use as a *meta layer*. By** default this is *single_layer_perceptron*
- **folds: The number of folds to use for cross validation when** generating the training set for the **meta_layer**. The default value for this is *2*.

  For a better explanation of this parameter, see:

  *Polley, Eric C. and van der Laan, Mark J., "Super Learner In Prediction" (May 2010). U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 266* https://biostats.bepress.com/ucbbiostat/paper266/

**class** soweego.linker.classifiers.**VotingClassifier**(*num_features*, ***kwargs*)

A basic ensemble classifier which uses a voting procedure to decide the final outcome of a prediction.

This class implements `sklearn.ensemble.VotingClassifier`.

It combines a set of classifiers and uses majority vote or average predicted probabilities to pick the final prediction. See scikit's user guide.

The parameter *voting* can have as values either **"hard"** or **"soft"**.

- **hard - the label predicted by the majority of base classifiers is used as the** final prediction. Note that this does not return probabilities, only the final label.
- **soft - the probability that a pair is a match is taken from all base classifiers** and then averaged. This average is what is returned by the classifier.

By default *voting=soft*.

**prob**(*feature_vectors*)

Classify record pairs and include the probability score of being a match.

**Parameters** **feature_vectors** (`DataFrame`) – a `DataFrame` computed via record pairs comparison. This should be `recordlinkage.Compare.compute()` output. See *`extract_features()`* for more details

**Return type** `Series`

**Returns** the classification results

### 8.4.5 train

Train supervised linking algorithms.

soweego.linker.train.**build_training_set**(*catalog*, *entity*, *dir_io*)

Build a training set.

**Parameters**

- **catalog** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **entity** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **dir_io** (`str`) – input/output directory where working files will be read/written

**Return type** Tuple[DataFrame, MultiIndex]

**Returns** the feature vectors and positive samples pair. Features are computed by comparing *(QID, catalog ID)* pairs. Positive samples are catalog IDs available in Wikidata

soweego.linker.train.**execute**(*classifier*, *catalog*, *entity*, *tune*, *k*, *dir_io*, ***kwargs*)

Train a supervised linker.

1. Build the training set relevant to the given catalog and entity
2. train a model with the given classifier

**Parameters**

- **classifier** (str) – {'naive_bayes', 'linear_support_vector_machines', 'support_vector_machines', 'single_layer_perceptron', 'multi_layer_perceptron'}. A supported classifier
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity
- **tune** (bool) – whether to run grid search for hyperparameters tuning or not
- **k** (int) – number of folds for hyperparameters tuning. It is used only when *tune=True*
- **dir_io** (str) – input/output directory where working files will be read/written
- **kwargs** – extra keyword arguments that will be passed to the model initialization

**Return type** BaseClassifier

**Returns** the trained model

### 8.4.6 link

Run supervised linkers.

soweego.linker.link.**execute**(*model_path*, *catalog*, *entity*, *threshold*, *name_rule*, *dir_io*)

Run a supervised linker.

1. Build the classification set relevant to the given catalog and entity
2. generate links between Wikidata items and catalog identifiers

**Parameters**

- **model_path** (str) – path to a trained model file
- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity
- **threshold** (float) – minimum confidence score for generated links. Those below this value are discarded. Must be a float between 0 and 1
- **name_rule** (bool) – whether to enable the rule on full names or not: if *True*, links with different full names are discarded after classification

- **dir_io** (str) – input/output directory where working files will be read/written

**Return type** Iterator[Series]

**Returns** the generator yielding chunks of links

### 8.4.7 evaluate

Evaluate supervised linking algorithms.

## 8.5 validator

Sync Wikidata to target catalogs and enrich items when extra data is available.

### 8.5.1 checks

A set of checks to validate Wikidata against target catalogs.

soweego.validator.checks.**bio**(*catalog*, *entity*, *wd_cache=None*)

Validate identifiers against available biographical data.

Look for:

- birth and death dates
- birth and death places
- gender

Also generate statements based on additional data found in the given catalog. They can be used to enrich Wikidata items.

**How it works:**

1. gather data from the given catalog
2. gather data from relevant Wikidata items
3. look for shared data between pairs of Wikidata and catalog items:

- when the pair does not share any data, the catalog identifier should be marked with a deprecated rank
- when the catalog item has more data than the Wikidata one, it should be added to the latter

**Parameters**

- **catalog** (str) – {'discogs', 'imdb', 'musicbrainz'}. A supported catalog
- **entity** (str) – {'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}. A supported entity
- **wd_cache** – (optional) a dict of links gathered from Wikidata in a previous run

**Return type** Tuple[Defaultdict[~KT, ~VT], Iterator[+T_co], Dict[~KT, ~VT]]

**Returns**

3 objects

1. `dict` of identifiers that should be deprecated
2. `generator` of statements that should be added
3. `dict` of biographical data gathered from Wikidata

`soweego.validator.checks.`**`dead_ids`**(*catalog*, *entity*, *wd_cache=None*)

Look for dead identifiers in Wikidata. An identifier is dead if it does not exist in the given catalog when this function is executed.

Dead identifiers should be marked with a deprecated rank in Wikidata.

**How it works:**

1. gather identifiers of the given catalog from relevant Wikidata items
2. look them up in the given catalog
3. if an identifier is not in the given catalog anymore, it should be deprecated

**Parameters**

- **`catalog`** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **`entity`** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **`wd_cache`** – (optional) a `dict` of identifiers gathered from Wikidata in a previous run

**Return type** `Tuple`[`Defaultdict`[~KT, ~VT], `Dict`[~KT, ~VT]]

**Returns** the `dict` pair of dead identifiers and identifiers gathered from Wikidata

`soweego.validator.checks.`**`links`**(*catalog*, *entity*, *wd_cache=None*)

Validate identifiers against available links.

Also generate statements based on additional links found in the given catalog. They can be used to enrich Wikidata items.

**How it works:**

1. gather links from the given catalog
2. gather links from relevant Wikidata items
3. look for shared links between pairs of Wikidata and catalog items:

- when the pair does not share any link, the catalog identifier should be marked with a deprecated rank
- when the catalog item has more links than the Wikidata one, they should be added to the latter

4. try to extract third-party identifiers from extra links

**Parameters**

- **`catalog`** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **`entity`** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity

- **wd_cache** – (optional) a `dict` of links gathered from Wikidata in a previous run

**Return type** `Tuple`[`Defaultdict`[~KT, ~VT], `List`[~T], `List`[~T], `Dict`[~KT, ~VT]]

**Returns**

4 objects

1. `dict` of identifiers that should be deprecated
2. `list` of third-party identifiers that should be added
3. `list` of URLs that should be added
4. `dict` of links gathered from Wikidata

### 8.5.2 enrichment

Enrichment of Wikidata based on data available in target catalogs.

`soweego.validator.enrichment.`**generate_statements**(*catalog*, *entity*, *bucket_size=5000*)

Generate statements about works by people.

**How it works:**

1. gather works and people identifiers of the given catalog from relevant Wikidata items
2. leverage catalog relationships between works and people
3. build Wikidata statements accordingly

**Parameters**

- **catalog** (`str`) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **entity** (`str`) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **bucket_size** (`int`) – (optional) how many target IDs should be looked up in the given catalog. For efficiency purposes

**Return type** `Iterator`[`Tuple`]

**Returns** the statements `generator`, yielding *(work_QID, PID, person_QID, person_catalog_ID)* `tuple` s

## 8.6 wikidata

Collect data from Wikidata through the SPARQL endpoint and the Web API.

### 8.6.1 api_requests

Set of specific Web API requests for Wikidata data collection.

soweego.wikidata.api_requests.**build_session**
Build the HTTP session for interaction with the Wikidata API.

Log in if credentials are found, otherwise go ahead with an unauthenticated session. If a previously cached session has expired, build a new one.

**Return type** requests.Session

**Returns** the HTTP session to interact with the Wikidata API

soweego.wikidata.api_requests.**get_biodata**(*qids*)
Collect biographical data for a given set of Wikidata items.

**Parameters** **qids** (Set[str]) – a set of QIDs

**Return type** Iterator[Tuple[str, str, str]]

**Returns** the generator yielding `(QID, PID, value)` triples

soweego.wikidata.api_requests.**get_data_for_linker**(*catalog*, *entity*, *qids*, *url_pids*, *ext_id_pids_to_urls*, *qids_and_tids*, *fileout*)
Collect relevant data for linking Wikidata to a given catalog. Dump the result to a given output stream.

This function uses multithreaded parallel processing.

**Parameters**

- **catalog** (str) – `{'discogs', 'imdb', 'musicbrainz'}`. A supported catalog
- **entity** (str) – `{'actor', 'band', 'director', 'musician', 'producer', 'writer', 'audiovisual_work', 'musical_work'}`. A supported entity
- **qids** (Set[str]) – a set of QIDs
- **url_pids** (Set[str]) – a set of PIDs holding URL values. Returned by *soweego.wikidata.sparql_queries.url_pids()*
- **ext_id_pids_to_urls** (Dict[~KT, ~VT]) – a `{PID: {formatter_URL: formatter_regex} }` dict. Returned by *soweego.wikidata.sparql_queries.external_id_pids_and_urls()*
- **fileout** (Textio) – a file stream open for writing
- **qids_and_tids** (Dict[~KT, ~VT]) – a `{QID: {'tid': {catalog_ID_set} }` dict. Populated by `soweego.commons.data_gathering.gather_target_ids()`

**Return type** None

soweego.wikidata.api_requests.**get_links**(*qids*, *url_pids*, *ext_id_pids_to_urls*)
Collect sitelinks and third-party links for a given set of Wikidata items.

**Parameters**

- **qids** (Set[str]) – a set of QIDs
- **url_pids** (Set[str]) – a set of PIDs holding URL values. Returned by *soweego.wikidata.sparql_queries.url_pids()*
- **ext_id_pids_to_urls** (Dict[~KT, ~VT]) – a `{PID: {formatter_URL: formatter_regex} }` dict. Returned by *soweego.wikidata.sparql_queries.external_id_pids_and_urls()*

**Return type** Iterator[Tuple]

**Returns** the generator yielding `(QID, URL)` pairs

soweego.wikidata.api_requests.**parse_value**(*value*)
Parse a value returned by the Wikidata API into standard Python objects.

The parser supports the following Wikidata data types:

- string > *str*
- URL > *str*
- monolingual text > *str*
- time > *tuple* `(time, precision)`
- item > *set* `{item_labels}`

**Parameters** **value** (Union[str, Dict[~KT, ~VT]]) – a data value from a call to the Wikidata API

**Return type** Union[str, Tuple[str, str], Set[str], None]

**Returns** the parsed Python object, or `None` if parsing failed

### 8.6.2 sparql_queries

Set of specific SPARQL queries for Wikidata data collection.

soweego.wikidata.sparql_queries.**external_id_pids_and_urls**()
Retrieve Wikidata properties holding identifier values, together with their formatter URLs and regular expressions.

**Return type** Iterator[Dict[~KT, ~VT]]

**Returns** the generator yielding `{PID: {formatter_URL: formatter_regex} }` dicts

soweego.wikidata.sparql_queries.**random**() → x in the interval [0, 1).

soweego.wikidata.sparql_queries.**run_query**(*query_type*, *class_qid*, *catalog_pid*, *result_per_page*)
Run a filled SPARQL query template against the Wikidata endpoint with eventual paging.

**Parameters**

- **query_type** (Tuple[str, str]) – a pair with one of `{'identifier', 'links', 'dataset', 'biodata'}` and `{'class', 'occupation'}`
- **class_qid** (str) – a Wikidata ontology class, like Q5
- **catalog_pid** (str) – a Wikidata property for identifiers, like P1953
- **result_per_page** (int) – a page size. Use `0` to switch paging off

**Return type** Iterator[Union[Tuple, str]]

**Returns** the query result generator, yielding `(QID, identifier_or_URL)` pairs, or `QID` strings only, depending on *query_type*

soweego.wikidata.sparql_queries.**subclasses_of**(*qid*)
Retrieve subclasses of a given Wikidata ontology class.

**Parameters** **qid** (str) – a Wikidata ontology class, like Q5

**Return type** Set[str]

**Returns** the QIDs of subclasses

soweego.wikidata.sparql_queries.**superclasses_of**(*qid*)
Retrieve superclasses of a given Wikidata ontology class.

**Parameters** **qid** (str) – a Wikidata ontology class, like Q5

**Return type** Set[str]

**Returns** the QIDs of superclasses

soweego.wikidata.sparql_queries.**url_pids**()
Retrieve Wikidata properties holding URL values.

**Return type** Iterator[str]

**Returns** the PIDs generator

CHAPTER

NINE

# CONTRIBUTE

**Note:** the best way is to *Import a new catalog*.

Please also have a look here:

## 9.1 Contribution guidelines

### 9.1.1 Workflow

1. Fork this repository;
2. follow the project structure;
3. commit **frequently** with **clear** messages;
4. send a pull request to the `master` branch of this repository.

### 9.1.2 Coding

1. **Style** - comply with PEP 8 and Wikimedia conventions:
   - *4 spaces* for indentation;
   - *snake-case* style AKA *underscore* as a word separator (files, variables, functions);
   - *UPPERCASE* constants;
   - anything else is *lowercase*;
   - *2* empty lines to separate functions;
   - *80* characters per line, and up to *100* when suitable;
   - *single-quoted* strings, unless single-quotes are in a string.
2. Type hints - add them at least to public function signatures;
3. **Documentation** - write Sphinx docstrings at least for public functions and classes:
   - use reST markup;
   - stick to PEP 257 and PEP 287;
   - pay special attention to info field lists;
   - cross-reference Python objects.

4. **Refactoring**:
    - fix pylint errors: `pylint -j 0 -E PATH_TO_YOUR_CONTRIBUTION`;
    - look at pylint warnings: `pylint -j 0 -d all -e W PATH_TO_YOUR_CONTRIBUTION`;
    - reduce complexity: `flake8 --select C90 --max-complexity 10 PATH_TO_YOUR_CONTRIBUTION`;
    - apply relevant refactoring suggestions: `pylint -j 0 -d all -e R PATH_TO_YOUR_CONTRIBUTION`.

CHAPTER

TEN

# EXPERIMENTS & NOTES

## 10.1 Experiments

### 10.1.1 Default evaluation technique

Applies to all experiments:

- stratified 5-fold cross validation over training/test splits;
- mean performance scores over the folds.

#### Single-layer perceptron optimizers

https://github.com/Wikidata/soweego/issues/285

#### Setting

- run: May 3 2019;
- output folder: `soweego-2.eqiad.wmflabs:/srv/dev/20190503/`;
- head commit: d0d390e622f2782a49a1bd0ebfc64478ed34aa0c;
- command: `python -m soweego linker evaluate slp ${Dataset} ${Entity} optimizer=${Optimizer}`.

#### Discogs band

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .782 | .945 | .856 |
| rmsprop | .801 | .930 | .860 |
| nadam | .805 | .925 | .861 |
| adamax | .795 | .938 | .861 |
| adam | .800 | .929 | .860 |
| adagrad | .802 | .927 | .859 |
| adadelta | .799 | .934 | .861 |

### Discogs musician

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .815 | .985 | .892 |
| rmsprop | .816 | .985 | .893 |
| nadam | .816 | .986 | .893 |
| adamax | .817 | .985 | .893 |
| adam | .816 | .985 | .893 |
| adagrad | .816 | .986 | .893 |
| adadelta | .815 | .986 | .892 |

### IMDb director

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .918 | .954 | .936 |
| rmsprop | .895 | .954 | .923 |
| nadam | .908 | .954 | .930 |
| adamax | .907 | .955 | .930 |
| adam | .909 | .953 | .931 |
| adagrad | .867 | .950 | .907 |
| adadelta | .902 | .954 | .927 |

### IMDb musician

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .912 | .927 | .920 |
| rmsprop | .913 | .929 | .921 |
| nadam | .913 | .929 | .921 |
| adamax | .913 | .928 | .921 |
| adam | .913 | .928 | .921 |
| adagrad | .873 | .860 | .866 |
| adadelta | .913 | .928 | .921 |

### IMDb producer

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .917 | .942 | .929 |
| rmsprop | .916 | .938 | .927 |
| nadam | .916 | .938 | .927 |
| adamax | .916 | .940 | .928 |
| adam | .916 | .938 | .927 |
| adagrad | .852 | .684 | .756 |
| adadelta | .916 | .939 | .928 |

#### IMDb writer

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .929 | .943 | .936 |
| rmsprop | .927 | .940 | .934 |
| nadam | .930 | .940 | .935 |
| adamax | .930 | .941 | .935 |
| adam | .930 | .940 | .935 |
| adagrad | .872 | .923 | .896 |
| adadelta | .931 | .941 | .936 |

#### MusicBrainz band

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .952 | .869 | .909 |
| rmsprop | .949 | .875 | .911 |
| nadam | .949 | .877 | .911 |
| adamax | .952 | .871 | .910 |
| adam | .951 | .875 | .911 |
| adagrad | .932 | .886 | .909 |
| adadelta | .952 | .874 | .911 |

#### MusicBrainz musician

| Optimizer | Precision | Recall | F-score |
|---|---|---|---|
| sgd | .942 | .957 | .949 |
| rmsprop | .941 | .958 | .949 |
| nadam | .941 | .958 | .949 |
| adamax | .941 | .958 | .949 |
| adam | .941 | .958 | .949 |
| adagrad | .946 | .953 | .950 |
| adadelta | .941 | .958 | .950 |

#### Takeaways

- All optimizers seem to do a similar job;
- no specific impact on the performance.

### 10.1.2 Max Levenshtein VS average Levenshtein

https://github.com/Wikidata/soweego/issues/176

#### Setting

- run: May 7 2019;

- output folder: `soweego-2.eqiad.wmflabs:/srv/dev/20190507/`;
- head commit: ddd5d719793ea217267413a52d1d2e5b90c341a7;
- command: `python -m soweego linker evaluate ${Algorithm} ${Dataset} ${Entity}`.

### Discogs band

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .787 | .955 | **.863** |
| nb avg | .789 | .941 | .859 |
| lsvm max | .780 | .960 | **.861** |
| lsvm avg | .785 | .946 | .858 |
| svm max | .777 | .963 | .860 |
| svm avg | .777 | .963 | .860 |
| slp max | .784 | .954 | **.861** |
| slp avg | .776 | .956 | .857 |
| mlp max | .822 | .925 | .870 |

### Discogs musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .831 | .975 | **.897** |
| nb avg | .836 | .958 | .893 |
| lsvm max | .818 | .985 | **.894** |
| lsvm avg | .814 | .986 | .892 |
| svm max | .815 | .985 | .892 |
| svm avg | .815 | .985 | .892 |
| slp max | .821 | .983 | **.895** |
| slp avg | .815 | .985 | .892 |
| mlp max | .852 | .963 | .904 |

### IMDb director

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .896 | .971 | .932 |
| nb avg | .897 | .971 | .932 |
| lsvm max | .919 | .943 | **.931** |
| lsvm avg | .919 | .942 | .930 |
| svm max | .911 | .950 | .930 |
| svm avg | .908 | .958 | **.932** |
| slp max | .917 | .953 | **.935** |
| slp avg | .867 | .953 | .908 |
| mlp max | .913 | .964 | .938 |

### IMDb musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .889 | .962 | .924 |
| nb avg | .891 | .960 | .924 |
| lsvm max | .917 | .938 | .927 |
| lsvm avg | .917 | .937 | .927 |
| svm max | .904 | .944 | .924 |
| svm avg | .908 | .942 | .924 |
| slp max | .924 | .929 | **.926** |
| slp avg | .922 | .914 | .918 |
| mlp max | .912 | .951 | .931 |

### IMDb producer

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .870 | .971 | .918 |
| nb avg | .871 | .970 | .918 |
| lsvm max | .920 | .940 | **.930** |
| lsvm avg | .920 | .938 | .929 |
| svm max | .923 | .927 | .925 |
| svm avg | .923 | .926 | .925 |
| slp max | .914 | .940 | **.927** |
| slp avg | .862 | .914 | .883 |
| mlp max | .911 | .956 | .933 |

### IMDb writer

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .904 | .975 | **.938** |
| nb avg | .910 | .961 | .935 |
| lsvm max | .936 | .949 | **.943** |
| lsvm avg | .936 | .948 | .942 |
| svm max | .932 | .954 | .943 |
| svm avg | .932 | .954 | .943 |
| slp max | .938 | .946 | **.942** |
| slp avg | .903 | .955 | .928 |
| mlp max | .930 | .963 | .946 |

#### MusicBrainz band

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .821 | .987 | .896 |
| nb avg | .822 | .985 | .896 |
| lsvm max | .944 | .879 | .910 |
| lsvm avg | .943 | .888 | **.914** |
| svm max | .930 | .891 | .910 |
| svm avg | .939 | .893 | **.915** |
| slp max | .953 | .865 | .907 |
| slp avg | .930 | .885 | .907 |
| mlp max | .906 | .918 | .911 |

#### MusicBrainz musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb max | .955 | .936 | .946 |
| nb avg | .955 | .936 | .946 |
| lsvm max | .941 | .963 | .952 |
| lsvm avg | .941 | .962 | .952 |
| svm max | .951 | .938 | .944 |
| svm avg | .950 | .938 | .944 |
| slp max | .942 | .957 | .949 |
| slp avg | .943 | .956 | .949 |
| mlp max | .939 | .970 | .954 |

#### Takeaways

Max Levenshtein has the following impact:

- NB is always improved or left untouched;
- LSVM is always improved, left untouched for IMDb director, but worsens for MusicBrainz band;
- SVM is often left untouched, but worsens for IMDb director and MusicBrainz band;
- SLP is always improved with the highest impact, left untouched for MusicBrainz;
- **conclusion:** max Levenshtein should replace the average one.

### 10.1.3 String kernel feature

https://github.com/Wikidata/soweego/issues/174

#### Setting

- run: May 8 2019;
- output folder: `soweego-2.eqiad.wmflabs:/srv/dev/20190508/`;
- head commit: 0c5137fc4fe446abdb6df6dbde277b7aa15881c5;

- command: `python -m soweego linker evaluate ${Algorithm} ${Dataset} ${Entity}`.

### Discogs band

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .788 | **.942** | .859 |
| nb | .789 | .941 | .859 |
| lsvm +sk | .785 | .946 | .858 |
| lsvm | .785 | .946 | .858 |
| svm +sk | .778 | .963 | **.861** |
| svm | .777 | .963 | .860 |
| slp +sk | **.783** | .947 | .857 |
| slp | .776 | **.956** | .857 |
| mlp +sk | .848 | .913 | .879 |

### Discogs musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .836 | .958 | .893 |
| nb | .836 | .958 | .893 |
| lsvm +sk | **.816** | .985 | .892 |
| lsvm | .814 | **.986** | .892 |
| svm +sk | .815 | .985 | .892 |
| svm | .815 | .985 | .892 |
| slp +sk | **.820** | .978 | .892 |
| slp | .815 | **.985** | .892 |
| mlp +sk | .868 | .948 | .906 |

### IMDb director

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .897 | .971 | .932 |
| nb | .897 | .971 | .932 |
| lsvm +sk | .923 | .949 | **.935** |
| lsvm | .919 | .942 | .930 |
| svm +sk | **.914** | .950 | .931 |
| svm | .908 | .958 | **.932** |
| slp +sk | **.918** | .955 | **.936** |
| slp | .867 | .953 | .908 |
| mlp +sk | .918 | .964 | .941 |

### IMDb musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .891 | **.961** | .924 |
| nb | .891 | .960 | .924 |
| lsvm +sk | .922 | .941 | **.931** |
| lsvm | .917 | .937 | .927 |
| svm +sk | .910 | .949 | **.929** |
| svm | .908 | .942 | .924 |
| slp +sk | .922 | .934 | **.928** |
| slp | .922 | .914 | .918 |
| mlp +sk | .914 | .958 | .935 |

### IMDb producer

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .871 | .970 | .918 |
| nb | .871 | .970 | .918 |
| lsvm +sk | .921 | .943 | **.932** |
| lsvm | .920 | .938 | .929 |
| svm +sk | .923 | **.927** | .925 |
| svm | .923 | .926 | .925 |
| slp +sk | .916 | .942 | **.929** |
| slp | .862 | .914 | .883 |
| mlp +sk | .912 | .959 | .935 |

### IMDb writer

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .910 | .961 | .935 |
| nb | .910 | .961 | .935 |
| lsvm +sk | .938 | .953 | **.945** |
| lsvm | .936 | .948 | .942 |
| svm +sk | .933 | .957 | **.945** |
| svm | .932 | .954 | .943 |
| slp +sk | .939 | .948 | **.943** |
| slp | .903 | **.955** | .928 |
| mlp +sk | .931 | .968 | .949 |

#### MusicBrainz band

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .821 | .985 | .896 |
| nb | **.822** | .985 | .896 |
| lsvm +sk | .940 | .895 | **.917** |
| lsvm | **.943** | .888 | .914 |
| svm +sk | .937 | .899 | **.918** |
| svm | **.939** | .893 | .915 |
| slp +sk | .952 | .873 | **.911** |
| slp | .930 | **.885** | .907 |
| mlp +sk | .937 | .904 | .920 |

#### MusicBrainz musician

| Algorithm | Precision | Recall | F-score |
|---|---|---|---|
| nb +sk | .955 | .936 | .946 |
| nb | .955 | .936 | .946 |
| lsvm +sk | .938 | **.965** | .951 |
| lsvm | .941 | .962 | **.952** |
| svm +sk | **.951** | .938 | .944 |
| svm | .950 | .938 | .944 |
| slp +sk | .941 | .958 | **.950** |
| slp | **.943** | .956 | .949 |
| mlp +sk | .939 | .972 | .955 |

#### Takeaways

The string kernel feature:

- has the most positive impact on SLP;
- slightly improves performance in most cases, but sligthly worsens:
  - precision in 1 case, i.e., NB for MusicBrainz band;
  - recall in 3 cases, i.e., SLP for Discogs band, LSVM & SLP for Discogs musician;
  - f-score in 2 cases, i.e., SVM for IMDb director, LSVM for MusicBrainz musician.
- **conclusion**: the string kernel feature should be added.

## 10.2 Evaluations

### 10.2.1 Setting

- run: April 11 2019 on `soweego-1` VPS instance;
- output folder: `/srv/dev/20190411`;
- head commit: 1505429997b878568a9e24185dc3afa7ad4720eb;

- command: `python -m soweego linker evaluate ${Algorithm} ${Dataset} ${Entity}`;
- evaluation technique: stratified 5-fold cross validation over training/test splits;
- mean performance scores over the folds.

### 10.2.2 Algorithms parameters

- Naïve Bayes (NB):
  - binarize = 0.1;
  - alpha = 0.0001;
- `liblinear` SVM (LSVM): default parameters as per scikit LinearSVC;
- `libsvm` SVM (SVM):
  - kernel = linear;
  - other parameters as per scikit SVC defaults;
- single-layer perceptron (SLP):
  - layer = fully connected (`Dense`);
  - activation = sigmoid;
  - optimizer = stochastic gradient descent;
  - loss = binary cross-entropy;
  - training batch size = 1,024;
  - training epochs = 100.
- multi-layer perceptron (MLP):
  - layers = 128 > BN > 32 > BN > 1
    - fully connected layers followed by BatchNormalization (BN)
  - activation:
    - hidden layers = relu;
    - output layer = sigmoid;
  - optimizer = Adadelta;
  - loss = binary cross-entropy
  - training batch size = 1,024;
  - training epochs = 1000;
  - early stopping:
    - patience = 100;

### 10.2.3 Performance

| Algorithm | Dataset | Entity | Precision (std) | Recall (std) | F-score (std) |
|---|---|---|---|---|---|
| NB | Discogs | Band | .789 (.0031) | .941 (.0004) | .859 (.002) |
| LSVM | Discogs | Band | .785 (.0058) | .946 (.0029) | .858 (.0034) |
| SVM | Discogs | Band | .777 (.003) | .963 (.0016) | .86 (.0024) |
| SLP | Discogs | Band | .776 (.0041) | .956 (.0012) | .857 (.0029) |
| NB | Discogs | Musician | .836 (.0018) | .958 (.0012) | .893 (.0013) |
| SVM | Discogs | Musician | .814 (.0015) | .986 (.0003) | .892 (.001) |
| SLP | Discogs | Musician | .815 (.002) | .985 (.0006) | .892 (.0012) |
| NB | IMDb | Actor | TODO | TODO | TODO |
| SVM | IMDb | Actor | TODO | TODO | TODO |
| SLP | IMDb | Actor | TODO | TODO | TODO |
| MLP | IMDb | Actor | TODO | TODO | TODO |
| NB | IMDb | Director | .897 (.00195) | .971 (.0012) | .932 (.001) |
| SVM | IMDb | Director | .919 (.0031) | .942 (.0019) | .93 (.002) |
| SLP | IMDb | Director | .867 (.0115) | .953 (.0043) | .908 (.0056) |
| NB | IMDb | Musician | .891 (.0042) | .96 (.0022) | .924 (.0026) |
| SVM | IMDb | Musician | .917 (.0043) | .937 (.0034) | .927 (.003) |
| SLP | IMDb | Musician | .922 (.005) | .914 (.0092) | .918 (.0055) |
| NB | IMDb | Producer | .871 (.0023) | .97 (.0037) | .918 (.0011) |
| SVM | IMDb | Producer | .92 (.005) | .938 (.0038) | .929 (.0026) |
| SLP | IMDb | Producer | .862 (.0609) | .914 (.0648) | .883 (.0185) |
| NB | IMDb | Writer | .91 (.003) | .961 (.0022) | .935 (.0022) |
| SVM | IMDb | Writer | .936 (.0029) | .948 (.0025) | .942 (.0026) |
| SLP | IMDb | Writer | .903 (.0154) | .955 (.0147) | .928 (.0047) |
| NB | MusicBrainz | Band | .822 (.00169) | .985 (.0008) | .896 (.001) |
| SVM | MusicBrainz | Band | .943 (.0019) | .888 (.0027) | .914 (.0016) |
| SLP | MusicBrainz | Band | .93 (.0265) | .885 (.0103) | .907 (.0082) |
| NB | MusicBrainz | Musician | .955 (.0009) | .936 (.0011) | .946 (.00068) |
| SVM | MusicBrainz | Musician | .941 (.0011) | .962 (.001) | .952 (.0004) |
| SLP | MusicBrainz | Musician | .943 (.0018) | .956 (.0019) | .949 (.0007) |

### 10.2.4 Confidence

The following plots display the confidence scores distribution and the total predictions yielded by each algorithm on each target classification set.

Note that linear SVM is omitted since it does not output probability scores.

Axes:

- x = # predictions;
- y = confidence score.

#### Discogs band

NB, SVM, SLP. MLP

#### Discogs musician

NB, SVM, SLP. MLP

#### IMDb director

NB, SVM, SLP. MLP

#### IMDb musician

NB, SVM, SLP. MLP

#### IMDb producer

NB, SVM, SLP. MLP

#### IMDb writer

NB, SVM, SLP. MLP

#### MusicBrainz band

NB, SVM, SLP. MLP

#### MusicBrainz musician

NB, SVM, SLP. MLP

### 10.2.5 Comparison

See the plots above to have a rough idea on the amount of confident predictions.

Threshold values:

- # predictions >= 0.0000000001, i.e., equivalent to almost all matches;
- # confident >= 0.8.

#### Discogs band

WD items: 50,316

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .789 | .785 | .777 | .776 | .833 |
| Recall | .941 | .946 | .963 | .957 | .914 |
| F-score | .859 | .858 | .86 | .857 | .872 |
| # predictions | 820 | 51 | 94,430 | 91,295 | 91,132 |
| # confident | 219 | N.A. | 1,660 | 5,355 | 11,114 |

### Discogs musician

WD items: 199,180

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .836 | .814 | .815 | .815 | .849 |
| Recall | .958 | .986 | .985 | .985 | .961 |
| F-score | .893 | .892 | .892 | .892 | .902 |
| # predictions | 3,872 | 200 | 533,301 | 517,450 | 514,488 |
| # confident | 1,101 | N.A. | 98,172 | 58,437 | 57,184 |

### IMDb director

WD items: 9,249

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .897 | .919 | .908 | .867 | .916 |
| Recall | .971 | .942 | .958 | .953 | .961 |
| F-score | .932 | .93 | .932 | .908 | .938 |
| # predictions | 192 | 10 | 17,557 | 17,187 | 16,881 |
| # confident | 60 | N.A. | 1,616 | 553 | 1,810 |

### IMDb musician

WD items: 217,139

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .891 | .917 | .908 | .922 | .903 |
| Recall | .96 | .937 | .942 | .914 | .951 |
| F-score | .924 | .927 | .924 | .918 | .926 |
| # predictions | 4,806 | 218 | 406,674 | 398,346 | 376,857 |
| # confident | 1,341 | N.A. | 21,462 | 7,244 | 16,272 |

### IMDb producer

WD items: 2,251

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .871 | .92 | .923 | .862 | .912 |
| Recall | .97 | .938 | .926 | .914 | .956 |
| F-score | .918 | .929 | .925 | .883 | .933 |
| # predictions | 56 | 3 | 5,249 | 5,116 | 5,094 |
| # confident | 15 | N.A. | 507 | 180 | 529 |

### IMDb writer

WD items: 16,446

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .91 | .936 | .932 | .903 | .921 |
| Recall | .961 | .948 | .954 | .955 | .962 |
| F-score | .935 | .942 | .943 | .928 | .941 |
| # predictions | 428 | 17 | 45,122 | 44,338 | 43,868 |
| # confident | 138 | N.A. | 2,934 | 1,548 | 3,234 |

#### MusicBrainz band

WD items: 32,658

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .822 | .943 | .939 | .93 | .933 |
| Recall | .985 | .888 | .893 | .885 | .902 |
| F-score | .896 | .914 | .915 | .907 | .918 |
| # predictions | 265 | 33 | 39,618 | 38,012 | 33,981 |
| # confident | 46 | N.A. | 1,475 | 501 | 1,506 |

#### MusicBrainz musician

WD items: 153,725

| Measure | NB | LSVM | SVM | SLP | MLP |
|---|---|---|---|---|---|
| Precision | .955 | .941 | .95 | .943 | .940 |
| Recall | .936 | .962 | .938 | .956 | .968 |
| F-score | .946 | .952 | .944 | .949 | .954 |
| # predictions | 2,833 | 154 | 280,029 | 260,530 | 194,505 |
| # confident | 1,212 | N.A. | 7,496 | 7,339 | 8,470 |

## 10.3 Notes on the *recordlinkage* library

https://recordlinkage.readthedocs.io/

### 10.3.1 General

- uses `pandas` for data structures, typically the `DataFrame`, `Series`, and `MultiIndex` classes
- https://pandas.pydata.org/pandas-docs/stable/dsintro.html
- https://pandas.pydata.org/pandas-docs/stable/advanced.html
- uses `jellyfish` under the hood for edit distances and phonetic algorithms.

### 10.3.2 Data format

- https://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe
- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

- uses `pandas.DataFrame` to represent datasets. It's basically a table with column headers
- conversion from a `dict` is easy: key = column header, value = cell
- a value is a list, so `defaultdict(list)` is helpful

```
dataset = pandas.DataFrame(
  {
    'catalog_id': [666, 777, 888],
    'name': ['huey', 'dewey', 'louie'],
    ...
  }
)
```

- remember the order of values, i.e., `666` -> `'huey'`

### 10.3.3 Cleaning

- AKA **pre-processing** AKA **normalization** AKA **standardization**;
- https://recordlinkage.readthedocs.io/en/latest/ref-preprocessing.html
- uses `pandas.Series`, a list-like object
- the `clean` function seems interesting at a first glimpse
- by default, it **removes text inside brackets**. Might be useful, but also trivial to re-implement
- terrible default regex, **removes everything that is not an ASCII letter!** Non-ASCII strings are just deleted! Use a custom regex or `None` in `replace_by_none=` kwarg to avoid this
- nice ASCII folding via `strip_accents='ascii'`, **not done** by default
- `strip_accents='unicode'` keeps intact some Unicode chars, e.g., œ
- non-latin scripts are just not handled
- the `phonetic` function has the same problems as in `jellyfish`, see #79.

```
>>> import pandas
>>> from recordlinkage.preprocessing import clean
>>> names = pandas.Series(
>>>  [
>>>    ', ',
>>>    'charles hartshorne',
>>>    '',
>>>    ' ',
>>>    ' ',
>>>    'àáâäæãåāèéêëēėęîïíīįìôöòóœøōõûüùúū'
>>>  ]
>>> )
>>> clean(names)
0
1    charles hartshorne
2
3
4
5
dtype: object
>>> clean(names, replace_by_none=None, strip_accents='ascii')
0                                   ,
```

(continues on next page)

(continued from previous page)

```
1                 charles hartshorne
2
3
4
5    aaaaaaaeeeeeeeiiiiiiooooooouuuuu
dtype: object
```

### 10.3.4 Indexing

- AKA **blocking** AKA **candidate acquisition**
- https://recordlinkage.readthedocs.io/en/latest/ref-index.html
- make pairs of records to reduce the space complexity (quadratic)
- a simple call to the `Index.block(FIELD)` function is not enough for names, as it makes pairs that **exactly** agree, i.e., **like an exact match**

```
>>> import recordlinkage
>>> index = recordlinkage.Index()
>>> index.block('name')
>>> candidate_pairs = index.index(source_dataset, target_dataset)
```

- we could inject the MariaDB full-text index #126 as a **user-defined algorithm**;
- https://recordlinkage.readthedocs.io/en/latest/ref-index.html#user-defined-algorithms
- https://recordlinkage.readthedocs.io/en/latest/ref-index.html#examples

### 10.3.5 Comparing

- AKA **feature extraction**
- https://recordlinkage.readthedocs.io/en/latest/ref-compare.html
- probably useful for #143;
- the `Compare.date` function can be useful for dates: https://recordlinkage.readthedocs.io/en/latest/ref-compare.html#recordlinkage.compare.Date
- the `Compare.string` function implements `jellyfish` string edit distances and others: https://recordlinkage.readthedocs.io/en/latest/ref-compare.html#recordlinkage.compare.String
- the string **edit distance feature** is **binary**, not **scalar**: `feature_vectors.sum(1).value_counts()` below shows that
- the `threshold` kwarg gives a binary score for pairs above or below its value, i.e., `1` or `0`. **It's not really a threshold**
- not clear how the feature is fired by default, i.e., `threshold=None`
- better always use the `threshold` kwarg then, typically `3` for Levenshtein and `0.85` for Jaro-Winkler

```
>>> import recordlinkage
>>> comp = recordlinkage.Compare()
>>> comp.string('name', 'label', threshold=3)
>>> feature_vectors = comp.compute(candidate_pairs, source_dataset, target_dataset)
>>> print(feature_vectors.sum(1).value_counts())
```

### 10.3.6 Classification

- train with `fit(training_feature_vectors, match_index)`
- classify with `predict(classification_feature_vectors)`
- we could give SVM a try: https://recordlinkage.readthedocs.io/en/latest/notebooks/classifiers.html#Support-Vector-Machines
- adapters are especially useful: https://recordlinkage.readthedocs.io/en/latest/ref-classifiers.html#adapters
- **it is possible to inject a neural network with "keras"**: https://recordlinkage.readthedocs.io/en/latest/ref-classifiers.html#recordlinkage.adapters.KerasAdapter
- remember to set comparison of fields with missing values to `0`, i.e., pair disagreement:
  - *Most classifiers can not handle comparison vectors with missing values.*
  - no worries, `compare.string` does that by default

### 10.3.7 Training workflow

INPUT = training set = existing QIDs with target IDs = dict `{ QID: target_ID }`.

1. get the QID statements from Wikidata
2. query MariaDB for target ID data
3. load both into 2 `pandas.DataFrame`
4. pre-process
5. make the index with blocking -> `match_index` arg
6. feature extraction with comparison -> `training_feature_vectors` arg.

### 10.3.8 Naïve Bayes

- https://recordlinkage.readthedocs.io/en/latest/ref-classifiers.html#recordlinkage.NaiveBayesClassifier
- https://recordlinkage.readthedocs.io/en/latest/notebooks/classifiers.html
- **code example** at https://github.com/J535D165/recordlinkage/blob/master/examples/supervised_learning_prob.py
- `recordlinkage.NaiveBayesClassifier` class
- works with **binary features**, also explains why the edit distance feature is binary
- the `binarize` kwarg translates into a threshold: features above and below this value become `1` and `0` respectively
- the code example uses `binary_vectors` and sets toy `m` and `u` probabilities:
  1. are comparison vectors (point 6 of the training workflow) the expected input?
  2. should we compute `m` and `u` on our own as well?

CHAPTER

ELEVEN

# LICENSE

The source code is under the terms of the GNU General Public License, version 3.

# PYTHON MODULE INDEX

## S

# INDEX

## Symbols



## A

## B



## C



## D

## E



## F



## G



## I

## L



## M



## P



## R

## S



## T



## U



## V



## W