
gunshotmatch-pipeline

Release 0.9.1

GunShotMatch Analysis Pipeline

Dominic Davis-Foster

May 10, 2024

Contents

1	Installation	1
1.1	from PyPI	1
1.2	from GitHub	1
I	API Reference	3
2	gunshotmatch_pipeline	5
2.1	prepare_datafile	5
2.2	project_from_repeats	5
3	gunshotmatch_pipeline.config	7
3.1	Configuration	7
4	gunshotmatch_pipeline.decision_tree	9
4.1	DecisionTreeVisualiser	9
4.2	data_from_projects	11
4.3	data_from_unknown	11
4.4	dotsafe_name	11
4.5	fit_decision_tree	12
4.6	get_feature_names	12
4.7	predict_unknown	12
4.8	simulate_data	12
4.9	visualise_decision_tree	13
4.10	gunshotmatch_pipeline.decision_tree.export	14
4.11	gunshotmatch_pipeline.decision_tree.predictions	16
5	gunshotmatch_pipeline.exporters	17
5.1	verify_saved_datafile	17
5.2	verify_saved_project	17
5.3	write_combined_csv	17
5.4	write_matches_json	18
6	gunshotmatch_pipeline.nist_ms_search	19
6.1	LazyEngine	19
6.2	PyMSNISTSearchCfg	20
6.3	engine_on_demand	20
6.4	nist_ms_search_engine	20
7	gunshotmatch_pipeline.peaks	21
7.1	align_and_filter_peaks	21
7.2	prepare_peak_list	21

8	gunshotmatch_pipeline.projects	23
8.1	GlobalSettings	23
8.2	LoaderMixin	24
8.3	ProjectSettings	24
8.4	Projects	26
8.5	process_projects	27
9	gunshotmatch_pipeline.results	29
9.1	Matches	29
9.2	MatchesCompounds	29
9.3	MatchesMetadata	30
9.4	compounds	30
9.5	compounds_from_matches	30
9.6	machine_learning_data	30
9.7	matches	30
9.8	unknown	31
9.9	unknown_machine_learning_data	31
10	gunshotmatch_pipeline.unknowns	33
10.1	UnknownSettings	33
10.2	filter_and_identify_peaks	35
10.3	process_unknown	35
11	gunshotmatch_pipeline.utils	37
11.1	project_plural	37
11.2	unknown_plural	37
11.3	friendly_name_mapping	37
11.4	NameMapping	37
II	Contributing	39
12	Contributing	41
12.1	Coding style	41
12.2	Automated tests	41
12.3	Type Annotations	41
12.4	Build documentation locally	42
13	Downloading source code	43
13.1	Building from source	44
14	License	45
	Python Module Index	47
	Index	49

Installation

1.1 from PyPI

```
$ python3 -m pip install gunshotmatch-pipeline --user
```

1.2 from GitHub

```
$ python3 -m pip install git+https://github.com/GunShotMatch/gunshotmatch-pipeline@master --user
```


Part I

API Reference

gunshotmatch_pipeline

GunShotMatch Pipeline.

Functions:

<code>prepare_datafile(filename, method[, verbose])</code>	Pipeline from raw datafile to a <code>Datafile</code> .
<code>project_from_repeats(repeats, name, method, ...)</code>	Construct a project from the given <code>Repeat</code> objects, using the given method.

prepare_datafile (*filename, method, verbose=False*)

Pipeline from raw datafile to a `Datafile`.

Parameters

- **filename** (`Union[str, Path, PathLike]`)
- **method** (`Method`)
- **verbose** (`bool`) – If `True` information about the GC-MS data in each datafile will be printed. Default `False`.

Return type `Tuple[Repeat, GCMS_data]`

project_from_repeats (*repeats, name, method, engine*)

Construct a project from the given `Repeat` objects, using the given method.

Parameters

- **repeats** (`List[Repeat]`)
- **name** (`str`) – The project name.
- **method** (`Method`)
- **engine** (`Engine`) – NIST MS Search engine.

Return type `Project`

gunshotmatch_pipeline.config

Configuration for GunShotMatch analysis.

class Configuration (*pyms_nist_search*)

Bases: *MethodBase*

Overall GunShotMatch configuration.

Methods:

<i>from_json</i> (<i>json_string</i>)	Parse a <i>Configuration</i> from a JSON string.
<i>from_toml</i> (<i>toml_string</i>)	Parse a <i>Configuration</i> from a TOML string.
<i>to_toml</i> ()	Convert a <i>Configuration</i> to a TOML string.

Attributes:

<i>pyms_nist_search</i>	Configuration for <i>pyms_nist_search</i> .
-------------------------	---

classmethod from_json (*json_string*)

Parse a *Configuration* from a JSON string.

Parameters **json_string** (*str*)

Return type *Configuration*

classmethod from_toml (*toml_string*)

Parse a *Configuration* from a TOML string.

Parameters **toml_string** (*str*)

Return type *Configuration*

pyms_nist_search

Type: *PyMSNISTSearchCfg*

Configuration for *pyms_nist_search*.

to_toml ()

Convert a *Configuration* to a TOML string.

Return type *str*

gunshotmatch_pipeline.decision_tree

Prepare data and train decision trees.

Classes:

<code>DecisionTreeVisualiser(classifier, ...)</code>	Class for exporting visualisations of a decision tree or random forest.
--	---

Functions:

<code>data_from_projects(projects[, normalize])</code>	Returns a <code>DataFrame</code> containing decision tree data for the given projects.
<code>data_from_unknown(unknown, feature_names[, ...])</code>	Returns a <code>DataFrame</code> containing decision tree data for the given unknown.
<code>dotsafe_name(name)</code>	Return a dot (graphviz) suitable name for a sample, with special characters escaped.
<code>fit_decision_tree(data, classifier)</code>	Fit the classifier to the data.
<code>get_feature_names(data)</code>	Return the feature names for the given data.
<code>predict_unknown(unknown, classifier, ...)</code>	Predict classes for an unknown sample from a decision tree or random forest.
<code>simulate_data(project[, normalize, n_simulated])</code>	Generate simulated peak area data for a project.
<code>visualise_decision_tree(data, classifier, ...)</code>	Visualise a decision tree with graphviz.

class DecisionTreeVisualiser (*classifier, feature_names, factorize_map*)

Bases: `object`

Class for exporting visualisations of a decision tree or random forest.

New in version 0.8.0.

Parameters

- **classifier** (`ClassifierMixin`) – Decision tree or random forest classifier.
- **feature_names** (`List[str]`) – The compounds the decision tree was trained on.
- **factorize_map** (`List[str]`) – List of class names in the order they appear as classes in the classifier.

Methods:

<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__getstate__()</code>	Used for pickling.
<code>__ne__(other)</code>	Return <code>self != other</code> .
<code>__repr__()</code>	Return a string representation of the <code>DecisionTreeVisualiser</code> .

continues on next page

Table 3 – continued from previous page

<code>__setattr__(name, val)</code>	Implement <code>setattr(self, name)</code> .
<code>__setstate__(state)</code>	Used for pickling.
<code>from_data(data, classifier, factorize_map)</code>	Alternative constructor from the pandas dataframe the classifier was trained on.
<code>visualise_tree([filename, filetype])</code>	Visualise the decision tree or random forest as an image.

Attributes:

<code>classifier</code>	Decision tree or random forest classifier.
<code>factorize_map</code>	List of class names in the order they appear as classes in the classifier.
<code>feature_names</code>	The compounds the decision tree was trained on.

`__eq__(other)`
Return `self == other`.

Return type `bool`

`__getstate__()`
Used for pickling.
Automatically created by attrs.

`__ne__(other)`
Return `self != other`.

Return type `bool`

`__repr__()`
Return a string representation of the *DecisionTreeVisualiser*.

Return type `str`

`__setattr__(name, val)`
Implement `setattr(self, name)`.

`__setstate__(state)`
Used for pickling.
Automatically created by attrs.

classifier

Type: `ClassifierMixin`

Decision tree or random forest classifier.

factorize_map

Type: `List[str]`

List of class names in the order they appear as classes in the classifier.

feature_names

Type: `List[str]`

The compounds the decision tree was trained on.

classmethod from_data (*data*, *classifier*, *factorize_map*)

Alternative constructor from the pandas dataframe the classifier was trained on.

Return type `DecisionTreeVisualiser`

visualise_tree (*filename*='decision_tree_graphivz', *filetype*='svg')

Visualise the decision tree or random forest as an image.

Parameters

- **filename** (`str`) – Output filename without extension; for random forest, the base filename (followed by `-tree-n`). Default `'decision_tree_graphivz'`.
- **filetype** (`str`) – Output filetype (e.g. `svg`, `png`, `pdf`). Default `'svg'`.

data_from_projects (*projects*, *normalize*=`False`)

Returns a `DataFrame` containing decision tree data for the given projects.

Parameters

- **projects** (`Projects`)
- **normalize** (`bool`) – Default `False`.

Return type `Tuple[DataFrame, List[str]]`

data_from_unknown (*unknown*, *feature_names*, *normalize*=`False`)

Returns a `DataFrame` containing decision tree data for the given unknown.

Parameters

- **unknown** (`UnknownSettings`)
- **feature_names** (`Collection[str]`) – The compounds the decision tree was trained on. Extra compounds in the unknown will be excluded.
- **normalize** (`bool`) – Default `False`.

Return type `DataFrame`

dotsafe_name (*name*)

Return a dot (graphviz) suitable name for a sample, with special characters escaped.

Parameters **name** (`str`)

Return type `str`

New in version 0.5.0.

fit_decision_tree (*data*, *classifier*)

Fit the classifier to the data.

Parameters

- **data** (*DataFrame*)
- **classifier** (*ClassifierMixin*)

Return type *List[str]*

Returns List of feature names

get_feature_names (*data*)

Return the feature names for the given data.

Parameters **data** (*DataFrame*)

Return type *List[str]*

predict_unknown (*unknown*, *classifier*, *factorize_map*, *feature_names*)

Predict classes for an unknown sample from a decision tree or random forest.

Parameters

- **unknown** (*UnknownSettings*)
- **classifier** (*ClassifierMixin*)
- **factorize_map** (*List[str]*) – List of class names in the order they appear as classes in the classifier.
- **feature_names** (*List[str]*) – The compounds the decision tree was trained on. Extra compounds in the unknown will be excluded.

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

Returns An iterator of predicted class names and their probabilities, ranked from most to least likely.

New in version 0.9.0.

simulate_data (*project*, *normalize=False*, *n_simulated=10*)

Generate simulated peak area data for a project.

Parameters

- **project** (*Project*)
- **normalize** (*bool*) – Default *False*.
- **n_simulated** (*int*) – The number of values to simulate. Default 10.

Return type *DataFrame*

visualise_decision_tree (*data*, *classifier*, *factorize_map*, *filename*='decision_tree_graphivz',
 filetype='svg')

Visualise a decision tree with graphviz.

Parameters

- **data** (`DataFrame`)
- **classifier** (`ClassifierMixin`)
- **factorize_map** (`List[str]`) – List of class names in the order they appear as classes in the classifier.
- **filename** (`str`) – Output filename without extension; for random forest, the base filename (followed by `-tree-n`). Default `'decision_tree_graphivz'`.
- **filetype** (`str`) – Output filetype (e.g. `svg`, `png`, `pdf`). Default `'svg'`.

4.10 gunshotmatch_pipeline.decision_tree.export

Export and load decision trees to/from JSON-safe dictionaries..

New in version 0.6.0.

Functions:

<code>serialise_decision_tree(model)</code>	Serialise a decision tree to a JSON-safe dictionary.
<code>deserialise_decision_tree(model_dict)</code>	Deserialise a decision tree.
<code>verify_saved_decision_tree(in_process, from_file)</code>	Verify the saved <code>DecisionTreeClassifier</code> matches the model in memory.
<code>serialise_random_forest(model)</code>	Serialise a random forest to a JSON-safe dictionary.
<code>deserialise_random_forest(model_dict)</code>	Deserialise a random forest.
<code>verify_saved_random_forest(in_process, from_file)</code>	Verify the saved <code>RandomForestClassifier</code> matches the model in memory.

serialise_decision_tree (*model*)

Serialise a decision tree to a JSON-safe dictionary.

Parameters `model` (`DecisionTreeClassifier`) – Trained decision tree.

Return type `Dict[str, Any]`

deserialise_decision_tree (*model_dict*)

Deserialise a decision tree.

Parameters `model_dict` (`Dict[str, Any]`) – JSON-safe representation of the decision tree.

Return type `DecisionTreeClassifier`

verify_saved_decision_tree (*in_process, from_file*)

Verify the saved `DecisionTreeClassifier` matches the model in memory.

Will raise an `AssertionError` if the data do not match.

Parameters

- **in_process** (`DecisionTreeClassifier`) – The `DecisionTreeClassifier` already in memory.
- **from_file** (`DecisionTreeClassifier`) – A `DecisionTreeClassifier` loaded from disk.

New in version 0.7.0.

serialise_random_forest (*model*)

Serialise a random forest to a JSON-safe dictionary.

Parameters `model` (`RandomForestClassifier`) – Trained random forest.

Return type `Dict[str, Any]`

deserialise_random_forest (*model_dict*)

Deserialise a random forest.

Parameters `model_dict` (`Dict[str, Any]`) – JSON-safe representation of the random forest.

Return type `RandomForestClassifier`

verify_saved_random_forest (*in_process, from_file*)

Verify the saved `RandomForestClassifier` matches the model in memory.

Will raise an `AssertionError` if the data do not match.

Parameters

- **in_process** (`RandomForestClassifier`) – The `RandomForestClassifier` already in memory.
- **from_file** (`RandomForestClassifier`) – A `RandomForestClassifier` loaded from disk.

New in version 0.7.0.

4.11 gunshotmatch_pipeline.decision_tree.predictions

Represents random forest classifier predictions for testing classifier performance.

New in version 0.9.0.

Classes:

<code>PredictionResult(name, class_name, predictions)</code>	Represents the predicted classes from a random forest classifier.
--	---

Functions:

<code>dump_predictions(predictions[, indent])</code>	Return a JSON representation of the predictions.
<code>load_predictions(predictions_json)</code>	Load predictions from the given JSON string.

namedtuple PredictionResult (*name, class_name, predictions*)

Bases: `NamedTuple`

Represents the predicted classes from a random forest classifier.

Fields

- 0) **name** (`str`) – the sample name e.g. “Unknown Western Double A”
- 1) **class_name** (`str`) – i.e. the ammo type e.g. “Western Double A”
- 2) **predictions** (`Tuple[Tuple[str, float], ...]`) – Tuples of (<class name>, <probability>).

property correct

Returns whether the top prediction matches the actual class name.

Return type `bool`

`__repr__()`

Return a nicely formatted representation string

dump_predictions (*predictions, indent=2*)

Return a JSON representation of the predictions.

Parameters

- **predictions** (`List[PredictionResult]`)
- **indent** (`Optional[int]`) – Default 2.

Return type `str`

load_predictions (*predictions_json*)

Load predictions from the given JSON string.

Parameters **predictions_json** (`str`)

Return type `List[PredictionResult]`

gunshotmatch_pipeline.exporters

Functions and classes for export to disk, and verification of saved data.

Functions:

<code>verify_saved_datafile(in_process, from_file)</code>	Verify the data in a saved <code>Datafile</code> matches the data in memory.
<code>verify_saved_project(in_process, from_file)</code>	Verify the data in a saved <code>Project</code> matches the data in memory.
<code>write_combined_csv(repeat, output_dir)</code>	Write a CSV file listing the top hits for each peak in the <code>Repeat</code> , with associated data.
<code>write_matches_json(project, output_dir)</code>	Write the JSON output file listing the determined “best match” for each peaks.

verify_saved_datafile (*in_process, from_file*)

Verify the data in a saved `Datafile` matches the data in memory.

Will raise an `AssertionError` if the data do not match.

Parameters

- **in_process** (`Datafile`) – The `Datafile` already in memory.
- **from_file** (`Datafile`) – A `Datafile` loaded from disk.

verify_saved_project (*in_process, from_file*)

Verify the data in a saved `Project` matches the data in memory.

Will raise an `AssertionError` if the data do not match.

Parameters

- **in_process** (`Project`) – The `Project` already in memory.
- **from_file** (`Project`) – A `Project` loaded from disk.

write_combined_csv (*repeat, output_dir*)

Write a CSV file listing the top hits for each peak in the `Repeat`, with associated data.

Parameters

- **project**
- **output_dir** (`PathPlus`) – Directory to save the file in

write_matches_json (*project*, *output_dir*)

Write the JSON output file listing the determined “best match” for each peaks.

Parameters

- **project** (*Project*)
- **output_dir** (*PathPlus*) – The directory to write the `<project.name>.json` file to.

gunshotmatch_pipeline.nist_ms_search

Configuration for `pyms_nist_search` and NIST MS Search.

Classes:

<code>LazyEngine</code> (<code>config</code> , <code>**kwargs</code>)	Initialize the NIST MS Serch engine on demand.
<code>PyMSNISTSearchCfg</code> (<code>library_path</code> [, <code>user_library</code>])	Configuration for <code>pyms_nist_search</code> .

Functions:

<code>engine_on_demand</code> (<code>config</code> , <code>**kwargs</code>)	Defer initialization of the NIST MS Serch engine until required (if at all).
<code>nist_ms_search_engine</code> (<code>config</code> , <code>**kwargs</code>)	Initialize the NIST MS Serch engine from <code>pyms_nist_search</code> .

class `LazyEngine` (`config`, `**kwargs`)

Bases: `object`

Initialize the NIST MS Serch engine on demand.

Parameters

- **config** (`PyMSNISTSearchCfg`)
- ****kwargs** – Keyword arguments for `pyms_nist_search.win_engine.Engine`

New in version 0.2.0.

Methods:

<code>deinit</code> ()	Cleanup the underlying engine and temporary directory.
------------------------	--

Attributes:

<code>engine</code>	The NIST MS Search engine.
---------------------	----------------------------

deinit ()

Cleanup the underlying engine and temporary directory.

property engine

The NIST MS Search engine.

The engine is created the first time this property is accessed.

Return type `Engine`

class `PyMSNISTSearchCfg` (*library_path*, *user_library=False*)

Bases: `libgunshotmatch.method.MethodBase`

Configuration for `pyms_nist_search`.

Parameters

- **library_path** (`str`) – Absolute path to the NIST library (mainlib or user).
- **user_library** (`bool`) – `True` for user libraries; `False` for the NIST mainlib. Default `False`.

Attributes:

<code>library_path</code>	Absolute path to the NIST library (mainlib or user).
<code>user_library</code>	<code>True</code> for user libraries; <code>False</code> for the NIST mainlib.

library_path

Type: `str`

Absolute path to the NIST library (mainlib or user).

user_library

Type: `bool`

`True` for user libraries; `False` for the NIST mainlib.

engine_on_demand (*config*, ***kwargs*)

Defer initialization of the NIST MS Serch engine until required (if at all).

Parameters

- **config** (`PyMSNISTSearchCfg`)
- ****kwargs** – Keyword arguments for `pyms_nist_search.win_engine.Engine`

Return type `Iterator[LazyEngine]`

New in version 0.2.0.

nist_ms_search_engine (*config*, ***kwargs*)

Initialize the NIST MS Serch engine from `pyms_nist_search`.

Parameters

- **config** (`PyMSNISTSearchCfg`)
- ****kwargs** – Keyword arguments for `pyms_nist_search.win_engine.Engine`

Return type `Iterator[Engine]`

gunshotmatch_pipeline.peaks

Peak detection and alignment functions.

Functions:

<code>align_and_filter_peaks</code> (project, method)	Perform peak alignment and peak filtering for the project, with the given method.
<code>prepare_peak_list</code> (datafile, gcms_data, method)	Construct and filter the peak list.

align_and_filter_peaks (*project, method*)

Perform peak alignment and peak filtering for the project, with the given method.

Parameters

- **project** (`Project`)
- **method** (`Method`)

Return type `DataFrame`

prepare_peak_list (*datafile, gcms_data, method*)

Construct and filter the peak list.

Parameters

- **datafile** (`Datafile`)
- **gcms_data** (`GCMS_data`)
- **method** (`Method`)

Return type `PeakList`

gunshotmatch_pipeline.projects

Metadata for project pipelines.

Classes:

<i>GlobalSettings</i> ([output_directory, method, ...])	Settings applied for all projects.
<i>LoaderMixin</i> ()	Mixin class providing <code>load_method()</code> and <code>load_config()</code> methods.
<i>ProjectSettings</i> (name, datafiles[, method, ...])	Settings for a specific project.
<i>Projects</i> ([global_settings, per_project_settings])	Reference data projects to process through the pipeline.

Functions:

<i>process_projects</i> (projects, output_dir[, ...])	Process projects with common methods and config.
---	--

class GlobalSettings (output_directory='output', method=None, config=None, data_directory=None)

Bases: `libgunshotmatch.method.MethodBase`, `gunshotmatch_pipeline.projects.LoaderMixin`

Settings applied for all projects.

Parameters

- **output_directory** (`str`) – Relative or absolute path to the directory the output files should be placed in. Default 'output'.
- **method** (`Optional[str]`) – Relative or absolute filename to the method TOML file. The table name is “method”. Default `None`.
- **config** (`Optional[str]`) – Relative or absolute filename to the configuration TOML file. The table name is “config”. Default `None`.
- **data_directory** (`Optional[str]`) – Relative or absolute path to the directory containing the data files. Default `None`.

The method and config files may point to the same TOML file.

Attributes:

<i>config</i>	Relative or absolute filename to the configuration TOML file.
<i>data_directory</i>	Relative or absolute path to the directory containing the data files.
<i>method</i>	Relative or absolute filename to the method TOML file.
<i>output_directory</i>	Relative or absolute path to the directory the output files should be placed in.

config

Type: `Optional[str]`

Relative or absolute filename to the configuration TOML file. The table name is “gunshotmatch”.

data_directory

Type: `Optional[str]`

Relative or absolute path to the directory containing the data files.

method

Type: `Optional[str]`

Relative or absolute filename to the method TOML file. The table name is “method”.

output_directory

Type: `str`

Relative or absolute path to the directory the output files should be placed in.

class LoaderMixin

Bases: `object`

Mixin class providing `load_method()` and `load_config()` methods.

Methods:

<code>load_config()</code>	Load the configuration for this project from the specified file.
<code>load_method()</code>	Load the method for this project from the specified file.

load_config()

Load the configuration for this project from the specified file.

Return type `Configuration`

load_method()

Load the method for this project from the specified file.

Return type `Method`

class ProjectSettings (*name, datafiles, method=None, config=None, data_directory=None*)

Bases: `libgunshotmatch.method.MethodBase`, `gunshotmatch_pipeline.projects.LoaderMixin`

Settings for a specific project.

Parameters

- **name** (`str`) – The project name.
- **datafiles** (`List[str]`) – List of input datafiles (paths relative to the `data_directory` option)
- **method** (`Optional[str]`) – Relative or absolute filename to the method TOML file. The table name is “method”. Default `None`.
- **config** (`Optional[str]`) – Relative or absolute filename to the configuration TOML file. The table name is “config”. Default `None`.
- **data_directory** (`Optional[str]`) – Relative or absolute path to the directory containing the data files. Default `None`.

Attributes:

<i>config</i>	Relative or absolute filename to the configuration TOML file.
<i>data_directory</i>	Relative or absolute path to the directory containing the data files.
<i>datafiles</i>	List of input datafiles (paths relative to the <i>data_directory</i> option)
<i>method</i>	Relative or absolute filename to the method TOML file.
<i>name</i>	The project name.

Methods:

<i>get_datafile_paths()</i>	Returns an iterator over paths to the datafiles.
-----------------------------	--

config**Type:** `Optional[str]`

Relative or absolute filename to the configuration TOML file. The table name is “config”.

data_directory**Type:** `Optional[str]`

Relative or absolute path to the directory containing the data files.

datafiles**Type:** `List[str]`List of input datafiles (paths relative to the *data_directory* option)**get_datafile_paths()**

Returns an iterator over paths to the datafiles.

The paths start with *data_directory* if set.**Return type** `Iterator[PathPlus]`**method****Type:** `Optional[str]`

Relative or absolute filename to the method TOML file. The table name is “method”.

name**Type:** `str`

The project name.

```
class Projects (global_settings=GlobalSettings(output_directory='output', method=None,  
               config=None, data_directory=None), per_project_settings={})
```

Bases: `libgunshotmatch.method.MethodBase`

Reference data projects to process through the pipeline.

Parameters

- **global_settings** (*GlobalSettings*) – Settings applied for all projects. Default `GlobalSettings(output_directory='output', method=None, config=None, data_directory=None)`.
- **per_project_settings** (*Dict[str, ProjectSettings]*) – Settings for specific projects. Default `{}`.

Methods:

<code>from_json(json_string)</code>	Parse a <i>Projects</i> from a JSON string.
<code>from_toml(toml_string)</code>	Parse a <i>Projects</i> from a TOML string.
<code>get_project_settings(project_name)</code>	Returns the settings for the given project, taking into account the global settings.
<code>has_common_config()</code>	Returns whether all projects have common configuration.
<code>has_common_method()</code>	Returns whether all projects have a common method.
<code>iter_loaded_projects()</code>	Iterate <i>Project</i> objects loaded from disk.
<code>iter_project_settings()</code>	Iterate over the per-project settings, taking into account the global settings.
<code>load_project(project_name)</code>	Load a previously created project.
<code>to_toml()</code>	Convert a <i>Configuration</i> to a TOML string.

Attributes:

<code>global_settings</code>	Settings applied for all projects.
<code>per_project_settings</code>	Settings for specific projects.

```
classmethod from_json (json_string)  
    Parse a Projects from a JSON string.
```

Parameters **json_string** (*str*)

Return type *Projects*

```
classmethod from_toml (toml_string)  
    Parse a Projects from a TOML string.
```

Parameters **toml_string** (*str*)

Return type *Projects*

```
get_project_settings (project_name)  
    Returns the settings for the given project, taking into account the global settings.
```

Parameters **project_name** (*str*)

Return type *ProjectSettings*

global_settings**Type:** `GlobalSettings`

Settings applied for all projects.

has_common_config()

Returns whether all projects have common configuration.

Return type `bool`**has_common_method()**

Returns whether all projects have a common method.

Return type `bool`**iter_loaded_projects()**Iterate `Project` objects loaded from disk.**Return type** `Iterator[Project]`**iter_project_settings()**

Iterate over the per-project settings, taking into account the global settings.

Return type `Iterator[ProjectSettings]`**load_project(project_name)**

Load a previously created project.

Parameters `project_name` (`str`)**Return type** `Project`**per_project_settings****Type:** `Dict[str, ProjectSettings]`

Settings for specific projects.

to_toml()Convert a `Configuration` to a TOML string.**Return type** `str`**process_projects(projects, output_dir, recreate=False)**

Process projects with common methods and config.

Parameters

- **projects** (`Projects`)
- **output_dir** (`Union[str, Path, PathLike]`)
- **recreate** (`bool`) – Force regeneration of `.gsmr` and `.gsmp` files. Default `False`.

Return type `Iterator[Project]`

gunshotmatch_pipeline.results

Results presented in different formats.

Classes:

<i>Matches</i>	Return type from <i>matches()</i> .
<i>MatchesCompounds</i>	Type hint for the <i>compounds</i> key in <i>Matches</i> .
<i>MatchesMetadata</i>	Type hint for the <i>metadata</i> key in <i>Matches</i> .

Functions:

<i>compounds</i> (*project[, normalize])	Returns data on the compounds in each repeat in the project(s).
<i>compounds_from_matches</i> (*matches_data[, ...])	Prepares data on the compounds in each repeat from the output of <i>matches()</i> for each project.
<i>machine_learning_data</i> (*project[, normalize])	Returns data formatted for training a decision tree or other machine learning model.
<i>matches</i> (project)	Returns data on the “best match” for each peak.
<i>unknown</i> (unknown_project[, normalize])	Returns results for an unknown sample.
<i>unknown_machine_learning_data</i> (unknown_project)	Returns data formatted for training a decision tree or other machine learning model.

typeddict Matches

Bases: `TypedDict`

Return type from *matches()*.

Required Keys

- **metadata** (*MatchesMetadata*)
- **compounds** (`Dict[str, MatchesCompounds]`)

typeddict MatchesCompounds

Bases: `TypedDict`

Type hint for the *compounds* key in *Matches*.

Required Keys

- **Mean Retention Time** (`float`)
- **Mean Peak Area** (`float`)
- **CAS** (`str`)
- **Retention Times** (`List[float]`)
- **Peak Areas** (`List[float]`)

- **Hit Numbers** (`List[int]`)
- **Match Factors** (`List[int]`)
- **Reverse Match Factors** (`List[int]`)

typeddict MatchesMetadata

Bases: `TypedDict`

Type hint for the metadata key in *Matches*.

Required Keys

- **project** (`str`)
- **original_filenames** (`List[str]`)
- **created** (`str`)

compounds (**project, normalize=False*)

Returns data on the compounds in each repeat in the project(s).

The output mapping gives the peak areas for each compound in the different projects, grouped by compound.

Parameters

- ***project** (`Project`)
- **normalize** (`bool`) – Default `False`.

Return type `Dict[str, Dict[str, List[float]]]`

compounds_from_matches (**matches_data, normalize=False*)

Prepares data on the compounds in each repeat from the output of *matches()* for each project.

The output mapping gives the peak areas for each compound in the different projects, grouped by compound.

Parameters

- ***matches_data** (*Matches*)
- **normalize** (`bool`) – Default `False`.

Return type `Dict[str, Dict[str, List[float]]]`

machine_learning_data (**project, normalize=False*)

Returns data formatted for training a decision tree or other machine learning model.

Parameters

- ***project** (`Project`)
- **normalize** (`bool`) – Default `False`.

Return type `Dict[str, Dict[str, float]]`

matches (*project*)

Returns data on the “best match” for each peak.

Parameters **project** (`Project`)

Return type *Matches*

unknown (*unknown_project*, *normalize=False*)

Returns results for an unknown sample.

The output mapping is formatted the same as that from *compounds()*, but with only one “project”.

Parameters

- **unknown_project** (*Project*)
- **normalize** (*bool*) – Default *False*.

Return type *Dict[str, Dict[str, List[float]]]*

unknown_machine_learning_data (*unknown_project*, *normalize=False*)

Returns data formatted for training a decision tree or other machine learning model.

Parameters

- **unknown_project** (*Project*)
- **normalize** (*bool*) – Default *False*.

Return type *Dict[str, Dict[str, float]]*

gunshotmatch_pipeline.unknowns

Metadata and pipeline for unknown samples.

Classes:

<code>UnknownSettings(name, datafile, method, ...)</code>	Settings for an unknown propellant or OGSR sample.
---	--

Functions:

<code>filter_and_identify_peaks(repeat, method, engine)</code>	Filter peaks by minimum peak area, then identify compounds.
<code>process_unknown(unknown, output_dir[, recreate])</code>	Process an “unknown” sample.

class UnknownSettings (*name, datafile, method, config, output_directory, data_directory=""*)

Bases: `libgunshotmatch.method.MethodBase`, `gunshotmatch_pipeline.projects.LoaderMixin`

Settings for an unknown propellant or OGSR sample.

Parameters

- **name** (*str*) – The unknown sample’s name or identifier.
- **datafile** (*str*) – The input datafile
- **method** (*str*) – Relative or absolute filename to the method TOML file. The table name is “method”.
- **config** (*str*) – Relative or absolute filename to the configuration TOML file. The table name is “config”.
- **output_directory** (*str*) – Relative or absolute path to the directory the output files should be placed in.
- **data_directory** (*str*) – Relative or absolute path to the directory containing the data files. Default ‘’.

Attributes:

<code>config</code>	Relative or absolute filename to the configuration TOML file.
<code>data_directory</code>	Relative or absolute path to the directory containing the data files.
<code>datafile</code>	The input datafile
<code>datafile_path</code>	The absolute path to the datafile.
<code>method</code>	Relative or absolute filename to the method TOML file.
<code>name</code>	The unknown sample’s name or identifier.
<code>output_directory</code>	Relative or absolute path to the directory the output files should be placed in.

Methods:

<code>from_json(json_string)</code>	Parse an <i>UnknownSettings</i> from a JSON string.
<code>from_toml(toml_string)</code>	Parse an <i>UnknownSettings</i> from a TOML string.
<code>to_toml()</code>	Convert an <i>UnknownSettings</i> to a TOML string.

config**Type:** `str`

Relative or absolute filename to the configuration TOML file. The table name is “config”.

data_directory**Type:** `str`

Relative or absolute path to the directory containing the data files.

datafile**Type:** `str`

The input datafile

property datafile_path

The absolute path to the datafile.

Return type `PathPlus`**classmethod from_json** (*json_string*)Parse an *UnknownSettings* from a JSON string.**Parameters** `json_string` (`str`)**Return type** *UnknownSettings***classmethod from_toml** (*toml_string*)Parse an *UnknownSettings* from a TOML string.**Parameters** `toml_string` (`str`)**Return type** *UnknownSettings***method****Type:** `str`

Relative or absolute filename to the method TOML file. The table name is “method”.

name**Type:** `str`

The unknown sample’s name or identifier.

output_directory**Type:** `str`

Relative or absolute path to the directory the output files should be placed in.

to_toml()

Convert an *UnknownSettings* to a TOML string.

Return type `str`

filter_and_identify_peaks (*repeat, method, engine*)

Filter peaks by minimum peak area, then identify compounds.

Parameters

- **repeat** (*Repeat*)
- **method** (*Method*)
- **engine** (*Engine*) – NIST MS Search engine.

process_unknown (*unknown, output_dir, recreate=False*)

Process an “unknown” sample.

Parameters

- **unknown** (*UnknownSettings*)
- **output_dir** (*Union[str, Path, PathLike]*)
- **recreate** (*bool*) – Force regeneration of `.gsmr` and `.gsmp` files. Default `False`.

Return type `Project`

gunshotmatch_pipeline.utils

General utility functions.

Classes:

<i>NameMapping</i>	Class for mapping IUPAC preferred names to more common, friendlier names.
--------------------	---

Data:

<i>friendly_name_mapping</i>	Mapping of IUPAC preferred names to more common, friendlier names.
------------------------------	--

Functions:

<i>project_plural(n)</i>
<i>unknown_plural(n)</i>

```
project_plural(*args, **kwargs) = Plural('project', 'projects')  
    domdf_python_tools.words.Plural for project.
```

```
unknown_plural(*args, **kwargs) = Plural('unknown', 'unknowns')  
    domdf_python_tools.words.Plural for unknown.
```

New in version 0.9.0.

friendly_name_mapping

Type: *NameMapping*

Mapping of IUPAC preferred names to more common, friendlier names.

class NameMapping

Bases: *Dict[str, str]*

Class for mapping IUPAC preferred names to more common, friendlier names.

On lookup, if the name has no known alias the looked-up name is returned.

New in version 0.4.0.

The module also provides either `tomli` or `tomllib` (depending on Python version) through the `tomllib` attribute.

Part II

Contributing

Contributing

gunshotmatch-pipeline uses `tox` to automate testing and packaging, and `pre-commit` to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

12.1 Coding style

`formate` is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

12.2 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

12.3 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

12.4 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

Downloading source code

The `gunshotmatch-pipeline` source code is available on GitHub, and can be accessed from the following URL:
<https://github.com/GunShotMatch/gunshotmatch-pipeline>

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/GunShotMatch/gunshotmatch-pipeline
```

```
Cloning into 'gunshotmatch-pipeline'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a ‘zip’ file by clicking:

Clone or download → Download Zip

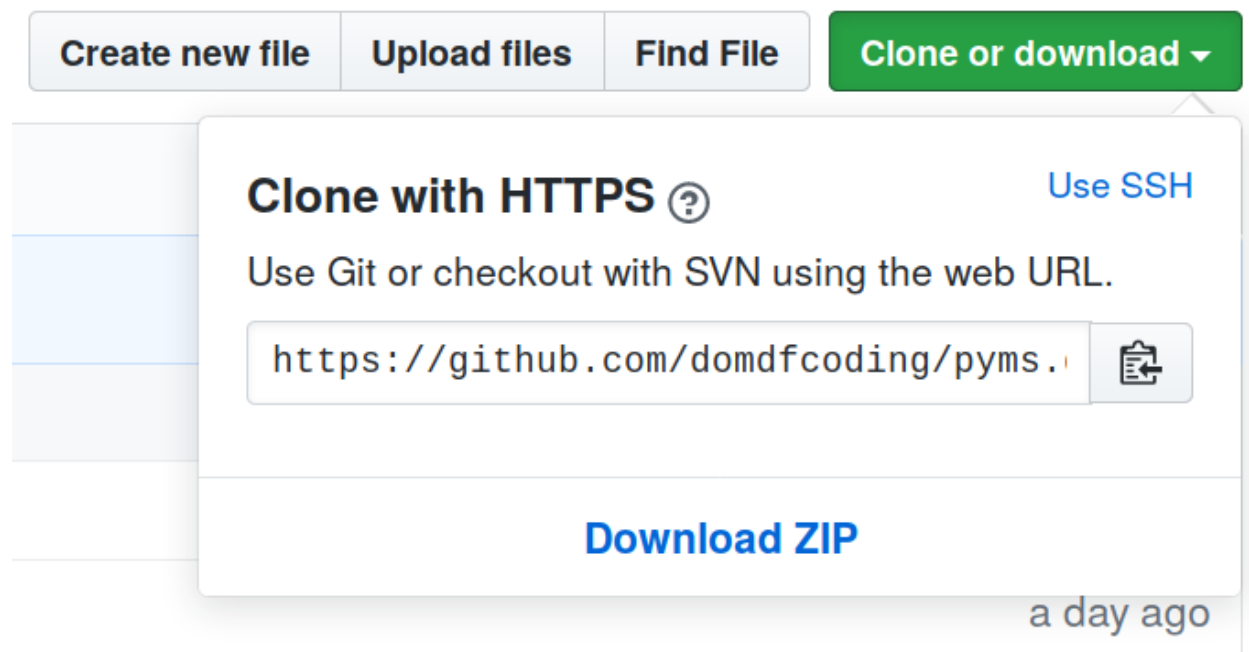


Fig. 1: Downloading a ‘zip’ file of the source code

13.1 Building from source

The recommended way to build `gunshotmatch-pipeline` is to use `tox`:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use `pep517.build` or another **PEP 517**-compatible build tool.

License

gunshotmatch-pipeline is licensed under the [MIT License](#)

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

[See more information on choosealicense.com](#) ⇒

```
Copyright (c) 2023 Dominic Davis-Foster
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.
```


Python Module Index

g

- `gunshotmatch_pipeline`, [5](#)
- `gunshotmatch_pipeline.config`, [7](#)
- `gunshotmatch_pipeline.decision_tree`, [9](#)
- `gunshotmatch_pipeline.decision_tree.export`,
[14](#)
- `gunshotmatch_pipeline.decision_tree.predictions`,
[16](#)
- `gunshotmatch_pipeline.exporters`, [17](#)
- `gunshotmatch_pipeline.nist_ms_search`,
[19](#)
- `gunshotmatch_pipeline.peaks`, [21](#)
- `gunshotmatch_pipeline.projects`, [23](#)
- `gunshotmatch_pipeline.results`, [29](#)
- `gunshotmatch_pipeline.unknowns`, [33](#)
- `gunshotmatch_pipeline.utils`, [37](#)

Symbols

`__eq__()` (*DecisionTreeVisualiser* method), 10
`__getstate__()` (*DecisionTreeVisualiser* method), 10
`__ne__()` (*DecisionTreeVisualiser* method), 10
`__repr__()` (*DecisionTreeVisualiser* method), 10
`__repr__()` (*PredictionResult* method), 16
`__setattr__()` (*DecisionTreeVisualiser* method), 10
`__setstate__()` (*DecisionTreeVisualiser* method), 10

A

`align_and_filter_peaks()` (in module *gunshotmatch_pipeline.peaks*), 21

C

`class_name` (*namedtuple* field)
 PredictionResult (*namedtuple* in *gunshotmatch_pipeline.decision_tree.predictions*), 16
`classifier` (*DecisionTreeVisualiser* attribute), 10
`compounds()` (in module *gunshotmatch_pipeline.results*), 30
`compounds_from_matches()` (in module *gunshotmatch_pipeline.results*), 30
`config` (*GlobalSettings* attribute), 23
`config` (*ProjectSettings* attribute), 25
`config` (*UnknownSettings* attribute), 34
`Configuration` (class in *gunshotmatch_pipeline.config*), 7
`correct()` (*PredictionResult* property), 16

D

`data_directory` (*GlobalSettings* attribute), 24
`data_directory` (*ProjectSettings* attribute), 25
`data_directory` (*UnknownSettings* attribute), 34
`data_from_projects()` (in module *gunshotmatch_pipeline.decision_tree*), 11
`data_from_unknown()` (in module *gunshotmatch_pipeline.decision_tree*), 11
`datafile` (*UnknownSettings* attribute), 34
`datafile_path()` (*UnknownSettings* property), 34
`datafiles` (*ProjectSettings* attribute), 25

`DecisionTreeVisualiser` (class in *gunshotmatch_pipeline.decision_tree*), 9
`deinit()` (*LazyEngine* method), 19
`deserialise_decision_tree()` (in module *gunshotmatch_pipeline.decision_tree.export*), 14
`deserialise_random_forest()` (in module *gunshotmatch_pipeline.decision_tree.export*), 15
`dotsafe_name()` (in module *gunshotmatch_pipeline.decision_tree*), 11
`dump_predictions()` (in module *gunshotmatch_pipeline.decision_tree.predictions*), 16

E

`engine()` (*LazyEngine* property), 19
`engine_on_demand()` (in module *gunshotmatch_pipeline.nist_ms_search*), 20

F

`factorize_map` (*DecisionTreeVisualiser* attribute), 10
`feature_names` (*DecisionTreeVisualiser* attribute), 10
`filter_and_identify_peaks()` (in module *gunshotmatch_pipeline.unknowns*), 35
`fit_decision_tree()` (in module *gunshotmatch_pipeline.decision_tree*), 12
`friendly_name_mapping` (in module *gunshotmatch_pipeline.utils*), 37
`from_data()` (*DecisionTreeVisualiser* class method), 11
`from_json()` (*Configuration* class method), 7
`from_json()` (*Projects* class method), 26
`from_json()` (*UnknownSettings* class method), 34
`from_toml()` (*Configuration* class method), 7
`from_toml()` (*Projects* class method), 26
`from_toml()` (*UnknownSettings* class method), 34

G

`get_datafile_paths()` (*ProjectSettings* method), 25

`get_feature_names()` (in module *gunshotmatch_pipeline.decision_tree*), 12
`get_project_settings()` (*Projects method*), 26
`global_settings` (*Projects attribute*), 27
`GlobalSettings` (class in *gunshotmatch_pipeline.projects*), 23
`gunshotmatch_pipeline`
 module, 5
`gunshotmatch_pipeline.config`
 module, 7
`gunshotmatch_pipeline.decision_tree`
 module, 9
`gunshotmatch_pipeline.decision_tree.export`
 module, 14
`gunshotmatch_pipeline.decision_tree.predictions`
 module, 16
`gunshotmatch_pipeline.exporters`
 module, 17
`gunshotmatch_pipeline.nist_ms_search`
 module, 19
`gunshotmatch_pipeline.peaks`
 module, 21
`gunshotmatch_pipeline.projects`
 module, 23
`gunshotmatch_pipeline.results`
 module, 29
`gunshotmatch_pipeline.unknowns`
 module, 33
`gunshotmatch_pipeline.utils`
 module, 37

H

`has_common_config()` (*Projects method*), 27
`has_common_method()` (*Projects method*), 27

I

`iter_loaded_projects()` (*Projects method*), 27
`iter_project_settings()` (*Projects method*), 27

L

`LazyEngine` (class in *gunshotmatch_pipeline.nist_ms_search*), 19
`library_path` (*PyMSNISTSearchCfg attribute*), 20
`load_config()` (*LoaderMixin method*), 24
`load_method()` (*LoaderMixin method*), 24
`load_predictions()` (in module *gunshotmatch_pipeline.decision_tree.predictions*), 16
`load_project()` (*Projects method*), 27
`LoaderMixin` (class in *gunshotmatch_pipeline.projects*), 24

M

`machine_learning_data()` (in module *gunshotmatch_pipeline.results*), 30
`Matches` (*typeddict in gunshotmatch_pipeline.results*), 29
`matches()` (in module *gunshotmatch_pipeline.results*), 30
`MatchesCompounds` (*typeddict in gunshotmatch_pipeline.results*), 29
`MatchesMetadata` (*typeddict in gunshotmatch_pipeline.results*), 30
`method` (*GlobalSettings attribute*), 24
`method` (*ProjectSettings attribute*), 25
`method` (*UnknownSettings attribute*), 34
`MIT License`, 45
module
 gunshotmatch_pipeline, 5
 gunshotmatch_pipeline.config, 7
 gunshotmatch_pipeline.decision_tree, 9
 gunshotmatch_pipeline.decision_tree.export, 14
 gunshotmatch_pipeline.decision_tree.predictions, 16
 gunshotmatch_pipeline.exporters, 17
 gunshotmatch_pipeline.nist_ms_search, 19
 gunshotmatch_pipeline.peaks, 21
 gunshotmatch_pipeline.projects, 23
 gunshotmatch_pipeline.results, 29
 gunshotmatch_pipeline.unknowns, 33
 gunshotmatch_pipeline.utils, 37

N

`name` (*namedtuple field*)
 PredictionResult (*namedtuple in gunshotmatch_pipeline.decision_tree.predictions*), 16
`name` (*ProjectSettings attribute*), 25
`name` (*UnknownSettings attribute*), 34
`NameMapping` (class in *gunshotmatch_pipeline.utils*), 37
`nist_ms_search_engine()` (in module *gunshotmatch_pipeline.nist_ms_search*), 20

O

`output_directory` (*GlobalSettings attribute*), 24
`output_directory` (*UnknownSettings attribute*), 34

P

`per_project_settings` (*Projects attribute*), 27
`predict_unknown()` (in module *gunshotmatch_pipeline.decision_tree*), 12

PredictionResult (*namedtuple in gunshotmatch_pipeline.decision_tree.predictions*), 16

 class_name (*namedtuple field*), 16

 name (*namedtuple field*), 16

 predictions (*namedtuple field*), 16

predictions (*namedtuple field*)

 PredictionResult (*namedtuple in gunshotmatch_pipeline.decision_tree.predictions*), 16

prepare_datafile() (*in module gunshotmatch_pipeline*), 5

prepare_peak_list() (*in module gunshotmatch_pipeline.peaks*), 21

process_projects() (*in module gunshotmatch_pipeline.projects*), 27

process_unknown() (*in module gunshotmatch_pipeline.unknowns*), 35

project_from_repeats() (*in module gunshotmatch_pipeline*), 5

project_plural (*in module gunshotmatch_pipeline.utils*), 37

Projects (*class in gunshotmatch_pipeline.projects*), 26

ProjectSettings (*class in gunshotmatch_pipeline.projects*), 24

pymms_nist_search (*Configuration attribute*), 7

PyMSNISTSearchCfg (*class in gunshotmatch_pipeline.nist_ms_search*), 20

Python Enhancement Proposals

 PEP 517, 44

S

serialise_decision_tree() (*in module gunshotmatch_pipeline.decision_tree.export*), 14

serialise_random_forest() (*in module gunshotmatch_pipeline.decision_tree.export*), 15

simulate_data() (*in module gunshotmatch_pipeline.decision_tree*), 12

T

to_toml() (*Configuration method*), 7

to_toml() (*Projects method*), 27

to_toml() (*UnknownSettings method*), 34

U

unknown() (*in module gunshotmatch_pipeline.results*), 31

unknown_machine_learning_data() (*in module gunshotmatch_pipeline.results*), 31

unknown_plural (*in module gunshotmatch_pipeline.utils*), 37

UnknownSettings (*class in gunshotmatch_pipeline.unknowns*), 33

user_library (*PyMSNISTSearchCfg attribute*), 20

V

verify_saved_datafile() (*in module gunshotmatch_pipeline.exporters*), 17

verify_saved_decision_tree() (*in module gunshotmatch_pipeline.decision_tree.export*), 14

verify_saved_project() (*in module gunshotmatch_pipeline.exporters*), 17

verify_saved_random_forest() (*in module gunshotmatch_pipeline.decision_tree.export*), 15

visualise_decision_tree() (*in module gunshotmatch_pipeline.decision_tree*), 13

visualise_tree() (*DecisionTreeVisualiser method*), 11

W

write_combined_csv() (*in module gunshotmatch_pipeline.exporters*), 17

write_matches_json() (*in module gunshotmatch_pipeline.exporters*), 18