gunshotmatch-pipeline

Release 0.9.1

GunShotMatch Analysis Pipeline

Dominic Davis-Foster

May 10, 2024

Contents

Chapter

ONE

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

Chapter

TWO

gunshotmatch_pipeline

GunShotMatch Pipeline.

Functions:

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

Chapter

THREE

gunshotmatch_pipeline.config

Configuration for GunShotMatch analysis.

class Configuration(*pyms_nist_search*)

Bases: MethodBase

Overall GunShotMatch configuration.

Methods:

from_json(json_string)	Parse a *Configuration* from a JSON string.
from_toml(toml_string)	Parse a *Configuration* from a TOML string.
to_toml()	Convert a *Configuration* to a TOML string.

Attributes:

pyms_nist_search	Configuration for pyms_nist_search.

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

Chapter

FOUR

gunshotmatch_pipeline.decision_tree

Prepare data and train decision trees.

Classes:

DecisionTreeVisualiser(classifier, ...)	Class for exporting visualisations of a decision tree or random forest.

Functions:

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

__eq__(other)	Return `self == other`.
__getstate__()	Used for pickling.
__ne__(other)	Return `self != other`.
__repr__()	Return a string representation of the *DecisionTreeVisualiser*.

continues on next page

Table 3 – continued from previous page

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

Attributes:

classifier	Decision tree or random forest classifier.
factorize_map	List of class names in the order they appear as classes in the classifier.
feature_names	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:

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

PredictionResult(name, class_name, predictions)	Represents the predicted classes from a random forest classifier.

Functions:

dump_predictions(predictions[, indent])	Return a JSON representation of the predictions.
load_predictions(predictions_json)	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*]

Chapter

FIVE

gunshotmatch_pipeline.exporters

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

Functions:

verify_saved_datafile(in_process, from_file)	Verify the data in a saved Datafile matches the data in memory.
verify_saved_project(in_process, from_file)	Verify the data in a saved Project matches the data in memory.
write_combined_csv(repeat, output_dir)	Write a CSV file listing the top hits for each peak in the Repeat, with associated data.
write_matches_json(project, output_dir)	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.

Chapter

SIX

gunshotmatch_pipeline.nist_ms_search

Configuration for pyms_nist_search and NIST MS Search.

Classes:

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

Functions:

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

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:

deinit()	Cleanup the underlying engine and temporary directory.

Attributes:

engine	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:

library_path	Absolute path to the NIST library (mainlib or user).
user_library	True for user libraries; False 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]

Chapter

SEVEN

gunshotmatch_pipeline.peaks

Peak detection and alignment functions.

Functions:

align_and_filter_peaks(project, method)	Perform peak alignment and peak filtering for the project, with the given method.
prepare_peak_list(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

Chapter

EIGHT

gunshotmatch_pipeline.projects

Metadata for project pipelines.

Classes:

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

Functions:

process_projects(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:

config	Relative or absolute filename to the configuration TOML file.
data_directory	Relative or absolute path to the directory containing the data files.
method	Relative or absolute filename to the method TOML file.
output_directory	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:

load_config()	Load the configuration for this project from the specified file.
load_method()	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:

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

Methods:

get_datafile_paths()	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:

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

Attributes:

global_settings	Settings applied for all projects.
per_project_settings	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]

Chapter

NINE

gunshotmatch_pipeline.results

Results presented in different formats.

Classes:

Matches	Return type from *matches()*.
MatchesCompounds	Type hint for the compounds key in *Matches*.
MatchesMetadata	Type hint for the metadata key in *Matches*.

Functions:

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

Chapter

TEN

gunshotmatch_pipeline.unknowns

Metadata and pipeline for unknown samples.

Classes:

UnknownSettings(name, datafile, method, ...)	Settings for an unknown propellant or OGSR sample.

Functions:

filter_and_identify_peaks(repeat, method, engine)	Filter peaks by minimum peak area, then identify compounds.
process_unknown(unknown, output_dir[, recreate])	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:

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

Methods:

from_json(json_string)	Parse an *UnknownSettings* from a JSON string.
from_toml(toml_string)	Parse an *UnknownSettings* from a TOML string.
to_toml()	Convert an *UnknownSettings* 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

Chapter

ELEVEN

gunshotmatch_pipeline.utils

General utility functions.

Classes:

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

Data:

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

Functions:

project_plural(n)	
unknown_plural(n)	

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

Chapter

TWELVE

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
```

Chapter

THIRTEEN

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

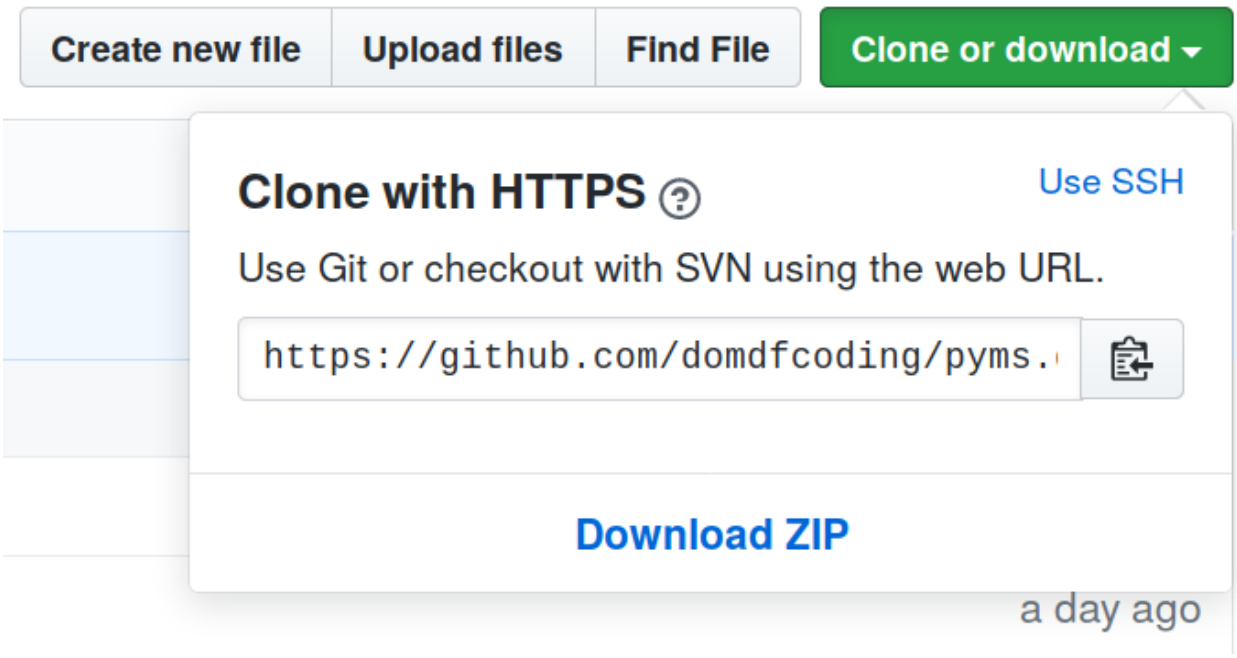

Fig. 1: Downloading a 'zip' file of the source code

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.

Chapter

FOURTEEN

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 ⇒

Python Module Index

g

Index

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

S

T

U

V

W