
TrajectoryNet

Release 0.1

Bence Gabor Peter

Nov 07, 2023

CONTENTS:

1	Usage	3
1.1	Installation	3
1.2	Install Anaconda environment manager	3
1.3	Prerequisites	3
1.4	Installation	4
1.5	Create a new conda environment	5
2	Usage	7
2.1	Create database from video files	7
2.2	Run clustering on database	8
3	trajectorynet	9
3.1	DetectionPipeline module	9
3.2	classification module	16
3.3	classifier module	28
3.4	clustering module	34
3.5	converter module	46
3.6	dataManagementClasses module	46
3.7	database_metadata module	59
3.8	detect module	59
3.9	evaluate module	59
3.10	examine_tracks module	59
3.11	masker module	59
3.12	traffic_statistics module	60
3.13	train module	62
3.14	utility package	62
3.14.1	Submodules	62
3.14.2	utility.databaseLoader module	62
3.14.3	utility.databaseLogger module	63
3.14.4	utility.dataset module	67
3.14.5	utility.featurevector module	69
3.14.6	utility.general module	71
3.14.7	utility.logging module	72
3.14.8	utility.models module	73
3.14.9	utility.plots module	73
3.14.10	utility.preprocessing module	75
3.14.11	utility.training module	77
3.14.12	Module contents	77
4	Indices and tables	79

Python Module Index	81
Index	83

TrajectoryNet is a Python package for trajectory prediction.

1.1 Installation

1.2 Install Anaconda environment manager

1.3 Prerequisites

To use GUI packages with Linux, you will need to install the following extended dependencies for Qt:

Debian

```
apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1  
↳ libxcomposite1 libasound2 libxi6 libxtst6
```

RedHat

```
yum install libXcomposite libXcursor libXi libXtst libXrandr alsa-lib mesa-libEGL  
↳ libXdamage mesa-libGL libXScrnSaver
```

ArchLinux

```
pacman -Sy libxau libxi libxss libxtst libxcursor libxcomposite libxdamage libxfixes  
↳ libxrandr libxrender mesa-libgl alsa-lib libglvnd
```

OpenSuse/SLES

```
zypper install libXcomposite1 libXi6 libXext6 libXau6 libX11-6 libXrandr2 libXrender1  
↳ libXss1 libXtst6 libXdamage1 libXcursor1 libxcb1 libasound2 libX11-xcb1 Mesa-libGL1  
↳ Mesa-libEGL1
```

Gentoo

```
emerge x11-libs/libXau x11-libs/libxcb x11-libs/libX11 x11-libs/libXext x11-libs/  
↳ libXfixes x11-libs/libXrender x11-libs/libXi x11-libs/libXcomposite x11-libs/libXrandr  
↳ x11-libs/libXcursor x11-libs/libXdamage x11-libs/libXScrnSaver x11-libs/libXtst media-  
↳ libs/alsa-lib media-libs/mesa
```

1.4 Installation

For x86 systems.

In your browser, download the Anaconda installer for Linux.

Search for “terminal” in your applications and click to open.

(Recommended) Verify the installer’s data integrity with SHA-256. For more information on hash verification, see cryptographic hash validation.

In the terminal, run the following:

```
shasum -a 256 /PATH/FILENAME
# Replace /PATH/FILENAME with your installation's path and filename.
```

Install for Python 3.7 or 2.7 in the terminal:

For Python 3.7, enter the following:

```
# Include the bash command regardless of whether or not you are using the Bash shell
bash ~/Downloads/Anaconda3-2020.05-Linux-x86_64.sh
# Replace ~/Downloads with your actual path
# Replace the .sh file name with the name of the file you downloaded
```

For Python 2.7, enter the following:

```
# Include the bash command regardless of whether or not you are using the Bash shell
bash ~/Downloads/Anaconda2-2019.10-MacOSX-x86_64.sh
# Replace ~/Downloads with your actual path
# Replace the .sh file name with the name of the file you downloaded
```

Press Enter to review the license agreement. Then press and hold Enter to scroll.

Enter “yes” to agree to the license agreement.

Use Enter to accept the default install location, use CTRL+C to cancel the installation, or enter another file path to specify an alternate installation directory. If you accept the default install location, the installer displays PREFIX=/home/<USER>/anaconda<2/3> and continues the installation. It may take a few minutes to complete.

Note

Anaconda recommends you accept the default install location. Do not choose the path as /usr for the Anaconda/Miniconda installation.

Anaconda recommends you enter “yes” to initialize Anaconda Distribution by running conda init.

If you enter “no”, then conda will not modify your shell scripts at all. In order to initialize conda after the installation process is done, run the following commands:

```
# Replace <PATH_TO_CONDA> with the path to your conda install
source <PATH_TO_CONDA>/bin/activate
conda init
```

For more information, see the FAQ.

The installer finishes and displays, “Thank you for installing Anaconda<2/3>!”

Close and re-open your terminal window for the installation to take effect, or enter the command source ~/.bashrc to refresh the terminal.

You can also control whether or not your shell has the base environment activated each time it opens.

```
# The base environment is activated by default
conda config --set auto_activate_base True

# The base environment is not activated by default
conda config --set auto_activate_base False

# The above commands only work if conda init has been run first
# conda init is available in conda versions 4.6.12 and later
```

Verify your installation.

Note

If you install multiple versions of Anaconda, the system defaults to the most current version, as long as you haven't altered the default install path.

To use this package clone the repository and install the dependencies.

1.5 Create a new conda environment

```
$ conda create -n <env_name> python=3.7
```

Activate the environment

```
$ conda activate <env_name>
```

Create a new conda environment from an environment.yml file

```
$ git clone https://github.com/Pecneb/computer_vision_research.git

$ conda env create -f environment.yml
```


USAGE

```
$ python3 trajectorynet/detect.py --help
usage: detect.py [-h] --video VIDEO --outdir OUTDIR [--database] [--joblib] --yolo-
↪model YOLO_MODEL [--iou IOU] [--score SCORE] [--device DEVICE] [--half] [--show] [--
↪max-age MAX_AGE] [--model MODEL]

Detect objects in a video.

optional arguments:
-h, --help            show this help message and exit
--video VIDEO          Path to video file.
--outdir OUTDIR        Path to output video file.
--database             Save results to database.
--joblib              Save results to database.
--yolo-model YOLO_MODEL
                        Path to model weights file.
--iou IOU             IoU threshold.
--score SCORE          Score threshold.
--device DEVICE        Device to run inference on.
--half               Use half precision.
--show               View output video.
--max-age MAX_AGE     Max age of a track.
--model MODEL         Path to trajectorynet model.
```

2.1 Create database from video files

With this example command, the program will create a joblib file containing trajectory objects for each video file in the directory `cv_research_video_dataset/Bellevue_116th_NE12th/`. The joblib file will be saved in the directory `research_data/Bellevue_NE116th/`.

```
$ python3 trajectorynet/detect.py --video ../../cv_research_video_dataset/Bellevue_116th_
↪NE12th/Bellevue_116th_NE12th__2017-09-11_12-08-33.mp4 --outdir research_data/Bellevue_
↪NE116th_test/ --joblib --iou 0.5 --device 0 --half --view --score 0.6 --model_
↪trajectorynet/yolov7/yolov7.pt
```

2.2 Run clustering on database

```
$ python3 trajectorynet/clustering.py -db research_data/Bellevue_NE116th_test/Bellevue_
↳ 116th_NE12th.joblib --outdir research_data/Bellevue_NE116th_test/ --n-jobs 6 --
↳ dimensions 4D optics --min-samples 10 --max-eps 0.1 --xi 0.05
```

TRAJECTORYNET

3.1 DetectionPipeline module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

```
class DetectionPipeline.DeepSORT(max_cosine_distance: float = 10.0, max_iou_distance: float = 0.7,  
                                nn_budget: float = 100, historyDepth: int = 30, debug: bool = False)
```

Bases: object

Detector class that can generate dataset for trajectory prediction model training.

Parameters

- **source** (*Union[int, str], optional*) – Video source, can be webcam number, video path, directory path containing videos, by default 0
- **history_length** (*int, optional*) – Length of history, the detections to hold in buffer, by default 30
- **output** (*Optional[str], optional*) – Output directory, by default None
- **max_cosine_distance** (*float, optional*) – Gating threshold for cosine distance metric (object appearance), by default 10.0
- **max_iou_distance** (*float, optional*) – Max intersection over union distance, by default 0.7
- **nn_budget** (*float, optional*) – Maximum size of the appearance descriptor gallery, by default 100

source

Video source, can be webcam number, video path, directory path containing videos

Type

Union[int, str]

history_length

Length of history, the detections to hold in buffer

Type

int

output

Output directory

Type

Optional[str]

max_cosine_distance

Gating threshold for cosine distance metric (object appearance)

Type

float

max_iou_distance

Max intersection over union distance

Type

float

nn_budget

Maximum size of the appearance descriptor gallery

Type

float

historyDepth

Length of history

Type

int

_history**Type**

List

Methods

<code>init_tracker_metric(max_cosine_distance, ...)</code>	Deepsort metric factory.
<code>make_detection_object(darknetDetection)</code>	Wrap a DarknetDetection object into a DeepSORT Detection object.
<code>tracker_factory(metric, max_iou_distance, ...)</code>	Create tracker object.
<code>update_history(history, new_detections[, ...])</code>	Update trajectory history with new detections.

static init_tracker_metric(*max_cosine_distance: float, nn_budget: float, metric: str = 'cosine'*) → NearestNeighborDistanceMetric

Deepsort metric factory.

Parameters

- **max_cosince_distance** (*float*) – Garing threshold for cosine distance metric
- **nn_budget** (*float*) – Maximum size of the appearance descriptor gallery,

- **metric** (*str*, *optional*) – Metric type, by default “cosine”

static make_detection_object (*darknetDetection*: [Detection](#)) → [Detection](#)

Wrap a DarknetDetection object into a DeepSORT Detection object.

Parameters

darknetDetection (*DarknetDetection*) – Detection representing a darknet/yolo detection.

Returns

A DeepSORT Detection that is wrapped around a darknet detection.

Return type

DeepSORTDetection

static tracker_factory (*metric*: *NearestNeighborDistanceMetric*, *max_iou_distance*: *float*, *historyDepth*: *int*) → [Tracker](#)

Create tracker object.

Parameters

- **metric** (*NearestNeighborDistanceMetric*) – Distance metric object
- **max_iou_distance** (*float*) –
- **historyDepth** (*int*) – Length of history.

Returns

Tracker object.

Return type

Tracker

update_history (*history*: *List*[[TrackedObject](#)], *new_detections*: *List*[[Detection](#)], *joblibbuffer*: *List*[[TrackedObject](#)] | *None* = *None*, *db_connection*: *str* | *None* = *None*)

Update trajectory history with new detections.

Parameters

- **history** (*List*[[TrackedObject](#)]) – History list.
- **new_detections** (*List*[*DarknetDetection*]) – New detections from yolo.
- **joblibbuffer** (*Optional*[*List*[[TrackedObject](#)]], *optional*) – The joblib buffer, which will be saved at the end of runtime, by default *None*

class [DetectionPipeline.Detector](#) (*source*: *str*, *outdir*: *str* | *None* = *None*, *model*: *str* | *None* = *None*, *database*: *bool* = *False*, *joblib*: *bool* = *False*, *debug*: *bool* = *True*)

Bases: [object](#)

Detection pipeline class. This class is used to run the detection pipeline.

Parameters

- **source** (*str*) – Video source path.
- **outdir** (*str*) – Output directory path.
- **model** (*str*) – Path to model weights file.
- **database** (*bool*, *optional*) – Save results to database, by default *False*
- **joblib** (*bool*, *optional*) – Save results to joblib, by default *False*
- **debug** (*bool*, *optional*) – Debug flag, by default *True*

_source

Video source path.

Type

Path

_outdir

Output directory path.

Type

Path

_model

Loaded scikit-learn model.

Type

Model

_dataset

LoadImages object, which is used to load images from video source.

Type

LoadImages

_database

Path to database file.

Type

Path

_joblib

Path to joblib file.

Type

Path

_joblibbuffer

Joblib buffer, which is used to store TrackedObject objects.

Type

List

_history

History list, which is used to store TrackedObject objects.

Type

List

generate_db_path(*source*: str | Path, *outdir*: str | Path | None = None, *suffix*: str = '.joblib', *logger*: Logger | None = None) → Path

Generate output path name from source and output directory path.

filter_objects(*new_detections*: List | torch.Tensor, *frame_number*: int, *names*: List[str] = ['car']) → List[DarknetDetection]

Filter out detections that are not in the names list.

run(*yolo*: Yolov7, *deepSort*: DeepSORT | None = None, *show*: bool = False)

Run detection pipeline.

Examples

```
>>> from DetectionPipeline import Detector, YOLOv7, DeepSORT
>>> detector = Detector(source="path/to/video.mp4", outdir="path/to/output/directory
↳", database=True, joblib=True) # database and joblib are optional
>>> yolo = YOLOv7(weights="path/to/model/weights.pt", conf_thres=0.5, iou_thres=0.5,
↳ half=True, device="cuda", debug=True)
>>> deepSort = DeepSORT(max_age=30, debug=True)
>>> detector.run(yolo, deepSort, show=True)
```

Methods

<code>filter_objects(new_detections, frame_number)</code>	Filter out detections that are not in the names list.
<code>generate_db_path(source[, outdir, suffix, ...])</code>	Generate output path name from source and output directory path.
<code>predict(history)</code>	Predict trajectories.
<code>run(yolo[, deepSort, show])</code>	Run detection pipeline.

static `filter_objects(new_detections: List | Tensor, frame_number: int, names: List[str] = ['car'])` → List[Detection]

Filter out detections that are not in the names list.

Parameters

- **new_detections** (List) – New detections from yolo
- **frame_number** (int) – Frame number
- **names** (List[str], optional) – Names to include, by default ["car"]

Returns

List of Detection objects

Return type

List[DarknetDetection]

static `generate_db_path(source: str | Path, outdir: str | Path | None = None, suffix: str = '.joblib', logger: Logger | None = None)` → Path

Generate output path name from source and output directory path.

Parameters

- **source** (Union[str, Path]) – Video source path.
- **outdir** (Optional[Union[str, Path]], optional) – Output directory path, if none use source directory path as output directory path, by default None
- **suffix** (str) – The db suffix, eg. .joblib, .db etc...

Returns

The path to output.

Return type

Path

predict (*history*: List[TrackedObject])

Predict trajectories.

Parameters

history (List[TrackedObject]) – History list.

run (*yolo*: YOLOv7, *deepSort*: DeepSORT | None = None, *show*: bool = False)

Run detection pipeline.

Parameters

- **yolo** (YOLOv7) – YOLOv7 object
- **deepSort** (Optional[DeepSORT], optional) – DeepSORT object, by default None
- **show** (bool, optional) – Show flag to visualize frames with cv2 GUI, by default False

class DetectionPipeline.YOLOv7 (*weights*: str, *conf_thres*: float = 0.6, *iou_thres*: float = 0.4, *imgsz*: int = 640, *stride*: int = 32, *augment*: bool = False, *half*: bool = True, *device*: int | str = '0', *batch_size*: int = 1, *debug*: bool = True)

Bases: object

Class to support YOLOv7 Object Detection Pipeline. This class gives support for pre- and postprocessing images. Gives a high level interface to run inference with the yolo7 cnn pytorch model.

Parameters

- **weights** (str) – Path to weights file
- **conf_thres** (float, optional) – Confidence threshold, by default 0.6
- **iou_thres** (float, optional) – Intersection over Union threshold, by default 0.4
- **imgsz** (int, optional) – Input image size, by default 640
- **stride** (int, optional) – Convolution stride, by default 32
- **half** (bool, optional) – Half precision (from float32 to float16), by default True
- **device** (Union[int, str], optional) – Device id, by default 0
- **batch_size** (int, optional) – Size of input batch (this means that the model will run inference on batch_size images at once), by default 1
- **debug** (bool, optional) – Debug flag, by default True

_logger

Yolo class logger object

Type

Logger

device

Pytorch device object, initialized from device id given in constructor args

Type

Device

model

Loaded pytorch model

Type

Model

stride

Convolution stride, and resize stride

Type

int

imgsz

Input image size

Type

int

conf_thres

Confidence threshold

Type

float, optional

iou_thres

Intersection over Union threshold

Type

float, optional

names

List of class/label names

Type

List[str]

colors

Random colors for each class/label

Type

List[Tuple(int, int, int)]

half

If half True half precision (float16) is used, this means faster inference but lower precision

Type

bool

Methods

<i>infer</i> (img)	Run inference on input images.
<i>postprocess</i> (pred, im0, im[, show])	Run NMS on infer output, then rescale them and create a vector with [label, conf, bbox]
<i>preprocess</i> (img0)	Preprocess raw image for inference.
<i>warmup</i> ()	Warm up model.

infer(img: ndarray) → Tuple[Tensor, ndarray]

Run inference on input images.

Parameters**img** (ndarray) – Input image**Returns**

Model output of shape and input image

Return typeTuple[*Tensor*, *ndarray*]**postprocess**(*pred: Tensor*, *im0: ndarray*, *im: ndarray*, *show: bool = True*) → List

Run NMS on infer output, then rescale them and create a vector with [label, conf, bbox]

Parameters

- **pred** (*pt.Tensor*) – Predictions given by infer
- **img** (*np.ndarray*) – Preprocessed image

Returns

List of shape (n,6), where n is the number of detections per image.

Return type

List

preprocess(*img0: ndarray*) → *ndarray*

Preprocess raw image for inference. Convert to NN input img size, into shape (3,640,640). Check for half precision flag (fp32 or fp16). Normalize img by dividing image pixel values with 255. If image's shape is (3,640,640) then add additional dim.

Parameters**img0** (*ndarray*) – Input image.**Returns**

Preprocessed image of shape (3,640,640)

Return type*ndarray***warmup**()

Warm up model.

3.2 classification module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

classification.BinaryClassificationTrain(*classifier: str*, *path2db: str*, ***argv*)

Deprecated, dont use.

Will update in time.

Parameters

- **classifier** (*str*) – *_description_*

- **path2db** (*str*) – *_description_*

`classification.BinaryClassificationWorkerTrain(path2db: str, path2model=None, **argv)`

`classification.BinaryDecisionTreeClassification(path2dataset: str, min_samples: int, max_eps: float, xi: float, min_cluster_size: int, n_jobs: int, from_half=False)`

`classification.CalibratedClassification(classifier: str, path2db: str, **argv)`

Run classification on database data.

Parameters

- **classifier** (*str*) – Type of the classifier.
- **path2db** (*str*) – Path to database file.

Returns

Returns false if bad classifier was given.

Return type

bool

`classification.CalibratedClassificationWorker(path2db: str, **argv)`

Run all the classification methods implemented.

Parameters

path2db (*str*) – Path to database file.

`classification.Classification(classifier: str, path2db: str, **argv)`

Run classification on database data.

Parameters

- **classifier** (*str*) – Type of the classifier.
- **path2db** (*str*) – Path to database file.

Returns

Returns false if bad classifier was given.

Return type

bool

`classification.ClassificationWorker(path2db: str, **argv)`

Run all of the classification methods implemented.

Parameters

path2db (*str*) – Path to database file.

`classification.DTClassification(X: ndarray, y: ndarray)`

Run decision tree classification.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

`classification.GNBClassification(X: ndarray, y: ndarray)`

Run Gaussian Naive Bayes Classification on samples X and labels y.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

GNB model

Return type

skelarn classifier

`classification.GPClassification(X: ndarray, y: ndarray)`

Run Gaussian Process Classification on samples X and labels y.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

GP model

Return type

skelarn classifier

`classification.KNNClassification(X: ndarray, y: ndarray, n_neighbours: int)`

Run K Nearest Neighbours classification on samples X and labels y with neighbour numbers n_neighbours.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels
- **n_neighbours** (*int*) – Number of neighbours to belong in a class.

Returns

KNN model

Return type

sklearn classifier

`classification.MLPCClassification(X: ndarray, y: ndarray)`

Run Multi Layer Perceptron Classification on samples X and labels y.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

MLPC model

Return type

skelarn classifier

`classification.SGDClassification(X: ndarray, y: ndarray)`

Run Stochastic Gradient Descent Classification on samples X and labels y.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

SGD model

Return type

skelarn classifier

`classification.SVMClassfictaion(X: ndarray, y: ndarray)`

Run Support Vector Machine classification with RBF kernel.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

SVM model

Return type

skelarn classifier

`classification.ValidateClassification(clfmodel, X_valid: ndarray, y_valid: ndarray)`

Validate fitted classification model.

Parameters

- **clfmodel** (*str*, *model*) – Can be a path to model file or model.
- **X_valid** (*np.ndarray*) – Test dataset.
- **y_valid** (*np.ndarray*) – Test dataset's labeling.

`classification.ValidateClassification_Probability(clfmodel, X_valid: ndarray, y_valid: ndarray, threshold: float64)`

Calculate accuracy of classification model using the predict_proba method of the classifier.

Parameters

- **clfmodel** (*str*, *model*) – Can be a path to model file or model.
- **X_valid** (*np.ndarray*) – Test dataset.
- **y_valid** (*np.ndarray*) – Test dataset's labeling.

`classification.VotingClassification(X: ndarray, y: ndarray)`

Run Voting Classification on samples X and labels y.

Parameters

- **X** (*np.ndarray*) – Dataset
- **y** (*np.ndarray*) – labels

Returns

Voting model

Return type

skelarn classifier

`classification.all_class_under_threshold(predictions: ndarray, true_classes: ndarray, X: ndarray, threshold: float) → ndarray`

Return numpy array of features that's predictions for all classes are under the given threshold.

Parameters

- **predictions** (*np.ndarray*) – Probability vectors.

- **true_classes** (*np.ndarray*) – Numpy array of the true classes ordered to feature vectors.
- **X** (*np.ndarray*) – Feature vectors.
- **threshold** (*float*) – Threshold.

Returns

numpy array of feature vectors, that's classes prediction probability is under threshold.

Return type

np.ndarray

`classification.calculate_metrics_exitpoints(dataset: str, test_ratio: float, output: str, threshold: float, enter_exit_dist: float, n_jobs: int, models_to_benchmark: List[str], mse_threshold: float = 0.5, preprocessed: bool = False, test_trajectory_part: float = 1, background: str | None = None, feature_version: str = '1', **estkwargs)`

Evaluate several one-vs-rest classifiers on the given dataset. Recluster clusters based on exitpoint centroids and evaluate classifiers on the new clusters.

Parameters

- **trainingSetPath** (*str | list[str]*) – Path to training dataset or list of paths to training datasets.
- **testingSetPath** (*str | list[str]*) – Path to testing dataset or list of paths to testing datasets.
- **output** (*str*) – Output directory path, where to save the results. Tables and Models.
- **threshold** (*float*) – Threshold for filtering trajectories.
- **n_jobs** (*int*) – Number of jobs to run in parallel.
- **mse_threshold** (*float, optional*) – MSE threshold for KMeans search. Defaults to 0.5.

`classification.cross_validate(path2dataset: str, outputPath: str | None = None, train_ratio=0.75, seed=1, n_splits=5, n_jobs=18, estimator_params_set=1, classification_features_version: str = 'v1', stride: int = 15, level: float | None = None, n_weights: int = 3, weights_preset: int = 1, threshold: float = 0.7, **estkwargs)`

Run cross validation on chosen classifiers with different feature vectors.

Parameters

- **path2dataset** (*str*) – dataset path.
- **outputPath** (*str, optional*) – The path to the output table to save the results. Defaults to None.
- **train_ratio** (*float, optional*) – The ratio of the training dataset compared to the test dataset. Defaults to 0.75.
- **seed** (*int, optional*) – Seed value to be able reproduce shuffle on dataset. Defaults to 1.
- **n_splits** (*int, optional*) – Cross validation split number. Defaults to 5.
- **n_jobs** (*int, optional*) – Number of parallel processes to run. Defaults to 18.
- **estimator_params_set** (*int, optional*) – Choose classifier parameter set. Defaults to 1.

- **classification_features_version** (*str*, *optional*) – Choose which feature vector to use for classification. Defaults to “v1”.
- **stride** (*int*, *optional*) – Size of the sliding window used to generate feature vectors from detection history. Defaults to 15.
- **level** (*bool*, *optional*) – Choose if dataset should be balanced or stay as it is after enrichment. Defaults to False.
- **n_weights** (*int*, *optional*) – The number of dimensions, that is going to be added between the first and last dimension in the feature vector. Defaults to n_weights.

Returns

cross validation results in pandas datastructure

Return type

tuple

```
classification.cross_validate_multiclass(path2dataset: str, outputPath: str | None = None,
                                         train_ratio=0.75, seed=1, n_splits=5, n_jobs=18,
                                         estimator_params_set=1, classification_features_version: str =
                                         'v1', stride: int = 15, level: float | None = None, n_weights: int
                                         = 3, weights_preset: int = 1, threshold: float = 0.7,
                                         **estkwargs)
```

Run cross validation on chosen classifiers with different feature vectors.

Parameters

- **path2dataset** (*str*) – dataset path.
- **outputPath** (*str*, *optional*) – The path to the output table to save the results. Defaults to None.
- **train_ratio** (*float*, *optional*) – The ratio of the training dataset compared to the test dataset. Defaults to 0.75.
- **seed** (*int*, *optional*) – Seed value to be able reproduce shuffle on dataset. Defaults to 1.
- **n_splits** (*int*, *optional*) – Cross validation split number. Defaults to 5.
- **n_jobs** (*int*, *optional*) – Number of paralell processes to run. Defaults to 18.
- **estimator_params_set** (*int*, *optional*) – Choose classifier parameter set. Defaults to 1.
- **classification_features_version** (*str*, *optional*) – Choose which feature vector to use for classification. Defaults to “v1”.
- **stride** (*int*, *optional*) – Size of the sliding window used to generate feature vectors from detection history. Defaults to 15.
- **level** (*bool*, *optional*) – Choose if dataset should be balanced or stay as it is after enrichment. Defaults to False.
- **n_weights** (*int*, *optional*) – The number of dimensions, that is going to be added between the first and last dimension in the feature vector. Defaults to n_weights.

Returns

cross validation results in pandas datastructure

Return type

tuple

```
classification.cross_validation_multiclass_submodule(args)
```

```
classification.cross_validation_submodule(args)
```

```
classification.data_preprocessing_for_calibrated_classifier(path2db: str, min_samples=10,  
                                                         max_eps=0.2, xi=0.1,  
                                                         min_cluster_size=10, n_jobs=18)
```

Preprocess database data for classification. Load, filter, run clustering on dataset then extract feature vectors from dataset.

Parameters

- **path2db** (*str*) – *_description_*
- **min_samples** (*int, optional*) – *_description_*. Defaults to 10.
- **max_eps** (*float, optional*) – *_description_*. Defaults to 0.1.
- **xi** (*float, optional*) – *_description_*. Defaults to 0.15.
- **min_cluster_size** (*int, optional*) – *_description_*. Defaults to 10.
- **n_jobs** (*int, optional*) – *_description_*. Defaults to 18.

Returns

Return X and y train and test dataset

Return type

List[np.ndarray]

```
classification.data_preprocessing_for_classifier(path2db: str, min_samples=10, max_eps=0.2,  
                                               xi=0.1, min_cluster_size=10, n_jobs=18,  
                                               from_half=False, features_v2=False,  
                                               features_v2_half=False, features_v3=False)
```

Preprocess database data for classification. Load, filter, run clustering on dataset then extract feature vectors from dataset.

Parameters

- **path2db** (*str*) – Path to database.
- **min_samples** (*int, optional*) – Optics Clustering param. Defaults to 10.
- **max_eps** (*float, optional*) – Optics Clustering param. Defaults to 0.1.
- **xi** (*float, optional*) – Optics clustering param. Defaults to 0.15.
- **min_cluster_size** (*int, optional*) – Optics clustering param. Defaults to 10.
- **n_jobs** (*int, optional*) – Paralell jobs to run. Defaults to 18.

Returns

X_train, y_train, metadata_train, X_test, y_test, metadata_test, filteredTracks

Return type

List[np.ndarray]

```
classification.data_preprocessing_for_classifier_from_joblib_model(model, min_samples=10,  
                                                                max_eps=0.2, xi=0.15,  
                                                                min_cluster_size=10,  
                                                                n_jobs=18,  
                                                                from_half=False,  
                                                                features_v2=False,  
                                                                features_v2_half=False,  
                                                                features_v3=False)
```

Preprocess database data for classification. Load, filter, run clustering on dataset then extract feature vectors from dataset.

Parameters

- **path2db** (*str*) – *_description_*
- **min_samples** (*int*, *optional*) – *_description_*. Defaults to 10.
- **max_eps** (*float*, *optional*) – *_description_*. Defaults to 0.1.
- **xi** (*float*, *optional*) – *_description_*. Defaults to 0.15.
- **min_cluster_size** (*int*, *optional*) – *_description_*. Defaults to 10.
- **n_jobs** (*int*, *optional*) – *_description_*. Defaults to 18.

Returns

X_train, y_train, metadata_train, X_test, y_test, metadata_test

Return type

List[np.ndarray]

classification.**exitpoint_metric_module**(*args*)

classification.**investigateRenitent**(*path2model: str, threshold: float, **argv*)

Filter out renitent predictions, that cant predict which class the detections is really in.

Parameters

path2model (*str*) – Path to model.

classification.**investigate_renitent_features**(*args*)

classification.**level_features**(*X: ndarray, y: ndarray, ratio_to_min: float = 2.0*)

Level out the nuber of features.

Parameters

- **X** (*np.ndarray*) – features of shape(n_samples, n_features)
- **y** (*np.ndarray*) – labels of shape(n_samples,)

Raises

ValueError – If the length of axis 0 of both X and y are not equal raise ValueError.

Returns

Numpy array of leveled out X and y.

Return type

np.ndarray, np.ndarray

classification.**main**()

classification.**make_feature_vectors_version_eight**(*trackedObjects: List[TrackedObject], labels: ndarray, reduced_labels: ndarray, k: int = 6, window: int = 7, poly: int = 2*) → Tuple[ndarray, ndarray, ndarray, ndarray]

Generate feature vectors from the history of trackedObjects for training and evaluation.

Parameters

- (**List[TrackedObject]**) (*trackedObjects*) –
- (**np.ndarray**) (*reduced_labels*) – with the same order as the trackedObjects

- (**np.ndarray**) – from cluster reduction.
- (**int** (*poly_degree*) –
- **optional**) (*Degree of polynom. Defaults to 2.*) –
- (**int** –
- **optional**) –
- (**int** –
- **optional**) –

Returns
tuple

Return type

Tuple containing the feature vectors, the corresponding labels and some metadata

`classification.make_feature_vectors_version_five(trackedObjects: List, labels: ndarray, max_stride: int, n_weights: int)`

`classification.make_feature_vectors_version_four(trackedObjects: List, max_stride: int, labels: ndarray)`

Make multiple feature vectors from one object's history. When max_stride is reached, use sliding window method to create the vectors.

Parameters

- **trackedObjects** (*list*) – list of tracked objects
- **max_stride** (*int*) – max window size
- **labels** (*np.ndarray*) – cluster label of each tracked object

Returns

`_description_`

Return type

`_type_`

`classification.make_feature_vectors_version_one(trackedObjects: List, labels: ndarray | None = None, pooled_labels: ndarray | None = None, k: int = 6, up_until: float = 1) → Tuple[ndarray, ndarray, ndarray, ndarray]`

Make feature vectors for classification algorithm

Parameters

- **trackedObjects** (*list*) – Tracked objects
- **k** (*int*) – K is the number of slices, the object history should be sliced up into.
- **labels** (*np.ndarray*) – Results of clustering.

Returns

Tuple of arrays. First array is the feature vector array,
second array is the labels array, third is the metadata array, and the last is the reduced labels array.

Return type

`Tuple[ndarray, ndarray, ndarray, ndarray]`

`classification.make_feature_vectors_version_one_half(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors for classification algorithm

Parameters

- **trackedObjects** (*list*) – Tracked objects
- **k** (*int*) – K is the number of slices, the object history should be sliced up into.
- **labels** (*np.ndarray*) – Results of clustering.

Returns

featureVectors, labels, timeOfFeatureVectors

Return type

np.ndarray, np.ndarray, np.ndarray

`classification.make_feature_vectors_version_seven(trackedObjects: List, labels: ndarray, max_stride: int)`

`classification.make_feature_vectors_version_six(trackedObjects: List, labels: ndarray, max_stride: int, weights: ndarray)`

`classification.make_feature_vectors_version_three(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors from track histories, such as starting from the first detection incrementing the vectors length by a given factor, building multiple vectors from one history. A vector is made up from the absolute first detection of the history, a relative middle detection, and a last detection, that's index is incremented, for the next feature vector until this last detection reaches the end of the history.

Parameters

- **trackedObjects** (*list*) – Tracked objects.
- **labels** (*np.ndarray*) – Labels of the tracks, which belongs to a given cluster, given by the clustering algo.

Returns

The newly created feature vectors, the labels created for each feature vector, and the metadata that contains the information of time frames, and to which object does the feature belongs to.

Return type

tuple of numpy arrays

`classification.make_feature_vectors_version_three_half(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors from track histories, such as starting from the first detection incrementing the vectors length by a given factor, building multiple vectors from one history. A vector is made up from the absolute first detection of the history, a relative middle detection, and a last detection, that's index is incremented, for the next feature vector until this last detection reaches the end of the history.

Parameters

- **trackedObjects** (*list*) – Tracked objects.
- **labels** (*np.ndarray*) – Labels of the tracks, which belongs to a given cluster, given by the clustering algo.

Returns

The newly created feature vectors, the labels created for each feature vector, and the metadata that contains the information of time frames, and to which object does the feature belongs to.

Return type

tuple of numpy arrays

`classification.make_feature_vectors_version_two(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors from track histories, such as starting from the first detection incrementing the vectors length by a given factor, building multiple vectors from one history. A vector is made up from the absolute first detection of the history, a relative middle detection, and a last detection, that's index is incremented, for the next feature vector until this last detection reaches the end of the history. Next to the coordinates, also the velocity of the object is being included in the feature vector.

Parameters

- **trackedObjects** (*list*) – Tracked objects.
- **labels** (*np.ndarray*) – Labels of the tracks, which belongs to a given cluster, given by the clustering algo.

Returns

The newly created feature vectors, the labels created for each feature vector, and the metadata that contains the information of time frames, and to which object does the feature belongs to.

Return type

tuple of numpy arrays

`classification.make_feature_vectors_version_two_half(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors from track histories, such as starting from the first detection incrementing the vectors length by a given factor, building multiple vectors from one history. A vector is made up from the absolute first detection of the history, a relative middle detection, and a last detection, that's index is incremented, for the next feature vector until this last detection reaches the end of the history. Next to the coordinates, also the velocity of the object is being included in the feature vector.

Parameters

- **trackedObjects** (*list*) – Tracked objects.
- **labels** (*np.ndarray*) – Labels of the tracks, which belongs to a given cluster, given by the clustering algo.

Returns

The newly created feature vectors, the labels created for each feature vector, and the metadata that contains the information of time frames, and to which object does the feature belongs to.

Return type

tuple of numpy arrays

`classification.make_features_for_classification(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors for classification algorithm

Parameters

- **trackedObjects** (*list*) – Tracked objects
- **k** (*int*) – K is the number of slices, the object history should be sliced up into.
- **labels** (*np.ndarray*) – Results of clustering.

Returns

featurevectors and the new labels to fit the featurevectors

Return type

np.ndarray

`classification.make_features_for_classification_velocity(trackedObjects: List, k: int, labels: ndarray)`

Make feature vectors for classification algorithm

Parameters

- **trackedObjects** (*list*) – Tracked objects
- **k** (*int*) – K is the number of slices, the object history should be sliced up into.
- **labels** (*np.ndarray*) – Results of clustering.

Returns

featureVectors, labels

Return type

np.ndarray, np.ndarray

`classification.plot_decision_tree(path2model: str)`

Draw out the decision tree in a tree graph.

Parameters

path2model (*str*) – Path to the joblib binary model file.

`classification.plot_module(args)`

`classification.preprocess_dataset_for_training(path2dataset: str, min_samples=10, max_eps=0.2, xi=0.15, min_cluster_size=10, n_jobs=18, cluster_features_version: str = '4D', threshold: float = 0.4, classification_features_version: str = 'v1', stride: int = 15, level: float | None = None, n_weights: int = 3, weights_preset: int = 1, p_norm: int = 2)`

`classification.train_binary_classifiers(path2dataset: str, outdir: str, **argv)`

`classification.train_binary_classifiers_submodule(args)`

`classification.true_class_under_threshold(predictions: ndarray, true_classes: ndarray, X: ndarray, threshold: float) → ndarray`

Return numpy array of featurevectors that's predictions for their true class is under given threshold.

Parameters

- **predictions** (*np.ndarray*) – Probability vectors.
- **true_classes** (*np.ndarray*) – Numpy array of the true classes ordered to feature vectors.
- **X** (*np.ndarray*) – Feature vectors.
- **threshold** (*float*) – Threshold.

Returns

numpy array of feature vectors, that's true class's prediction probability is under threshold.

Return type

np.ndarray

`classification.validate_models(path2models: str, **argv)`

Validate trained classifiers.

Parameters

path2models (*str*) – Path to parent directory containing models.

3.3 classifier module

Binary Classifier Class implementation Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

```
class classifier.BinaryClassifier(trackData: list, classifier: ClassifierMixin, classifier_argv: dict | None =  
                                None)
```

Bases: ClassifierMixin, BaseEstimator

Base Binary Classifier

A classifier that can take any type of scikit-learn classifier as the binary classifier. At the creation of the classifier object, a scikit learn classifier have to be given as the first argument, the second argument is a dictionary of args that will be passed to the scikit learn classifier.

Parameters

classifier (*scikit-learn classifier*) – A scikit-learn classifier that will be used to create the multiclass binary classifier.

Attributes

y_
[ndarray, shape (n_samples,)] The labels passed during *fit()*.

classes_
[ndarray, shape (n_classes,)] The classes seen at *fit()*.

label_mtx_
[np.ndarray shape (n_cluster, n_samples)] Matrix of labels, each label gets its own array, that is filled with the actual label and the other labels but labeled as class 0.

models

[scikit-learn classifier] Sklearn Classifiers fitted as binary Classifiers.

Methods

<code>fit(X, y)</code>	Fit sklearn classifier models with given dataset X and class labels y.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X[, threshold, top])</code>	Return predicted top labels of dataset X
<code>predict_proba(X)</code>	Return predicted probabilities of dataset X
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>validate(X_test, y_test, threshold)</code>	Validate trained models.
<code>validate_predictions(X_test, y_test, threshold)</code>	Validate trained models.

fit(X: ndarray, y: ndarray)

Fit sklearn classifier models with given dataset X and class labels y.

Parameters

- **X** (np.ndarray shape (n_samples, n_features)) – The training input samples.
- **y** (np.ndarray shape (n_samples,)) – The target values. An array of int. (labels)

Returns

self – Returns self. Fitted model.

Return type

object

predict(X: ndarray, threshold: float32 = 0.5, top: int = 1)

Return predicted top labels of dataset X

Parameters

- **X** (np.ndarray) – Feature vector of shape(n_samples, n_features) for prediction.
- **top** (int) – Number of classes with the highest probability

Returns

lists of prediction result class labels, length=top

Return type

np.ndarray

predict_proba(X: ndarray)

Return predicted probabilities of dataset X

Parameters

- **X** (np.ndarray shape (n_samples, n_features)) –
- **Returns** –
- **np.ndarray** (shape (n_samples, n_classes)) – Prediction probabilities of

validate(X_test: ndarray, y_test: ndarray, threshold: float32)

Validate trained models. :param X_test: Validation dataset of shape(n_samples, n_features). :type X_test: np.ndarray :param y_test: Validation class labels shape(n_samples, 1). :type y_test: np.ndarray :param threshold: Probability threshold, if prediction probability higher than the threshold, then it counts as a valid prediction. :type threshold: np.float32

validate_predictions(*X_test: ndarray, y_test: ndarray, threshold: float32, top: int = 1*)

Validate trained models. :param X_test: Validation dataset of shape(n_samples, n_features). :type X_test: np.ndarray :param y_test: Validation class labels shape(n_samples, 1). :type y_test: np.ndarray :param threshold: Probability threshold, if prediction probability higher than the threshold, then it counts as a valid prediction. :type threshold: np.float32

class classifier.OneVsRestClassifierExtended(*estimator, tracks=None, n_jobs=16, centroid_labels=None, centroid_coordinates=None*)

Bases: `OneVsRestClassifier`

Extended One vs. Rest Classifier.

This class extends the *OneVsRestClassifier* class from scikit-learn to provide additional functionality. Specifically, the *predict()* method takes two extra arguments, *threshold* and *top_n*. Only the top *n* classes over the *threshold* will be returned. Additionally, two new methods are implemented: *validate()* and *validate_predictions()*. The *validate()* method calculates the balanced accuracy of the classifier. It takes three arguments: *X_test*, *y_test*, and *threshold*. *X_test* and *y_test* are the testing datasets, and *threshold* is a float value that determines the threshold for the classifier. The *validate_predictions()* method validates the predictions of the classifier. It takes two arguments: *X_test* and *y_test*.

estimator

A regressor or a classifier that implements fit. When a classifier is passed, *decision_function* will be used in priority and it will fallback to *predict_proba* if it is not available. When a regressor is passed, *predict* is used.

Type

estimator object

tracks

Filtered track dataset, that was used to create feature vectors for clustering, and X, y feature vectors and labels.

Type

list

n_jobs

Number of processes to run. Default n_jobs value is 16.

Type

int

centroid_labels

List of centroid labels.

Type

list

centroid_coordinates

List of centroid coordinates.

Type

list

Parameters

- **estimator** (*classifier*) – Scikit-Learn classifier object, that will be used to create the binary classifiers.
- **tracks** (*list[TrackedObject], optional*) – Filtered track dataset, that was used to create feature vectors for clustering, and X, y feature vectors and labels.

- **n_jobs** (*int*, *optional*) – Number of processes to run. Default n_jobs value is 16.
- **centroid_labels** (*list*, *optional*) – List of centroid labels.
- **centroid_coordinates** (*list*, *optional*) – List of centroid coordinates.

Attributes

coef_

DEPRECATED: Attribute *coef_* was deprecated in version 0.24 and will be removed in 1.1 (renaming of 0.26).

intercept_

DEPRECATED: Attribute *intercept_* was deprecated in version 0.24 and will be removed in 1.1 (renaming of 0.26).

multilabel_

Whether this is a multilabel classifier.

n_classes_

Number of classes.

Methods

decision_function (X)	Decision function for the OneVsRestClassifier.
fit (X, y[, centroids, scale])	Fit the classifier to the training data.
get_params ([deep])	Get parameters for this estimator.
partial_fit (X, y[, classes, centroids])	Incrementally trains the classifier on a batch of samples.
predict (X[, top, centroids])	Predict class labels for samples in X.
predict_proba (X[, classes, centroids, scale])	Predict class probabilities for X.
score (X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params (**params)	Set the parameters of this estimator.
validate (X_test, y_test, threshold[, centroids])	Validate the classifier on the given test data and return the balanced accuracy for each class.
validate_predictions (X_test, y_test[, top, ...])	Validates the predictions made by the model.

fit(X, y, centroids: dict | None = None, scale: bool = False)

Fit the classifier to the training data.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like of shape (n_samples,)*) – The target values.
- **centroids** (*dict*, *optional*) – A dictionary containing the centroids for each class. If provided, the training data will be augmented with the distances between each sample and its corresponding class centroid.
- **scale** (*bool*, *default=False*) – Whether to scale the input data before fitting the classifier.

Returns

self – Returns self.

Return type

object

partial_fit(*X*, *y*, *classes=None*, *centroids=None*)

Incrementally trains the classifier on a batch of samples.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The input data.
- **y** (*array-like of shape (n_samples,)*) – The target values.
- **classes** (*array-like of shape (n_classes,)*, *default=None*) – List of all the classes that can possibly appear in the y vector. Must be provided on the first call to `partial_fit`, but can be omitted on subsequent calls if the set of classes hasn't changed.
- **centroids** (*dict*, *default=None*) – A dictionary containing the centroids for each class. If provided, the function will compute the distance between each sample and the corresponding class centroid, and add these distances as additional features to the input data.

Returns

self – Returns self.

Return type

object

predict(*X: ndarray*, *top: int = 1*, *centroids: dict | None = None*)

Predict class labels for samples in X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The input samples.
- **top** (*int*, *optional (default=1)*) – The number of top classes to return.
- **centroids** (*dict*, *optional (default=None)*) – The centroids of the classes.

Returns

prediction_result – The predicted class labels for each sample in X.

Return type

ndarray of shape (n_samples, top)

predict_proba(*X: ndarray*, *classes: ndarray | None = None*, *centroids: dict | None = None*, *scale: bool = False*)

Predict class probabilities for X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The input samples.
- **classes** (*array-like of shape (n_classes,)*, *default=None*) – The classes to consider for classification. By default, all classes in the training data are considered.
- **centroids** (*dict*, *default=None*) – A dictionary containing the centroids for each class. If provided, the function will compute the distance between each sample and the corresponding class centroid, and add these distances as additional features to the input data.
- **scale** (*bool*, *default=False*) – Whether to scale the input data using the scaler fitted during training.

Returns

Y – The class probabilities of the input samples. The order of the classes corresponds to that in the attribute `classes_`.

Return type

ndarray of shape (n_samples, n_classes)

Raises

- **NotFittedError** – If the estimator is not fitted.
- **ValueError** – If the input data is not a 2D array.

Notes

If *centroids* is provided, the function will add 6 additional features to the input data, corresponding to the Euclidean distance between each sample and the centroid of each class. This is done to allow the classifier to take into account the relative position of each sample with respect to the class centroids.

If *classes* is provided, the function will only compute the class probabilities for the specified classes, and will return a matrix of shape (n_samples, len(classes)).

validate(*X_test*: ndarray, *y_test*: ndarray, *threshold*: float32, *centroids*: dict | None = None)

Validate the classifier on the given test data and return the balanced accuracy for each class.

Parameters

- **X_test** (*np.ndarray*) – The test data to be used for validation.
- **y_test** (*np.ndarray*) – The true labels for the test data.
- **threshold** (*np.float32*) – The probability threshold for classification.
- **centroids** (*dict*, *optional*) – The centroids for each class, used for weighted classification.

Returns

balanced_accuracy – The balanced accuracy for each class.

Return type

list of float

validate_predictions(*X_test*: ndarray, *y_test*: ndarray, *top*: int = 1, *centroids*: dict | None = None)

Validates the predictions made by the model.

Parameters

- **X_test** (*np.ndarray*) – The input data to validate the predictions for.
- **y_test** (*np.ndarray*) – The true class labels for the input data.
- **top** (*int*, *optional*) – The number of top predictions to consider, by default 1.
- **centroids** (*dict*, *optional*) – A dictionary containing the centroids of the classes, by default None.

Returns

The mean average precision (MAP) score for the predictions.

Return type

float

Notes

This function calculates the mean average precision (MAP) score for the predictions made by the model. The MAP score is calculated as the ratio of true positive predictions to the total number of predictions made. The function takes in the input data *X_test* and the true class labels *y_test*, and optionally the number of top predictions to consider *top* and a dictionary containing the centroids of the classes *centroids*. If *centroids* is not provided, the function will use the default centroids calculated during training.

Examples

```
>>> classifier = TrajectoryNetClassifier()
>>> X_test = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> y_test = np.array([0, 1, 2])
>>> map_score = classifier.validate_predictions(X_test, y_test)
>>> print(map_score)
0.0
```

3.4 clustering module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

clustering.affinityPropagation_on_enter_and_exit_points(*path2db: str, threshold: float*)

Run affinity propagation clustering on first and last detections of objects. This way, the enter and exit areas on a video can be determined.

Parameters

- **path2db** (*str*) – Path to database file
- **threshold** (*float*) – Threshold value for filtering algorithm.

clustering.affinityPropagation_on_featureVector(*featureVectors: ndarray*)

Run affinity propagation clustering algorithm on list of feature vectors.

Parameters

featureVector (*list*) – A numpy ndarray of numpy ndarrays. ex.: `[[x,y,x,y], [x2,y2,x2,y2]]`

clustering.aoi_clutsering_search_birch(*tracks_path, outdir, threshold, n_jobs=18, dimensions='4D', **estkwargs*)

`clustering.calc_cluster_centers(tracks, labels, exit=True)`

Calculate center of mass for every class's exit points. These will be the exit points of the clusters.

Parameters

- **tracks** (*List[TrackedObject]*) – List of tracked objects
- **labels** (*_type_*) – Labels of tracked objects

Returns

The center of mass for every class's exits points

Return type

cluster_centers

`clustering.cluster_optics_dbscan_on_featurevectors(featureVectors: ndarray, min_samples: int, xi: float, min_cluster_size: float, eps: float, n_jobs=-1)`

Run clustering with optics, then dbscan using the results of the optics clustering.

Parameters

- **featureVectors** (*np.ndarray*) – Numpy array of feature vectors, value of a feature vector can vary.
- **min_samples** (*int*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **xi** (*float*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.
- **min_cluster_size** (*float*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2).
- **eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **n_jobs** (*int, optional*) – Number of processes to run simultaneously. Defaults to -1.

Returns

list of labels

Return type

list

`clustering.cluster_optics_dbscan_on_nx4(trackedObjects: List, min_samples: int, xi: float, min_cluster_size: float, eps: float, threshold: float, outdir: str, n_jobs=16, show=True)`

Run optics clustering on N x 4 (x,y,x,y) feature vectors.

Parameters

- **trackedObjects** (*list*) – List of tracks.
- **min_samples** (*int*) – The number of samples in a neighborhood for a point to be considered as a core point. Also, up and down steep regions can't have more than min_samples consecutive non-steep points.
- **xi** (*float*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.

- **min_cluster_size** (*float*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2).
- **eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **threshold** (*float*) – Threshold value for filtering algorithm.
- **path2db** (*str*) – Path to database file.
- **show** (*bool, optional*) – Boolean flag to show plot. Defaults to True.

```
clustering.cluster_optics_dbscan_plotter(path2db: str, outdir: str, min_samples=10, xi=0.05,  
                                         threshold=0.3, min_cluster_size=0.05, eps=0.2, n_jobs=16)
```

```
clustering.clustering_on_2D_feature_vectors(estimator, trackedObjects: List[TrackedObject], outdir:  
                                           str, n_jobs: int = -1, filter_threshold: float | None = None,  
                                           **estkwargs)
```

Run clustering on 4 dimensional feature vectors. Create plots of every cluster with their tracks, and create histograms from the length of the tracks.

Parameters

- **estimator** (*estimator*) – sklearn estimator class
- **trackedObjects** (*list[TrackedObject]*) – TrackedObject python object dataset.
- **outdir** (*str*) – Output directory path.
- **n_jobs** (*int, optional*) – Number of processes to run. Defaults to -1, that means all cpu threads will be utilized.

```
clustering.clustering_on_4D_feature_vectors(estimator, trackedObjects: List[TrackedObject], outdir:  
                                           str, n_jobs: int = -1, filter_threshold: float | None = None,  
                                           **estkwargs)
```

Run clustering on 4 dimensional feature vectors. Create plots of every cluster with their tracks, and create histograms from the length of the tracks.

Parameters

- **estimator** (*estimator*) – sklearn estimator class
- **trackedObjects** (*list[TrackedObject]*) – TrackedObject python object dataset.
- **outdir** (*str*) – Output directory path.
- **n_jobs** (*int, optional*) – Number of processes to run. Defaults to -1, that means all cpu threads will be utilized.

```
clustering.clustering_on_6D_feature_vectors(estimator, trackedObjects: List[TrackedObject], outdir:  
                                           str, n_jobs: int = -1, filter_threshold: float | None = None,  
                                           **estkwargs)
```

Run clustering on 6 dimensional feature vectors. Create plots of every cluster with their tracks, and create histograms from the length of the tracks.

Parameters

- **estimator** (*estimator*) – sklearn estimator class
- **trackedObjects** (*list[TrackedObject]*) – TrackedObject python object dataset.
- **outdir** (*str*) – Output directory path.

- **n_jobs** (*int*, *optional*) – Number of processes to run. Defaults to -1, that means all cpu threads will be utilized.

`clustering.clustering_on_feature_vectors(X: ndarray, estimator, n_jobs: int = -1, **estkwargs)`

Run clustering with given estimator with given parameters.

Parameters

- **X** (*np.ndarray*) – Feature vectors
- **estimator** (*estimator*) – sklearn estimator class
- **n_jobs** (*int*, *optional*) – Number of processes to run. Defaults to -1, that means all cpu threads will be utilized.

Returns

The labels ordered to the X features.

Return type

ndarray

`clustering.clustering_search_on_2D_feature_vectors(estimator, database: str, outdir: str, filter_threshold: float = (0.1, 0.7), n_jobs: int = -1, **estkwargs)`

`clustering.clustering_search_on_4D_feature_vectors(estimator, database: str, outdir: str, filter_threshold: float = (0.1, 0.7), n_jobs: int = -1, **estkwargs)`

`clustering.clustering_search_on_6D_feature_vectors(estimator, database: str, outdir: str, filter_threshold: float = (0.1, 0.7), n_jobs: int = -1, **estkwargs)`

`clustering.dbscan_clustering_on_nx4(trackedObjects: List, eps: float, min_samples: int, n_jobs: int, threshold: float, outdir: str, show=True, shuffle=False)`

Run dbscan clustering on N x 4 (x,y,x,y) feature vectors.

Parameters

- **trackedObjects** (*list*) – List of tracks.
- **eps** (*float*, *optional*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other. Defaults to 0.1.
- **min_samples** (*int*, *optional*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **n_jobs** (*int*, *optional*) – The number of parallel jobs to run.
- **threshold** (*float*) – Threshold value for filtering algorithm.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **show** (*bool*, *optional*) – Boolean flag value to show plot or not. Defaults to True.

`clustering.dbscan_on_featureVectors(featureVectors: ndarray, eps: float, min_samples: int, n_jobs: int)`

Run dbscan clustering algorithm on extracted feature vectors.

Parameters

- **featureVectors** (*np.ndarray*) – A numpy array of extracted features to run the clustering on.

- **eps** (*float, optional*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other. Defaults to 0.1.
- **min_samples** (*int, optional*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **n_jobs** (*int, optional*) – The number of parallel jobs to run.

Returns

Cluster labels for each point in the dataset given to fit(). Noisy samples are given the label -1.

Return type

labels

`clustering.dbscan_worker(path2db: str, outdir: str, eps: float, min_samples: int, n_jobs: int, threshold=(0.1, 0.7), k=(2, 16), shuffle=False)`

Run dbscan clustering on different threshold and n_cluster levels.

Parameters

- **path2db** (*str*) – Path to database file.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **min_samples** (*int*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **n_jobs** (*int*) – The number of parallel jobs to run.
- **threshold** (*tuple, optional*) – Threshold for filtering algorithm. Defaults to (0.1, 0.7).
- **k** (*tuple, optional*) – n_cluster number. Defaults to (2,16).

Returns

Returns False if bad k parameters were given.

Return type

bool

`clustering.elbow_on_clustering(X: ndarray, threshold: float, dirpath: str, model='kmeans', metric='silhouette', show=True)`

Plot elbow diagram with kmeans and spectral clustering, and with different thresholds. Use this function instead of elbow_visualizer if want to save plot.

Parameters

- **path2db** (*str*) – Path to database.
- **threshold** (*int, tuple, list*) – Give a range of threshold to do filtering with.
- **model** (*str*) – Name of cluster algorithm. Choices: 'silhouette', 'calinski-harabasz', 'davies-bouldin'.

`clustering.elbow_on_kmeans(path2db: str, threshold: float, n_jobs=None)`

Evaluate clustering results and create elbow diagram.

Parameters

- **path2db** (*str*) – Path to database file.
- **threshold** (*float*) – Threshold value for filtering algorithm.

`clustering.elbow_plot_worker(path2db: str, threshold=(0.1, 0.7), n_jobs=None)`

This function generates multiple elbow plots, with different clustering algorithms, score metrics and thresholds.

Parameters

- **path2db** (*str*) – Path to database file.
- **threshold** (*tuple*, *optional*) – Threshold range. Defaults to (0.01, 0.71).

`clustering.elbow_plotter(path2db: str, threshold: float, model: str, metric: str, n_jobs=None)`

Simply plots an elbow diagram with the given parameters.

Parameters

- **path2db** (*str*) – Path to database file.
- **threshold** (*float*) – threshold value for filtering algorithm
- **model** (*str*) – clustering algorithm
- **metric** (*str*) – scoring metric

`clustering.elbow_visualizer(X, k, model='kmeans', metric='silhouette', distance_metric='euclidean', show=False) → Figure`

Create elbow plot, to visualize what cluster number fits the best for the dataset.

Parameters

- **X** (*np.ndarray*) – dataset for the clustering
- **k** (*int*, *tuple*, *list*) – number of clusters, can be an int, tuple, list, if int then it will run clusters from 2 to given number
- **model** (*str*, *optional*) – Name of clustering algorithm. Defaults to 'kmeans'. Choices: 'kmeans', 'spectral'.
- **metric** (*str*, *optional*) – The scoring metric, to score the clusterings with. Defaults to 'silhouette'. Choices: 'silhouette', 'calinski-karabasz', 'davies-bouldin'.
- **distance_metric** (*str*, *optional*) – Some of the metric algorithm need a distance metric. Defaults to 'euclidean'. For now this is the only one, but who knows what the future brings.
- **show** (*bool*) – True if want to show plot

Returns

Returns a matplotlib figure object, that can be saved.

Return type

plt.Figure

`clustering.k_means_on_featureVectors(featureVectors: ndarray, n_clusters: int)`

Run kmeans clustering algorithm on extracted feature vectors.

Parameters

- **featureVectors** (*np.ndarray*) – a numpy array of extracted features
- **n_clusters** (*int*) – number of initial clusters for kmeans

Returns

vector of labels, same length as given featureVectors vector

Return type

np.ndarray

`clustering.kmeans_clustering_on_nx2(path2db: str, n_clusters: int, threshold: float)`

Run kmeans clustering on filtered feature vectors.

Parameters

- **path2db** (*str*) – Path to database file.
- **n_clusters** (*int*) – number of initial clusters for kmeans
- **threshold** (*float*) – the threshold for the filtering algorithm

`clustering.kmeans_clustering_on_nx4(trackedObjects: List, n_clusters: int, threshold: float, outdir: str, show=True)`

Run kmeans clustering on N x 4 (x,y,x,y) feature vectors.

Parameters

- **trackedObjects** (*list*) – List of object tracks.
- **n_clusters** (*int*) – Number of clusters.
- **threshold** (*float*) – Threshold value for the false positive filter algorithm.
- **outdir** (*str*) – Output directory path, where to save plotted images.

`clustering.kmeans_mse_clustering(X: ndarray, Y: ndarray, n_jobs: int = 10, mse_threshold: float = 0.5) → Tuple[ndarray, KMeans, int, float]`

Run kmeans clustering with different number of clusters, and calculate mean squared error for each cluster. Return the clustering results based on cluster exit point centers.

Parameters

- **X** (*np.ndarray*) – Already clustered trajectories.
- **Y** (*np.ndarray*) – Labels of trajectories.
- **n_jobs** (*int, optional*) – Number of processes to run. Defaults to 10.
- **mse_threshold** (*float, optional*) – The mean squared error threshold of min search. Defaults to 0.5.

Returns

Returns the labels of the trajectories, the classifier with the best mean squared error, the number of clusters and the mean squared error.

Return type

`tuple(ndarray, classifier, int, float)`

`clustering.kmeans_mse_search(database: str, dirpath: str, threshold: float = 0.7, n_jobs: int = 10, mse_threshold: float = 0.5, preprocessed: bool = False, **estkwargs)`

Run kmeans clustering with different number of clusters, and calculate mean squared error for each cluster. Return the number of clusters where the mean squared error is below the threshold.

Parameters

- **database** (*str*) – Database path.
- **dirpath** (*str*) – Path to directory where to save the plot.
- **threshold** (*float, optional*) – Trajectory filtering threshold. Defaults to 0.7.
- **n_jobs** (*int, optional*) – Number of processes to run. Defaults to 10.
- **mse_threshold** (*float, optional*) – Mean squared error threshold. Defaults to 0.5.

Returns

Returns the classifier with the best number of clusters, the number of clusters and the mean squared error.

Return type

tuple(classifier, int, float)

`clustering.kmeans_worker(path2db: str, outdir: str, threshold=(0.1, 0.7), k=(2, 16), n_jobs=None)`

This function automates the task of running kmeans clustering on different cluster numbers.

Parameters

- **path2db** (*str*) – path to database file
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **n_cluster_start** (*int*) – starting number cluster
- **n_cluster_end** (*int*) – ending number cluster
- **threshold** (*float*) – threshold for filtering algorithm

Returns

returns false if some crazy person uses the program

Return type

bool

`clustering.main()`

`clustering.makeFeatureVectors_Nx2(trackedObjects: List) → ndarray`

Create 2D feature vectors from tracks. The enter and exit coordinates are put in different vectors. Only creating 2D vectors.

Parameters

trackedObjects (*list*) – list of tracked objects

Returns

numpy array of feature vectors

Return type

np.ndarray

`clustering.make_2D_feature_vectors(trackedObjects: List) → ndarray`

Create 2D feature vectors from tracks. The enter and exit coordinates are put in one vector. Creating 4D vectors.
v = [exitX, exitY]

Parameters

trackedObjects (*list*) – list of tracked objects

Returns

numpy array of feature vectors

Return type

np.ndarray

`clustering.make_4D_feature_vectors(trackedObjects: List) → ndarray`

Create 4D feature vectors from tracks. The enter and exit coordinates are put in one vector. Creating 4D vectors.
v = [enterX, enterY, exitX, exitY]

Parameters

trackedObjects (*list*) – list of tracked objects

Returns

numpy array of feature vectors

Return type

np.ndarray

`clustering.make_6D_feature_vectors(trackedObjects: List) → ndarray`

Create 6D feature vectors from tracks. The enter, middle and exit coordinates are put in one vector. Creating 6D vectors. $v = [\text{enterX}, \text{enterY}, \text{exitX}, \text{exitY}]$

Parameters

trackedObjects (*list*) – list of tracked objects

Returns

numpy array of feature vectors

Return type

np.ndarray

`clustering.optics_clustering_on_nx4(trackedObjects: List, min_samples: int, xi: float, min_cluster_size: float, max_eps: float, threshold: float, outdir: str, n_jobs=16, show=True)`

Run optics clustering on $N \times 4$ (x,y,x,y) feature vectors.

Parameters

- **trackedObjects** (*list*) – List of tracks.
- **min_samples** (*int*) – The number of samples in a neighborhood for a point to be considered as a core point. Also, up and down steep regions can't have more than min_samples consecutive non-steep points.
- **xi** (*float*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.
- **min_cluster_size** (*float*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2).
- **max_eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **threshold** (*float*) – Threshold value for filtering algorithm.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **show** (*bool, optional*) – Boolean flag to show plot. Defaults to True.

`clustering.optics_dbscan_worker(path2db: str, outdir: str, min_samples=10, xi=0.05, min_cluster_size=0.05, eps=0.2, threshold=(0.1, 0.7), k=(2, 16), n_jobs=16)`

Run dbscan clustering on different threshold and n_cluster levels.

Parameters

- **path2db** (*str*) – Path to database file.
- **min_samples** (*int*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **xi** (*float*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.

- **min_cluster_size** (*float*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2).
- **n_jobs** (*int*) – The number of parallel jobs to run.
- **threshold** (*tuple*, *optional*) – Threshold for filtering algorithm. Defaults to (0.1, 0.7).
- **k** (*tuple*, *optional*) – n_cluster number. Defaults to (2,16).

Returns

Returns False if bad k parameters were given.

Return type

bool

`clustering.optics_on_featureVectors(featureVectors: ndarray, min_samples: int, xi: float, min_cluster_size: float, n_jobs: int, max_eps: float, p: int)`

Run optics clustering algorithm on extracted feature vectors.

Parameters

- **featureVectors** (*np.ndarray*) – A numpy array of extracted features to run the clustering on.
- **min_samples** (*int*, *optional*) – The number of samples in a neighborhood for a point to be considered as a core point. Defaults to 10.
- **xi** (*float*, *optional*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary. Defaults to 0.05.
- **min_cluster_size** (*float*, *optional*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2). If None, the value of min_samples is used instead. Defaults to 0.05.
- **max_eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **p** (*int*) – Parameter for distance metric. Default = 2.

Returns

Cluster labels for each point in the dataset given to fit(). Noisy samples and points which are not included in a leaf cluster of **cluster_hierarchy_** are labeled as -1.

Return type

labels

`clustering.optics_worker(path2db: str, outdir: str, min_samples: int, xi: float, min_cluster_size: float, max_eps: float, threshold=(0.1, 0.7), k=(2, 16), n_jobs=16)`

Run dbscan clustering on different threshold and n_cluster levels.

Parameters

- **path2db** (*str*) – Path to database file.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **min_samples** (*int*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **xi** (*float*) – Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.

- **min_cluster_size** (*float*) – Minimum number of samples in an OPTICS cluster, expressed as an absolute number or a fraction of the number of samples (rounded to be at least 2).
- **n_jobs** (*int*) – The number of parallel jobs to run.
- **threshold** (*tuple*, *optional*) – Threshold for filtering algorithm. Defaults to (0.1, 0.7).
- **k** (*tuple*, *optional*) – n_cluster number. Defaults to (2,16).

Returns

Returns False if bad k parameters were given.

Return type

bool

`clustering.simple_dbscan_plotter(path2db: str, threshold: float, eps: float, min_samples: int, n_jobs: int)`

Run dbscan on dataset.

Parameters

- **path2db** (*str*) – Path to database file.
- **threshold** (*float*) – Threshold for filtering algorithm.
- **eps** (*float*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- **min_samples** (*int*) – The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
- **n_jobs** (*int*) – The number of parallel jobs to run.

`clustering.simple_kmeans_plotter(path2db: str, outdir: str, threshold: float, n_clusters: int, n_jobs=None)`

Just plots and saves one clustering.

Parameters

- **path2db** (*str*) – Path to database.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **threshold** (*float*) – threshold value to filtering algorithm
- **n_clusters** (*int*) – number of clusters

`clustering.simple_optics_plotter(path2db: str, outdir: str, min_samples=10, xi=0.05, threshold=0.3, min_cluster_size=0.05, max_eps=0.2, n_jobs=16)`

`clustering.simple_spectral_plotter(path2db: str, outdir: str, threshold: float, n_clusters: int, n_jobs=None)`

Create on spectral clustering plot with given parameters.

Parameters

- **path2db** (*str*) – Path to database
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **threshold** (*float*) – threshold value for filtering algorithm
- **n_clusters** (*int*) – number of cluster

`clustering.spectral_clustering_on_nx4`(*trackedObjects: List, n_clusters: int, threshold: float, outdir: str, show=True*)

Run spectral clustering on N x 4 (x,y,x,y) feature vectors.

Parameters

- **trackedObjects** (*list[TrackedObjects]*) – Track dataset.
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **n_clusters** (*int*) – Number of clusters.
- **threshold** (*float*) – Threshold value for the false positive filter algorithm.
- **outdir** – Output directory path, where to save plotted images.

`clustering.spectral_on_featureVectors`(*featureVectors: ndarray, n_clusters: int*)

Run spectral clustering algorithm on extracted feature vectors.

Parameters

- **featureVectors** (*np.ndarray*) – a numpy array of extracted features
- **n_clusters** (*int*) – number of initial clusters for spectral

Returns

vector of labels, same length as given featureVectors vector

Return type

np.ndarray

`clustering.spectral_worker`(*path2db: str, outdir: str, threshold=(0.1, 0.7), k=(2, 16), n_jobs=None*)

This function automates the task of running spectral clustering on different cluster numbers.

Parameters

- **path2db** (*str*) – path to database file
- **outdir** (*str*) – Output directory path, where to save plotted images.
- **n_cluster_start** (*int*) – starting number cluster
- **n_cluster_end** (*int*) – ending number cluster
- **threshold** (*float*) – threshold for filtering algorithm

Returns

returns false if some crazy person uses the program

Return type

bool

`clustering.submodule_aoi_birch`(*args*)

`clustering.submodule_aoi_kmeans`(*args*)

`clustering.submodule_birch`(*args*)

`clustering.submodule_dbscan`(*args*)

`clustering.submodule_kmeans`(*args*)

`clustering.submodule_optics(args)`

Optics clustering has 3 important parameters, min samples, max eps and xi. The goal is to close the gap between the number of input feature vectors and the clustered feature vectors. The ratio of this - clustered / all - equation should be closing on in 1.0. An other important factor is the cluster number, it should represent our data.

Parameters

args (*arguments object*) – Console line arguments.

`clustering.upscale_cluster_centers(centroids, framewidth: int, frameheight: int)`

Scale centroids of clusters up to the video's resolution.

Parameters

- **centroids** (*dict*) – Output of `aoiextraction()` function.
- **framewidth** (*int*) – Width resolution of the video.
- **frameheight** (*int*) – Height resolution of the video.

Returns

Upscaled centroid coordinates.

Return type

dict

3.5 converter module

3.6 dataManagementClasses module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

class dataManagementClasses.Detection(*label: str, confidence: float, X: float, Y: float, Width: float, Height: float, frameID: int*)

Bases: object

A class representing a detection in a video frame.

label

The label of the detected object.

Type

str

confidence

The confidence score of the detection.

Type

float

X

The x-coordinate of the top-left corner of the bounding box.

Type

float

Y

The y-coordinate of the top-left corner of the bounding box.

Type

float

Width

The width of the bounding box.

Type

float

Height

The height of the bounding box.

Type

float

frameID

The ID of the frame in which the detection was made.

Type

int

VX

The x-velocity of the detected object. Default is None.

Type

float, optional

VY

The y-velocity of the detected object. Default is None.

Type

float, optional

AX

The x-acceleration of the detected object. Default is None.

Type

float, optional

AY

The y-acceleration of the detected object. Default is None.

Type

float, optional

objID

The ID of the detected object. Default is None.

Type

int, optional

__repr__() → str

Returns a string representation of the Detection object.

__eq__(*other*) → bool

Returns True if the given Detection object is equal to this Detection object, False otherwise.

AX: float

AY: float

Height: float

VX: float

VY: float

Width: float

X: float

Y: float

confidence: float

frameID: int

label: str

objID: int

class dataManagementClasses.**TrackedObject**(*id: int, first: Detection, max_age: int = 30*)

Bases: object

A class representing a tracked object in a video.

objID

The ID of the object.

Type

int

label

The label of the object.

Type

int

futureX

A list of future X positions of the object.

Type

list

futureY

A list of future Y positions of the object.

Type

list

history

A list of Detection objects representing the history of the object.

Type

List[*Detection*]

history_X

An array of X positions of the object in the history.

Type

np.ndarray

history_Y

An array of Y positions of the object in the history.

Type

np.ndarray

history_VX_calculated

An array of calculated X velocities of the object in the history.

Type

np.ndarray

history_VY_calculated

An array of calculated Y velocities of the object in the history.

Type

np.ndarray

history_AX_calculated

An array of calculated X accelerations of the object in the history.

Type

np.ndarray

history_AY_calculated

An array of calculated Y accelerations of the object in the history.

Type

np.ndarray

isMoving

A boolean indicating whether the object is moving or not.

Type

bool

time_since_update

The time elapsed since the last update of the object.

Type

int

max_age

The maximum age of the object.

Type

int

mean

A list of mean values of the object.

Type

list

X

The X position of the object.

Type

int

Y

The Y position of the object.

Type

int

VX

The X velocity of the object.

Type

float

VY

The Y velocity of the object.

Type

float

AX

The X acceleration of the object.

Type

float

AY

The Y acceleration of the object.

Type

float

_dataset

The dataset of the object.

Type

str

__init__(*self*, *id*: int, *first*: [Detection](#), *max_age*: int = 30)

Initializes a TrackedObject instance.

__repr__(*self*) → str

Returns a string representation of the TrackedObject instance.

__hash__(*self*) → int
Returns a hash value of the TrackedObject instance.

__eq__(*self*, *other*) → bool
Returns True if the TrackedObject instance is equal to the other TrackedObject instance.

avgArea(*self*) → np.ndarray
Calculates the average area of the bounding boxes of the object.

updateAccel(*self*, *new_vx*, *old_vx*, *new_vy*, *old_vy*) → None
Calculates acceleration based on Kalman filter's velocity.

upscale_feature(*featureVector*: np.ndarray, *framewidth*: int = 1920, *frameheight*: int = 1080) → np.ndarray
Rescales normalized coordinates with the given frame sizes.

downscale_feature(*featureVector*: np.ndarray, *framewidth*: int = 1920, *frameheight*: int = 1080) → np.ndarray
Normalizes coordinates with the given frame sizes.

feature_v1(*self*) → np.ndarray | None
Extracts version 1 feature vector from history.

feature_v1_SG(*self*, *window_length*: int = 7, *polyorder*: int = 2) → np.ndarray | None
Extracts version 1SG feature vector from history.

feature_v3(*self*) → np.ndarray
Returns version 3 feature vector.

feature_v7(*self*, *history_size*: int = 30, *weights*: np.ndarray | None = None) → np.ndarray | None
Extracts feature version 7 from history.

feature_v7_SG(*self*, *history_size*: int = 30, *weights*: np.ndarray = None, *window_length*: int = 7, *polyorder*: int = 2) → np.ndarray | None
Extracts feature version 7SG from history.

feature_v8(*self*, *history_size*: int = 30) → np.ndarray | None
Extracts feature version 8 from history.

feature_v8_SG(*self*, *history_size*: int = 30, *window_length*: int = 7, *polyorder*: int = 2) → np.ndarray | None
Extracts feature version 8SG from history.

feature_v5(*self*, *n_weights*: int) → np.ndarray | None
Returns version 5 feature vector.

feature_vector(*self*, *version*: str = '1', ***kwargs*) → np.ndarray | None
Returns the corresponding feature vector version of the given version argument.

Methods

<code>avgArea()</code>	Calculate the average area of the bounding boxes of the object.
<code>downscale_feature(featureVector[, ...])</code>	Normalize coordinates with the given frame sizes.
<code>feature_v1()</code>	Extract version 1 feature vector from history.
<code>feature_v1_SG([window_length, polyorder])</code>	Extract version 1SG feature vector from history.
<code>feature_v3()</code>	Return version 3 feature vector.
<code>feature_v5(n_weights)</code>	Return version 5 feature vector.
<code>feature_v7([history_size, weights])</code>	Extract feature version 7 from history.
<code>feature_v7_SG([history_size, weights, ...])</code>	Extract feature version 7SG from history.
<code>feature_v8([history_size])</code>	Extract feature version 7 from history.
<code>feature_v8_SG([history_size, window_length, ...])</code>	Extract feature version 7SG from history.
<code>feature_vector([version])</code>	Return the corresponding feature vector version of the given version argument.
<code>update([detection, mean])</code>	Update the tracked object state with new detection.
<code>updateAccel(new_vx, old_vx, new_vy, old_vy)</code>	Calculate acceleration based on Kalman filter's velocity.
<code>update_accel([k])</code>	Calculate velocity from X,Y coordinates.
<code>update_velocity([k])</code>	Calculate velocity from X,Y coordinates.
<code>upscale_feature(featureVector[, framewidth, ...])</code>	Rescale normalized coordinates with the given frame sizes.

AX: float

AY: float

VX: float

VY: float

X: int

Y: int

avgArea() → ndarray

Calculate the average area of the bounding boxes of the object.

Returns

The average area of the bounding boxes of the object.

Return type

float

static downscale_feature(*featureVector: ndarray, framewidth: int = 1920, frameheight: int = 1080*) → ndarray

Normalize coordinates with the given frame sizes. Normalization is done by dividing the coordinates with the frame sizes, but the X coordinates are divided with the ratio of the frame width and height.

Parameters

- **featureVector** (ndarray) – Feature vector of the track
- **framewidth** (int, optional) – Frame width, by default 1920
- **frameheight** (int, optional) – Frame height, by default 1080

Returns

Normalized feature vector

Return type

ndarray

feature_v1() → ndarray | None

Extract version 1 feature vector from history.

Returns

Feature vector

Return type

ndarray

feature_v1_SG(*window_length: int = 7, polyorder: int = 2*) → ndarray | None

Extract version 1SG feature vector from history.

Returns

Feature vector

Return type

ndarray

feature_v3() → ndarray

Return version 3 feature vector.

Returns

Feature vector version 3

Return type

ndarray

feature_v5(*n_weights: int*) → ndarray | None

Return version 5 feature vector.

This feature vector consists of the first and last detection's X and Y coordinates. The *n_weight* number controls the number of inserted coordinates between the first and the last detection's coordinates.

Parameters

n_weights (*int*) – The number of inserted coordinates between the first and the last detection's coordinates.

Returns

A numpy ndarray or None if the number of inserted coordinates is less than 1.

Return type

Optional[np.ndarray]

feature_v7(*history_size: int = 30, weights: ndarray | None = None*) → ndarray | None

Extract feature version 7 from history.

Parameters

history_size (*int, optional*) – Size of the history that should be used for feature vector creation, by default 30

Returns

Feature vector or None, if history is not yet the size of *history_size*, then return None

Return type

Optional[np.ndarray]

feature_v7_SG(*history_size: int = 30, weights: ndarray | None = None, window_length: int = 7, polyorder: int = 2*) → ndarray | None

Extract feature version 7SG from history.

Parameters

- **history_size** (*int, optional*) – Size of the history that should be used for feature vector creation, by default 30
- **window_length** (*int, optional*) – The size of the Savitzky Golay filter, by default 7
- **polyorder** (*int, optional*) – The polynomial order of the Savitzky Golay filter, by default 7

Returns

Feature vector or None, if history is not yet the size of history_size, then return None

Return type

Optional[np.ndarray]

feature_v8(*history_size: int = 30*) → ndarray | None

Extract feature version 7 from history.

Parameters

history_size (*int, optional*) – Size of the history that should be used for feature vector creation, by default 30

Returns

Feature vector or None, if history is not yet the size of history_size, then return None

Return type

Optional[np.ndarray]

feature_v8_SG(*history_size: int = 30, window_length: int = 7, polyorder: int = 2*) → ndarray | None

Extract feature version 7SG from history.

Parameters

- **history_size** (*int, optional*) – Size of the history that should be used for feature vector creation, by default 30
- **window_length** (*int, optional*) – The size of the Savitzky Golay filter, by default 7
- **polyorder** (*int, optional*) – The polynomial order of the Savitzky Golay filter, by default 7

Returns

Feature vector or None, if history is not yet the size of history_size, then return None

Return type

Optional[np.ndarray]

feature_vector(*version: str = '1', **kwargs*) → ndarray | None

Return the corresponding feature vector version of the given version argument. Pass additional keyword arguments.

Parameters

version (*str, optional*) – Version string of the feature vector, by default '1'

Returns

Feature vector

Return type
np.ndarray

futureX: list
futureY: list
history: List[[Detection](#)]
history_AX_calculated: ndarray
history_AY_calculated: ndarray
history_VX_calculated: ndarray
history_VY_calculated: ndarray
history_X: ndarray
history_Y: ndarray
isMoving: bool
label: int
max_age: int
mean: list
objID: int
time_since_update: int

update(*detection*: [Detection](#) | None = None, *mean*=None)

Update the tracked object state with new detection.

Parameters

- **detection** ([Detection](#), *optional*) – New Detection from yolo, by default None
- **mean** (List, *optional*) – Object values calculated by Kalman filter, by default None

updateAccel(*new_vx*, *old_vx*, *new_vy*, *old_vy*) → None

Calculate acceleration based on Kalman filter's velocity.

Calculates acceleration using the formula: $(\text{new_v} - \text{old_v}) / (\text{new_time} - \text{old_time})$ where the time between the two detections is (new_time-old_time).

Parameters

- **new_vx** (*float*) – The new velocity of x.
- **old_vx** (*float*) – The old velocity of x from the previous detection.
- **new_vy** (*float*) – The new velocity of y.
- **old_vy** (*float*) – The old velocity of y from the previous detection.

Return type
None

update_accel(*k*: int = 2)

Calculate velocity from X,Y coordinates.

update_velocity(*k: int = 10*)

Calculate velocity from X,Y coordinates.

static upscale_feature(*featureVector: ndarray, framewidth: int = 1920, frameheight: int = 1080*) →
ndarray

Rescale normalized coordinates with the given frame sizes.

Parameters

- **featureVector** (*ndarray*) – Feature vector of the track
- **framewidth** (*int, optional*) – Frame width, by default 1920
- **frameheight** (*int, optional*) – Frame height, by default 1080

Returns

Upscaled feature vector

Return type

ndarray

dataManagementClasses.detectionFactory(*objID, frameNum, label, confidence, x, y, width, height, vx, vy, ax, ay*)

Create a Detection object.

Parameters

- **objID** (*int*) – Object ID, to which object the Detection belongs to.
- **frameNum** (*int*) – Frame number when the Detection occurred.
- **label** (*str*) – Label of the object, etc: car, person...
- **confidence** (*float*) – Confidence number, of how confident the neural network is in the detection.
- **x** (*float*) – X coordinate of the object.
- **y** (*float*) – Y coordinate of the object.
- **width** (*float*) – Width of the bounding box of the object.
- **height** (*float*) – Height of the bounding box of the object.
- **vx** (*float*) – Velocity on the X axis.
- **vy** (*float*) – Velocity on the Y axis.
- **ax** (*float*) – Acceleration on the X axis.
- **ay** (*float*) – Acceleration on the Y axis.

Returns

The Detection object, which is to be returned.

Return type

Detection

dataManagementClasses.detectionParser(*rawDetectionData*)

Convert raw detection data loaded from database to class Detection and numpy arrays.

Parameters

rawDetectionData (*list*) – Raw values loaded from database.

Returns

Tuple containing detections, and all the history numpy arrays.

Return type

tuple

`dataManagementClasses.findEnterAndExitPoints(path2db: str)`

Extracts only the first and the last detections of tracked objects.

Parameters

path2db (*str*) – Path to the database file.

Returns

A tuple containing two lists: enterDetections and exitDetections. enterDetections : list

List of first detections of objects.

exitDetections

[list] List of last detections of objects.

Return type

tuple

`dataManagementClasses.insert_weights_into_feature_vector(start: int, stop: int, n_weights: int, X: ndarray, Y: ndarray, insert_idx: int, feature_vector: ndarray) → ndarray`

Insert coordinates into feature vector starting from the start_insert_idx index.

Parameters

- **start** (*int*) – First index of inserted coordinates.
- **stop** (*int*) – Stop index of coordinate vectors, which will not be inserted, this is the open end of the limits.
- **n_weights** (*int*) – Number of weights to be inserted.
- **X** (*np.ndarray*) – X coordinate array.
- **Y** (*np.ndarray*) – Y coordinate array.
- **insert_idx** (*int*) – The index where the coordinates will be inserted into the feature vector.
- **feature_vector** (*np.ndarray*) – The feature vector to insert the coordinates into.

Returns

The updated feature vector with the inserted coordinates.

Return type

np.ndarray

`dataManagementClasses.parseRawObject2TrackedObject(rawObjID: int, path2db: str)`

Takes an objID and the path to the database, then returns a trackedObject object if detections can be assigned to the object.

Parameters

- **rawObjID** (*int*) – ID of an object.
- **path2db** (*str*) – Path to database.

Returns

TrackedObject object from dataManagement class, if no detections can be assigned to it, then returns False.

Return type

trackedObject or bool

dataManagementClasses.**preprocess_database_data**(*path2db: str*)

Preprocesses database data (detections) by assigning detections to objects.

Parameters

path2db (*str*) – Path to database file.

Returns

List of object tracks.

Return type

list

dataManagementClasses.**preprocess_database_data_multiprocessed**(*path2db: str, n_jobs=None*)

Preprocesses database data (detections) by assigning detections to objects.

Parameters

- **path2db** (*str*) – Path to database file.
- **n_jobs** (*int, optional*) – Number of parallel jobs to run, by default None.

Returns

List of object tracks.

Return type

List

dataManagementClasses.**trackedObjectFactory**(*detections: tuple*)

Create a TrackedObject object from a list of detections.

Parameters

detections (*List*) – A list of Detection objects.

Returns

The TrackedObject object created from the list of detections.

Return type

TrackedObject

dataManagementClasses.**tracks2joblib**(*path2db: str, n_jobs: int = 18*)

Extract tracks from database and save them in a joblib object.

Parameters

- **path2db** (*str*) – Path to database.
- **n_jobs** (*int, optional*) – Number of parallel jobs to run, by default 18.

Return type

None

3.7 database_metadata module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

`database_metadata.coordinates2heatmap(path2db)`

Create heatmap from detection data. Every object has its own coloring.

Parameters

path2db (*str*) – Path to database file.

`database_metadata.main()`

`database_metadata.printConfig(path2db)`

`database_metadata.warn(*arg, **args)`

3.8 detect module

`detect.detect(save_img=False)`

3.9 evaluate module

3.10 examine_tracks module

3.11 masker module

`masker.main()`

`masker.masker(img: ndarray)`

3.12 traffic_statistics module

Statistical analysis of datasets generated by detector.py Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

`traffic_statistics.extractHourlyData(dataset: List[TrackedObject])`

Extracts hourly data from a list of TrackedObject instances.

Parameters

dataset (*List[TrackedObject]*) – A list of TrackedObject instances.

Returns

hourlyData – A 2D numpy array of hourly data, where each row represents an hour and each column represents a TrackedObject instance.

Return type

numpy.ndarray

`traffic_statistics.hourlyHistogram(hourlyLabels: ndarray, classes: list, output: str) → None`

Plot a histogram of hourly traffic data and save the figure to a file.

Parameters

- **hourlyLabels** (*np.ndarray*) – An array of hourly traffic data.
- **classes** (*list*) – A list of class labels for the traffic data.
- **output** (*str*) – The output directory to save the histogram figure.

Return type

None

`traffic_statistics.hourlyStatisticsModule(args)`

Compute hourly traffic statistics.

Parameters

args (*argparse.Namespace*) – Namespace object containing the command line arguments.

Return type

None

Notes

This function computes hourly traffic statistics by clustering tracked objects and generating histograms of the resulting clusters. The resulting statistics are saved to disk in various formats.

`traffic_statistics.hourlyTable(paths, hourlyTracks, hourlyLabels, classes, output)`

Generate hourly statistics table and heatmaps.

Parameters

- **paths** (*list of str*) – List of file paths.
- **hourlyTracks** (*list of numpy.ndarray*) – List of numpy arrays containing tracks for each hour.
- **hourlyLabels** (*list of numpy.ndarray*) – List of numpy arrays containing labels for each hour.
- **classes** (*list of int*) – List of classes to count.
- **output** (*str*) – Output directory path.

Return type

None

Notes

This function generates a pandas DataFrame containing hourly counts of vehicles for each class, and saves it to an Excel file. It also generates three heatmaps: one with absolute counts, one normalized by row, and one normalized by column. These heatmaps are saved as PNG files.

`traffic_statistics.main()`

`traffic_statistics.printSamplesToExcel(tracks, output, n_samples=10)`

Write a sample of the tracks to an Excel file.

Parameters

- **tracks** (*list*) – A list of Track objects.
- **output** (*str*) – The path to the output directory.
- **n_samples** (*int, optional*) – The number of samples to write to the Excel file. Default is 10.

Return type

None

`traffic_statistics.trafficHistogram(dataset: List[TrackedObject], labels: ndarray, output: str, bg_img: str)`

Generate a histogram and heatmap of the traffic dataset.

Parameters

- **dataset** (*List[TrackedObject]*) – A list of TrackedObject instances representing the traffic dataset.
- **labels** (*np.ndarray*) – An array of labels for the dataset.
- **output** (*str*) – The output directory for the generated figures.
- **bg_img** (*str*) – The path to the background image to use for the heatmap.

Return type

None

Notes

This function generates two figures: 1. A histogram of the traffic dataset, with each class represented by a different color. 2. A heatmap of the traffic dataset, with each class represented by a different arrow.

The figures are saved to the specified output directory, if provided, and displayed on screen.

`traffic_statistics.trafficHistogramModule(args)`

Generate a traffic histogram from a dataset of vehicle trajectories.

Parameters

args (*argparse.Namespace*) – Command-line arguments parsed by argparse.

Return type

None

Notes

This function loads a dataset of vehicle trajectories, filters the trajectories if specified, extracts 4D feature vectors from the trajectories, clusters the feature vectors using OPTICS, and generates a traffic histogram from the resulting labels and filtered trajectories.

`traffic_statistics.trafficStatisticTable(args)`

3.13 train module

`train.train(hyp, opt, device, tb_writer=None)`

3.14 utility package

3.14.1 Submodules

3.14.2 utility.databaseLoader module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

`utility.databaseLoader.loadDetections(path2db: str) → list`

`utility.databaseLoader.loadDetectionsOfObject(path2db: str, objID: int) → list`

`utility.databaseLoader.loadMetadata(path2db: str) → list`

`utility.databaseLoader.loadObjects(path2db: str) → list`

Load objects from database. A raw object entry in the database looks like this: [objID, label]

Parameters

path2db (*str*) – Path to database file

Returns

list of raw object entries

Return type

list

`utility.databaseLoader.loadPredictions(path2db: str) → list`

`utility.databaseLoader.loadRegression(path2db: str) → list`

`utility.databaseLoader.queryLastObjID(path2db) → int`

3.14.3 utility.databaseLogger module

Predicting trajectories of objects Copyright (C) 2022 Bence Peter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Contact email: ecneb2000@gmail.com

`utility.databaseLogger.bbox2float(img0: ndarray, x: int, y: int, w: int, h: int, vx: float, vy: float, ax: float, ay: float, vx_c: float, vy_c: float, ax_c: float, ay_c: float)`

Downscale bounding box values to with given image's width and height using aspect ratio.

Parameters

- **img0** (*numpy.ndarray*) – 3D numpy array containing BGR values shape(height, width, 3)
- **x** (*int*) – coordinate x
- **y** (*int*) – coordinate y
- **w** (*int*) – width of the bounding box
- **h** (*int*) – height of the bounding box
- **vx** (*float*) – velocity of dimesion x
- **vy** (*float*) – velocity of dimesion y
- **ax** (*float*) – acceleration of dimension x

- **ay** (*float*) – acceleration of dimension y
- **vx_c** (*float*) – differentiation of x
- **vy_c** (*float*) – differentiation of y
- **ax_c** (*float*) – differentiation if vx_c
- **ay_c** (*float*) – differentiation of vy_c

Returns

tuple of downscaled values

Return type

tuple

`utility.databaseLogger.closeConnection(conn: Connection)`

Closes connection to the database.

Parameters

conn (*sqlite3.Connection*) – Database connection, that is going to be closed.

`utility.databaseLogger.detectionExists(conn: Connection, objID: int, frameNumber: int)`

Check if entry already exists in database. No multiple detections can occur in a single frame, that have the same objID.

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **objID** (*int*) – ID of tracked object.
- **frameNum** (*int*) – Frame in the detection happened.

Returns

True if entry already exists.

Return type

bool

`utility.databaseLogger.downdscalecoord(scalenum: float, coord: float, a: float)`

Downscale coordinate number with given scale number and aspect ratio.

Parameters

- **scalenum** (*float*) – number to scale the coordinate number with
- **coord** (*float*) – coordinate
- **a** (*float*) – aspect ratio

Returns

downscaled value

Return type

float

`utility.databaseLogger.getConnection(db_path: str) → Connection`

Creates connection to the database.

Parameters

db_name (*str*) – path to the database file.

Returns

sqlite3 Connection object.

Return type

sqlite3.Connection

`utility.databaseLogger.getLatestFrame(conn: Connection)`

Gets the number of last frame, when the last sessions logging was stopped.

Parameters

conn (*sqlite3.Connection*) – Connection to the database.

`utility.databaseLogger.init_db(outpath: str)`

Initialize SQLite3 database. Input video_name which is the DIR name. DB_name will be the name of the database. If directory does not exists, then create one. Creates database from given schema.

Parameters

- **video_name** (*str*) – The video source's name is the dir name.
- **db_name** (*str*) – Database name.

`utility.databaseLogger.logBuffer(conn: Connection, frame: ndarray, buffer: list)`

Log buffer to the database after main loop is ended.

Parameters

- **conn** (*sqlite3.Connection*) – connection object to the database
- **frame** (*np.ndarray*) – frame
- **buffer** (*list*) – the buffered data to log,
- **list** *[[objID (in this sepcific scenario the buffer looks like this) – obj.history[-1].frameID, obj.history[-1].confidence obj.X, obj.Y, obj.history[-1].Width, obj.history[-1].Height, obj.VX, obj.VY, obj.AX, obj.AY, obj.futureX, obj.futureY]]*

:param

[obj.history[-1].frameID,] obj.history[-1].confidence obj.X, obj.Y, obj.history[-1].Width, obj.history[-1].Height, obj.VX, obj.VY, obj.AX, obj.AY, obj.futureX, obj.futureY]

`utility.databaseLogger.logBufferSpeedy(conn: Connection, frame: ndarray, buffer: list)`

Log buffer to the database after main loop is ended. Faster than logBuffer(), thanks to executemany() function.

Parameters

- **conn** (*sqlite3.Connection*) – connection object to the database
- **frame** (*np.ndarray*) – frame
- **buffer** (*list*) – the buffered data to log,
- **list** *[[objID (in this sepcific scenario the buffer looks like this) – obj.history[-1].frameID, obj.history[-1].confidence obj.X, obj.Y, obj.history[-1].Width, obj.history[-1].Height, obj.VX, obj.VY, obj.AX, obj.AY, obj.futureX, obj.futureY]]*

:param

[obj.history[-1].frameID,] obj.history[-1].confidence obj.X, obj.Y, obj.history[-1].Width, obj.history[-1].Height, obj.VX, obj.VY, obj.AX, obj.AY, obj.futureX, obj.futureY]

`utility.databaseLogger.logDetection(conn: Connection, img0: ndarray, objID: int, frameNum: int, confidence: float, x_coord: int, y_coord: int, width: int, height: int, x_vel: float, y_vel: float, x_acc: float, y_acc: float, x_vel_calc: float, y_vel_calc: float, x_acc_calc: float, y_acc_calc: float)`

Logging detections to the database. Downscale bbox coordinates to floats. Insert entry to database if there is no similar entry found.

Parameters

- **conn** (*sqlite3.Connection*) – Database connection, where data will be logged.
- **img0** (*numpy matrix*) – actual image where detections happened.
- **objID** (*int*) – track id of detection
- **frameNum** (*int*) – number of actual frame
- **label** (*str*) – The label of the detected object.
- **confidence** (*float*) – Confidence of the detection.
- **x_coord** (*int*) – X center coord of bbox
- **y_coord** (*int*) – Y center coord of bbox
- **width** (*int*) – Width of bbox.
- **height** (*int*) – Height of bbox.

```
utility.databaseLogger.logMetaData(conn: Connection, historyDepth: int, futureDepth: int, yoloVersion:  
                                   str, device: str, imsz: int, stride: int, conf_thres: float, iou_thres:  
                                   float)
```

Log environment data to the database.

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **historyDepth** (*int*) – Length of stored detection history in the memory
- **futureDepth** (*int*) – Length of prediction vector.

```
utility.databaseLogger.logObject(conn: Connection, objID: int, label: str)
```

Log new object to the database.

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **objID** (*int*) – Unique id of the new track.
- **label** (*str*) – The type of an object, ex. car, person, etc...

```
utility.databaseLogger.logPredictions(conn: Connection, img0: ndarray, objID: int, frameNumber: int,  
                                       x: ndarray, y: ndarray)
```

Log predictions to database, in format (objID, frameNum, idx, x, y)

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **x** (*int*) – X coordinate prediction
- **y** (*int*) – Y coordinate prediction

```
utility.databaseLogger.logRegression(conn: Connection, linearFunction: str, polynomFunction: str,  
                                     polynomDegree: int, trainingPoints: int)
```

Log regression function data to database.

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **linearFunction** (*str*) – Name of the linear regression function.
- **polynomFunction** (*str*) – Name of the polynomial regression function.
- **polynomDegree** (*int*) – Degree of polynomial features.
- **trainingPoints** (*int*) – Number of training points.

`utility.databaseLogger.objExists(conn: Connection, objID: int, label: str) → bool`

Check if entry already exists in database. No multiple detections can occur in a single frame, that have the same objID.

Parameters

- **conn** (*sqlite3.Connection*) – Connection to the database.
- **objID** (*int*) – ID of tracked object.
- **frameNum** (*int*) – Frame in the detection happened.

Returns

True if entry already exists.

Return type

bool

`utility.databaseLogger.prediction2float(img0: ndarray, x: float, y: float)`

Downscale prediction coordinates to floats.

Parameters

- **img0** (*numpy.ndarray*) – Actual frame when prediction occurred.
- **x** (*int*) – X coordinate
- **y** (*int*) – Y coordinate

3.14.4 utility.dataset module

`utility.dataset.downscale_TrackedObjects(trackedObjects: list, img: ndarray)`

Normalize the values of the detections with the given np.ndarray image.

Parameters

- **trackedObjects** (*list[TrackedObject]*) – list of tracked objects
- **img** (*np.ndarray*) – image to downscale from

`utility.dataset.loadDatasetMultiprocessed(path, n_jobs=-1)`

`utility.dataset.loadDatasetMultiprocessedCallback(result)`

`utility.dataset.loadDatasetsFromDirectory(path: str | Path) → ndarray | bool`

Load all datasets from a directory.

Parameters

path (*Union[str, Path]*) – Path to directory containing datasets.

Returns

Numpy array containing all datasets, or False if path is not a directory.

Return type

Union[np.ndarray, bool]

`utility.dataset.load_dataset(path2dataset: str | List[str] | Path) → ndarray`

Load a dataset from a file or a directory.

Parameters`path2dataset` (Union[str, List[str], Path]) –**Returns**

Numpy array containing the dataset.

Return type

np.ndarray

Raises**IOError** – Wrong file type.`utility.dataset.load_dataset_with_labels(path)``utility.dataset.mergeDatasets(datasets: ndarray)`

Merge datasets into one.

Parameters`datasets` (ndarray) – List of datasets to merge. shape(n, m) where n is the number of datasets and m is the number of tracks in the dataset.**Returns**

Merged dataset.

Return type

ndarray

`utility.dataset.save_trajectories(trajectories: List | ndarray, output: str | Path, classifier: str = 'SVM', name: str = 'trajectories') → List[str]`

Save trajectories to a file.

Parameters

- `trajectories` (Union[List, np.ndarray]) – Trajectories to save.
- `output` (Union[str, Path]) – Output directory path.
- `classifier` (str, optional) – Name of classifier, by default “SVM”
- `name` (str, optional) – Additional name to identify file, by default “trajectories”

Returns

List of saved file paths.

Return type

List[str]

3.14.5 utility.featurevector module

class `utility.featurevector.FeatureVector`

Bases: `object`

Class representing a feature vector.

Methods

<code>__call__([version])</code>	Call self as a function.
<code>factory_1(trackedObjects, labels, pooled_labels)</code>	Generate feature vectors from the histories of trajectories.
<code>factory_1_SG(trackedObjects, labels, ...[, ...])</code>	Generate feature vectors from the histories of trajectories.
<code>factory_7(trackedObjects, labels, ...[, weights])</code>	Generate feature vectors from the histories of trajectories.
<code>factory_7_SG(trackedObjects, labels, ...[, ...])</code>	Factory method for computing feature vectors using 7th order Savitzky-Golay filter.
<code>factory_8(trackedObjects, labels, ...[, weight])</code>	Computes the feature vectors for a list of tracked objects.

factory_10

factory_8_SG

factory_9

static `factory_1(trackedObjects: List, labels: ndarray, pooled_labels: ndarray, k: int = 6, up_until: float = 1) → Tuple[ndarray, ndarray, ndarray, ndarray]`

Generate feature vectors from the histories of trajectories. Divide the trajectory into k parts, then make k number of feature vectors.

Parameters

- **trackedObjects** (*List*) – List of trajectories.
- **labels** (*np.ndarray*) – List of corresponding labels.
- **pooled_labels** (*np.ndarray*) – List of corresponding pooled labels.
- **k** (*int, optional*) – Number of subtrajectories, by default 6

Returns

Tuple containing the generated feature vectors, corresponding labels, corresponding pooled labels, corresponding metadata.

Return type

`Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]`

static `factory_10(trackedObjects: List, labels: ndarray, pooled_labels: ndarray, max_stride: int = 30, window_length: int = 7, polyorder: int = 2) → Tuple[ndarray, ndarray, ndarray, ndarray]`

static `factory_1_SG(trackedObjects: List, labels: ndarray, pooled_labels: ndarray, k: int = 6, up_until: float = 1, window_length: int = 7, polyorder: int = 2) → Tuple[ndarray, ndarray, ndarray, ndarray]`

Generate feature vectors from the histories of trajectories. Divide the trajectory into k parts, then make k number of feature vectors. Apply savgol filter on the coordinates and velocities of the trajectory part.

Parameters

- **trackedObjects** (*List*) – List of trajectories.
- **labels** (*np.ndarray*) – List of corresponding labels.
- **pooled_labels** (*np.ndarray*) – List of corresponding pooled labels.
- **k** (*int, optional*) – Number of subtrajectories, by default 6
- **window_length** (*int, optional*) – Window of savgol filter, by default 7
- **polyorder** (*int, optional*) – Degree of polynom used in savgol filter, by default 2

Returns

Tuple containing the generated feature vectors, corresponding labels, corresponding pooled labels, corresponding metadata.

Return type

Tuple[*np.ndarray*, *np.ndarray*, *np.ndarray*, *np.ndarray*]

static factory_7(*trackedObjects: List*, *labels: ndarray*, *pooled_labels: ndarray*, *max_stride: int*, *weights: ndarray | None = array([1, 1, 100, 100, 2, 2, 200, 200], dtype=float32)*) → Tuple[*ndarray*, *ndarray*, *ndarray*, *ndarray*]

Generate feature vectors from the histories of trajectories. Make feature vectors from the whole trajectory, with a stride of *max_stride*.

Parameters

- **trackedObjects** (*List*) – Tracked objects.
- **labels** (*np.ndarray*) – Labels.
- **pooled_labels** (*np.ndarray*) – Pooled labels.
- **max_stride** (*int*) – Maximum stride length.
- **weights** (*Optional[np.ndarray], optional*) – Weight vector, by default None

Returns

Feature vectors, corresponding labels and pooled labels, and metadata.

Return type

Tuple[*np.ndarray*, *np.ndarray*, *np.ndarray*, *np.ndarray*]

static factory_7_SG(*trackedObjects: List*, *labels: ndarray*, *pooled_labels: ndarray*, *max_stride: int*, *weights: ndarray | None = array([1, 1, 100, 100, 2, 2, 200, 200], dtype=float32)*, *window_length: int = 7*, *polyorder: int = 2*) → Tuple[*ndarray*, *ndarray*, *ndarray*, *ndarray*]

Factory method for computing feature vectors using 7th order Savitzky-Golay filter.

Parameters

- **trackedObjects** (*List*) – List of tracked objects.
- **labels** (*np.ndarray*) – Array of labels.
- **pooled_labels** (*np.ndarray*) – Array of pooled labels.
- **max_stride** (*int*) – Maximum stride.

- **weights** (*Optional* [*np.ndarray*], *optional*) – Array of weights for each feature, by default `np.array([1, 1, 100, 100, 2, 2, 200, 200], dtype=np.float32)`
- **window_length** (*int*, *optional*) – Window length for Savitzky-Golay filter, by default 7
- **polyorder** (*int*, *optional*) – Polynomial order for Savitzky-Golay filter, by default 2

Returns

Tuple containing feature vectors, labels, pooled labels, and metadata.

Return type

Tuple[*np.ndarray*, *np.ndarray*, *np.ndarray*, *np.ndarray*]

static factory_8(*trackedObjects*: *List*, *labels*: *ndarray*, *pooled_labels*: *ndarray*, *max_stride*: *int*, *weight*: *float* = 0.8) → Tuple[*ndarray*, *ndarray*, *ndarray*, *ndarray*]

Computes the feature vectors for a list of tracked objects.

Parameters

- **trackedObjects** (*List*) – List of tracked objects.
- **labels** (*np.ndarray*) – Array of labels for each tracked object.
- **pooled_labels** (*np.ndarray*) – Array of pooled labels for each tracked object.
- **max_stride** (*int*) – Maximum stride to be used in the feature vector computation.
- **weight** (*float*, *optional*) – Weight to be used in the feature vector computation, by default 0.8.

Returns

Tuple containing the computed feature vectors, labels, pooled labels, and metadata.

Return type

Tuple[*np.ndarray*, *np.ndarray*, *np.ndarray*, *np.ndarray*]

static factory_8_SG(*trackedObjects*: *List*, *labels*: *ndarray*, *pooled_labels*: *ndarray*, *max_stride*: *int*, *weight*: *float* = 0.8, *window_length*: *int* = 7, *polyorder*: *int* = 2) → Tuple[*ndarray*, *ndarray*, *ndarray*, *ndarray*]

static factory_9(*trackedObjects*: *List*, *labels*: *ndarray*, *pooled_labels*: *ndarray*, *max_stride*: *int* = 30) → Tuple[*ndarray*, *ndarray*, *ndarray*, *ndarray*]

3.14.6 utility.general module

`utility.general.checkDir(path2db)`

Check for dir of given database, to be able to save plots.

Parameters

path2db (*str*) – Path to database.

`utility.general.diff(x_1: float, x_2: float, dt: float) → float`

Differentiate with function $x_{(i+1)} - x_i / dt$

Parameters

- **x_1** (*float*) – x_i
- **x_2** (*float*) – $x_{(i+1)}$
- **dt** (*float*) – dt

Returns

dx

Return type

float

`utility.general.diffmap(a: array, t: array, k: int)`Differentiate an array *a* with time vector *t*, and *k* i+k in the function $x_{(i+k)} - x_i / t_{(i+k)} - t_i$ **Parameters**

- **a** (*np.array*) – array of values to differentiate
- **t** (*np.array*) – times to differentiate with
- **k** (*int*) – stepsize

Returns

Return dX and t timestamps of dX with the logic dx_i, t_i+k

Return type

np.array, np.array

`utility.general.dt(t1: float, t2: float) → float`

Calculate dt

Parameters

- **t1** (*float*) – t_i
- **t2** (*float*) – t_(i+1)

Returns

dt

Return type

float

`utility.general.strfy_dict_params(params: dict)`

Stringify params stored in dictionaries.

Parameters**params** (*dict*) – Dict storing the params.**Returns**

Stringified params returned in the format “_param1_value1_param2_value2”.

Return type

str

3.14.7 utility.logging module

`utility.logging.init_logger(name: str | None = None, filename: str | None = None) → Logger`

Initialize logger.

Parameters

- **name** (*Optional[str]*, *optional*) – Name of logger, by default None
- **filename** (*Optional[str]*, *optional*) – Name of log file, by default None

Returns

Logger instance

Return type

Logger

3.14.8 utility.models module`utility.models.load_model(path2model: str) → Any`

Load model from disk.

Parameters**path2model** (*str*) – Path to model**Returns**

Model object

Return type

Any

`utility.models.save_model(savedir: str, classifier_type: str, model, version: str | None = None) → bool`

Save ML Model.

Parameters

- **savedir** (*str*) – Path to directory where the model should be placed.
- **classifier_type** (*str*) – Type of classifier, eg. KNN, DT, SVM, etc...
- **model** (`OneVSRestClassifierExtended`) – The model object itself.
- **version** (*Optional[str]*, *optional*) – Version string, eg. 1, 7, 8SG, etc..., by default None

Returns

Return True if saving was successful, False otherwise.

Return type

bool

3.14.9 utility.plots module`utility.plots.cvCoord2npCoord(Y: ndarray) → ndarray`

Convert OpenCV coordinates to numpy coordinates.

Parameters**Y** (*np.ndarray*) – OpenCV coordinates**Returns**

Numpy coordinates

Return type

np.ndarray

`utility.plots.makeColormap(path2db)`

Make colormap based on number of objects logged in the database. This colormap vector will be input to matplotlib scatter plot.

Parameters**path2db** (*str*) – Path to database**Returns**

list of color gradient vectors (R,G,B)

Return type

colormap

`utility.plots.plot_misclassified(misclassifiedTracks: List, output: str | None = None)`

Plot the misclassified tracks.

Parameters

- **misclassifiedTracks** (*list*) – A list of misclassified tracks.
- **output** (*str, optional*) – The output directory to save the plot.

Return type

None

Examples

```
>>> plot_misclassified(misclassifiedTracks, output="output_dir")
```

`utility.plots.plot_misclassified_feature_vectors(misclassifiedFV: ndarray, output: str | None = None, background: str | None = None, classifier: str = 'SVM')`

Plot the misclassified feature vectors.

Parameters

- **misclassifiedFV** (*np.ndarray*) – The misclassified feature vectors.
- **output** (*str, optional*) – The output directory to save the plot, by default None.
- **background** (*str, optional*) – The path to the background image, by default None.
- **classifier** (*str, optional*) – The name of the classifier, by default “SVM”.

Returns

The function only generates and displays the plot.

Return type

None

`utility.plots.plot_one_cluster(cluster_center: ndarray, im: ndarray, color: Tuple[int, int, int] = (0, 0, 255), radius: int = 3, line_thickness: int = 2)`

Draw one cluster.

Parameters

- **cluster_center** (*np.ndarray*) – Center coordinates of the predicted cluster.
- **color** (*Tuple[int, int, int], optional*) – Color of the cluster, by default (0, 0, 255).
- **line_thickness** (*int, optional*) – Thickness of the line, by default 2.

Notes

This function is used to draw one cluster on the image.

Examples

```
>>> draw_one_cluster(cluster_center)
```

```
utility.plots.plot_one_prediction(bbox: ndarray, cluster_center: ndarray, im: ndarray, color: Tuple[int,
int, int] = (0, 0, 255), line_thickness: int = 2)
```

Draw one prediction.

Parameters

- **bbox** (*np.ndarray*) – Bounding box.
- **cluster_center** (*np.ndarray*) – Center coordinates of the predicted cluster.
- **im** (*np.ndarray*) – Image to draw on.

Notes

This function is used to draw one prediction on the image.

Examples

```
>>> draw_one_prediction(bbox, cluster_center, im)
```

```
utility.plots.savePlot(fig: Figure, name: str)
```

3.14.10 utility.preprocessing module

```
utility.preprocessing.euclidean_distance(q1: float, p1: float, q2: float, p2: float)
```

Calculate euclidean distance of 2D vectors.

Parameters

- **q1** (*float*) – First element of vector1.
- **p1** (*float*) – First element of vector2.
- **q2** (*float*) – Second element of vector1.
- **p2** (*float*) – Second element of vector2.

Returns

euclidean distance of the two vectors.

Return type

float

```
utility.preprocessing.filter_by_class(trackedObjects: list, label='car')
```

Only return objects with given label.

Parameters

- **trackedObjects** (*list*) – list of tracked objects
- **label** (*str*, *optional*) – label of object. Defaults to “car”.

Returns

list of “label” objects

Return type

list

`utility.preprocessing.filter_out_edge_detections(trackedObjects: list, threshold: float)`

Filter out objects, that enter and exit detections coordinates are in the threshold value.

Parameters

- **trackedObjects** (*list*) – list of object trackings
- **threshold** (*float*) – objects only under this value will be returned

Returns

filtered list of tracks

Return type

list

`utility.preprocessing.filter_out_false_positive_detections_by_enter_exit_distance(trackedObjects: list, threshold: float)`

This function is to filter out false positive detections, that enter and exit points are closer, than the threshold value given in the arguments.

Parameters

- **trackedObjects** (*list*) – list of object trackings
- **threshold** (*float*) – if enter and exit points are closer than this value, they are excluded from the return list

Returns

list of returned filtered tracks

Return type

list

`utility.preprocessing.filter_out_noise_trajectories(trackedObjects: list, distance: float = 0.05)`

`utility.preprocessing.filter_trajectories(trackedObjects: list, threshold: float = 0.7, enter_exit_dist: float = 0.4, detectionDistanceFiltering: bool = True, detDist: float = 0.05)`

Run filtering process on trajectory dataset.

Parameters

- **trackedObjects** (*list*) – Trajectory dataset
- **threshold** (*float*) – threshold for min max search

Returns

filtered trajectories without noise

Return type

denoised_trajectories

`utility.preprocessing.is_noise(trackedObject, threshold: float = 0.1)`

`utility.preprocessing.search_min_max_coordinates(trackedObjects: list)`

Search for minimum and maximum of x,y coordinates, to find the edges of the range of interest.

Parameters

trackedObjects (*list*) – list of trajectory objects

Returns

tuple consisting of min x,y and max x,y coordinates

Return type

tuple

`utility.preprocessing.shuffle_data(trackedObjects: list) → list`

3.14.11 utility.training module

`utility.training.iter_minibatches(X: ndarray, y: ndarray, batch_size: int)`

Generate minibatches for training.

Parameters

- **X** (*np.ndarray*) – Feature vectors shape(n_samples, n_features)
- **y** (*np.ndarray*) – Labels of vectors shape(n_samples,)

3.14.12 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

C

- [classification](#), 16
- [classifier](#), 28
- [clustering](#), 34

d

- [database_metadata](#), 59
- [dataManagementClasses](#), 46
- [detect](#), 59
- [DetectionPipeline](#), 9

e

- [evaluate](#), 59

m

- [masker](#), 59

t

- [traffic_statistics](#), 60
- [train](#), 62

U

- [utility](#), 77
- [utility.databaseLoader](#), 62
- [utility.databaseLogger](#), 63
- [utility.dataset](#), 67
- [utility.featurevector](#), 69
- [utility.general](#), 71
- [utility.logging](#), 72
- [utility.models](#), 73
- [utility.plots](#), 73
- [utility.preprocessing](#), 75
- [utility.training](#), 77

Symbols

__eq__() (*dataManagementClasses.Detection* method), 48
__eq__() (*dataManagementClasses.TrackedObject* method), 51
__hash__() (*dataManagementClasses.TrackedObject* method), 50
__init__() (*dataManagementClasses.TrackedObject* method), 50
__repr__() (*dataManagementClasses.Detection* method), 48
__repr__() (*dataManagementClasses.TrackedObject* method), 50
_database (*DetectionPipeline.Detector* attribute), 12
_dataset (*DetectionPipeline.Detector* attribute), 12
_dataset (*dataManagementClasses.TrackedObject* attribute), 50
_history (*DetectionPipeline.DeepSORT* attribute), 10
_history (*DetectionPipeline.Detector* attribute), 12
_joblib (*DetectionPipeline.Detector* attribute), 12
_joblibbuffer (*DetectionPipeline.Detector* attribute), 12
_logger (*DetectionPipeline.Yolov7* attribute), 14
_model (*DetectionPipeline.Detector* attribute), 12
_outdir (*DetectionPipeline.Detector* attribute), 12
_source (*DetectionPipeline.Detector* attribute), 11

A

affinityPropagation_on_enter_and_exit_points() (*in module clustering*), 34
affinityPropagation_on_featureVector() (*in module clustering*), 34
all_class_under_threshold() (*in module classification*), 19
aoi_clutsering_search_birch() (*in module clustering*), 34
Attributes (*classifier.BinaryClassifier* attribute), 28
avgArea() (*dataManagementClasses.TrackedObject* method), 51, 52
AX (*dataManagementClasses.Detection* attribute), 47, 48
AX (*dataManagementClasses.TrackedObject* attribute), 50, 52

AY (*dataManagementClasses.Detection* attribute), 47, 48
AY (*dataManagementClasses.TrackedObject* attribute), 50, 52

B

bbox2float() (*in module utility.databaseLogger*), 63
BinaryClassificationTrain() (*in module classification*), 16
BinaryClassificationWorkerTrain() (*in module classification*), 17
BinaryClassifier (*class in classifier*), 28
BinaryDecisionTreeClassification() (*in module classification*), 17

C

calc_cluster_centers() (*in module clustering*), 34
calculate_metrics_exitpoints() (*in module classification*), 20
CalibratedClassification() (*in module classification*), 17
CalibratedClassificationWorker() (*in module classification*), 17
centroid_coordinates (*classifier.OneVSRestClassifierExtended* attribute), 30
centroid_labels (*classifier.OneVSRestClassifierExtended* attribute), 30
checkDir() (*in module utility.general*), 71
classification module, 16
Classification() (*in module classification*), 17
ClassificationWorker() (*in module classification*), 17
classifier module, 28
closeConnection() (*in module utility.databaseLogger*), 64
cluster_optics_dbscan_on_featurevectors() (*in module clustering*), 35
cluster_optics_dbscan_on_nx4() (*in module clustering*), 35

cluster_optics_dbscan_plotter() (in module clustering), 36
 clustering module, 34
 clustering_on_2D_feature_vectors() (in module clustering), 36
 clustering_on_4D_feature_vectors() (in module clustering), 36
 clustering_on_6D_feature_vectors() (in module clustering), 36
 clustering_on_feature_vectors() (in module clustering), 37
 clustering_search_on_2D_feature_vectors() (in module clustering), 37
 clustering_search_on_4D_feature_vectors() (in module clustering), 37
 clustering_search_on_6D_feature_vectors() (in module clustering), 37
 colors (DetectionPipeline.Yolov7 attribute), 15
 conf_thres (DetectionPipeline.Yolov7 attribute), 15
 confidence (dataManagementClasses.Detection attribute), 46, 48
 coordinates2heatmap() (in module database_metadata), 59
 cross_validate() (in module classification), 20
 cross_validate_multiclass() (in module classification), 21
 cross_validation_multiclass_submodule() (in module classification), 21
 cross_validation_submodule() (in module classification), 22
 cvCoord2npCoord() (in module utility.plots), 73
D
 data_preprocessing_for_calibrated_classifier() (in module classification), 22
 data_preprocessing_for_classifier() (in module classification), 22
 data_preprocessing_for_classifier_from_joblib_model() (in module classification), 22
 database_metadata module, 59
 dataManagementClasses module, 46
 dbscan_clustering_on_nx4() (in module clustering), 37
 dbscan_on_featureVectors() (in module clustering), 37
 dbscan_worker() (in module clustering), 38
 DeepSORT (class in DetectionPipeline), 9
 detect module, 59
 detect() (in module detect), 59
 Detection (class in dataManagementClasses), 46
 detectionExists() (in module utility.databaseLogger), 64
 detectionFactory() (in module dataManagementClasses), 56
 detectionParser() (in module dataManagementClasses), 56
 DetectionPipeline module, 9
 Detector (class in DetectionPipeline), 11
 device (DetectionPipeline.Yolov7 attribute), 14
 diff() (in module utility.general), 71
 diffmap() (in module utility.general), 72
 downscale_feature() (dataManagementClasses.TrackedObject method), 51
 downscale_feature() (dataManagementClasses.TrackedObject static method), 52
 downscale_TrackedObjects() (in module utility.dataset), 67
 dowscalecoord() (in module utility.databaseLogger), 64
 dt() (in module utility.general), 72
 DTClassification() (in module classification), 17
E
 elbow_on_clustering() (in module clustering), 38
 elbow_on_kmeans() (in module clustering), 38
 elbow_plot_worker() (in module clustering), 38
 elbow_plotter() (in module clustering), 39
 elbow_visualizer() (in module clustering), 39
 estimator (classifier.OneVSRestClassifierExtended attribute), 30
 euclidean_distance() (in module utility.preprocessing), 75
 evaluate module, 59
 exitpoint_metric_module() (in module classification), 23
 extractHourlyData() (in module traffic_statistics), 60
 model()
F
 factory_1() (utility.featurevector.FeatureVector static method), 69
 factory_10() (utility.featurevector.FeatureVector static method), 69
 factory_1_SG() (utility.featurevector.FeatureVector static method), 69
 factory_7() (utility.featurevector.FeatureVector static method), 70
 factory_7_SG() (utility.featurevector.FeatureVector static method), 70
 factory_8() (utility.featurevector.FeatureVector static method), 71
 factory_8_SG() (utility.featurevector.FeatureVector static method), 71

- factory_9() (*utility.featurevector.FeatureVector* static method), 71
- feature_v1() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v1_SG() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v3() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v5() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v7() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v7_SG() (*dataManagementClasses.TrackedObject* method), 51, 53
- feature_v8() (*dataManagementClasses.TrackedObject* method), 51, 54
- feature_v8_SG() (*dataManagementClasses.TrackedObject* method), 51, 54
- feature_vector() (*dataManagementClasses.TrackedObject* method), 51, 54
- FeatureVector (*class in utility.featurevector*), 69
- filter_by_class() (*in module utility.preprocessing*), 75
- filter_objects() (*DetectionPipeline.Detector* method), 12
- filter_objects() (*DetectionPipeline.Detector* static method), 13
- filter_out_edge_detections() (*in module utility.preprocessing*), 76
- filter_out_false_positive_detections_by_enter_exit_distance() (*in module utility.preprocessing*), 76
- filter_out_noise_trajectories() (*in module utility.preprocessing*), 76
- filter_trajectories() (*in module utility.preprocessing*), 76
- findEnterAndExitPoints() (*in module dataManagementClasses*), 57
- fit() (*classifier.BinaryClassifier* method), 29
- fit() (*classifier.OneVSRestClassifierExtended* method), 31
- frameID (*dataManagementClasses.Detection* attribute), 47, 48
- futureX (*dataManagementClasses.TrackedObject* attribute), 48, 55
- futureY (*dataManagementClasses.TrackedObject* attribute), 48, 55
- G**
- generate_db_path() (*DetectionPipeline.Detector* method), 12
- generate_db_path() (*DetectionPipeline.Detector* static method), 13
- getConnection() (*in module utility.databaseLogger*), 64
- getLatestFrame() (*in module utility.databaseLogger*), 65
- GNBClassification() (*in module classification*), 17
- GPCClassification() (*in module classification*), 18
- H**
- half (*DetectionPipeline.Yolov7* attribute), 15
- Height (*dataManagementClasses.Detection* attribute), 47, 48
- history (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_AX_calculated (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_AY_calculated (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_length (*DetectionPipeline.DeepSORT* attribute), 9
- history_VX_calculated (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_VY_calculated (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_X (*dataManagementClasses.TrackedObject* attribute), 49, 55
- history_Y (*dataManagementClasses.TrackedObject* attribute), 49, 55
- historyDepth (*DetectionPipeline.DeepSORT* attribute), 10
- hourlyHistogram() (*in module traffic_statistics*), 60
- hourlyStatisticsModule() (*in module traffic_statistics*), 60
- hourlyTable() (*in module traffic_statistics*), 61
- I**
- imgsz (*DetectionPipeline.Yolov7* attribute), 15
- infer() (*DetectionPipeline.Yolov7* method), 15
- init_db() (*in module utility.databaseLogger*), 65
- init_logger() (*in module utility.logging*), 72
- init_tracker_metric() (*DetectionPipeline.DeepSORT* static method), 10
- insert_weights_into_feature_vector() (*in module dataManagementClasses*), 57
- investigate_renitent_features() (*in module classification*), 23
- investigateRenitent() (*in module classification*), 23
- iou_thres (*DetectionPipeline.Yolov7* attribute), 15
- is_noise() (*in module utility.preprocessing*), 76
- isMoving (*dataManagementClasses.TrackedObject* attribute), 49, 55
- iter_minibatches() (*in module utility.training*), 77
- K**
- k_means_on_featureVectors() (*in module clustering*), 39

- kmeans_clustering_on_nx2() (in module clustering), 39
 kmeans_clustering_on_nx4() (in module clustering), 40
 kmeans_mse_clustering() (in module clustering), 40
 kmeans_mse_search() (in module clustering), 40
 kmeans_worker() (in module clustering), 41
 KNNClassification() (in module classification), 18
- ## L
- label (dataManagementClasses.Detection attribute), 46, 48
 label (dataManagementClasses.TrackedObject attribute), 48, 55
 level_features() (in module classification), 23
 load_dataset() (in module utility.dataset), 68
 load_dataset_with_labels() (in module utility.dataset), 68
 load_model() (in module utility.models), 73
 loadDatasetMultiprocessed() (in module utility.dataset), 67
 loadDatasetMultiprocessedCallback() (in module utility.dataset), 67
 loadDatasetsFromDirectory() (in module utility.dataset), 67
 loadDetections() (in module utility.databaseLoader), 62
 loadDetectionsOfObject() (in module utility.databaseLoader), 63
 loadMetadata() (in module utility.databaseLoader), 63
 loadObjects() (in module utility.databaseLoader), 63
 loadPredictions() (in module utility.databaseLoader), 63
 loadRegression() (in module utility.databaseLoader), 63
 logBuffer() (in module utility.databaseLogger), 65
 logBufferSpeedy() (in module utility.databaseLogger), 65
 logDetection() (in module utility.databaseLogger), 65
 logMetaData() (in module utility.databaseLogger), 66
 logObject() (in module utility.databaseLogger), 66
 logPredictions() (in module utility.databaseLogger), 66
 logRegression() (in module utility.databaseLogger), 66
- ## M
- main() (in module classification), 23
 main() (in module clustering), 41
 main() (in module database_metadata), 59
 main() (in module masker), 59
 main() (in module traffic_statistics), 61
 make_2D_feature_vectors() (in module clustering), 41
 make_4D_feature_vectors() (in module clustering), 41
 make_6D_feature_vectors() (in module clustering), 42
 make_detection_object() (DetectionPipeline.DeepSORT static method), 11
 make_feature_vectors_version_eight() (in module classification), 23
 make_feature_vectors_version_five() (in module classification), 24
 make_feature_vectors_version_four() (in module classification), 24
 make_feature_vectors_version_one() (in module classification), 24
 make_feature_vectors_version_one_half() (in module classification), 24
 make_feature_vectors_version_seven() (in module classification), 25
 make_feature_vectors_version_six() (in module classification), 25
 make_feature_vectors_version_three() (in module classification), 25
 make_feature_vectors_version_three_half() (in module classification), 25
 make_feature_vectors_version_two() (in module classification), 26
 make_feature_vectors_version_two_half() (in module classification), 26
 make_features_for_classification() (in module classification), 26
 make_features_for_classification_velocity() (in module classification), 26
 makeColormap() (in module utility.plots), 73
 makeFeatureVectors_Nx2() (in module clustering), 41
 masker
 module, 59
 masker() (in module masker), 59
 max_age (dataManagementClasses.TrackedObject attribute), 49, 55
 max_cosine_distance (DetectionPipeline.DeepSORT attribute), 10
 max_iou_distance (DetectionPipeline.DeepSORT attribute), 10
 mean (dataManagementClasses.TrackedObject attribute), 50, 55
 mergeDatasets() (in module utility.dataset), 68
 MLPClassification() (in module classification), 18
 model (DetectionPipeline.Yolov7 attribute), 14
 module
 classification, 16
 classifier, 28
 clustering, 34
 database_metadata, 59
 dataManagementClasses, 46

detect, 59
 DetectionPipeline, 9
 evaluate, 59
 masker, 59
 traffic_statistics, 60
 train, 62
 utility, 77
 utility.databaseLoader, 62
 utility.databaseLogger, 63
 utility.dataset, 67
 utility.featurevector, 69
 utility.general, 71
 utility.logging, 72
 utility.models, 73
 utility.plots, 73
 utility.preprocessing, 75
 utility.training, 77

N

n_jobs (*classifier.OneVSRestClassifierExtended* attribute), 30
 names (*DetectionPipeline.Yolov7* attribute), 15
 nn_budget (*DetectionPipeline.DeepSORT* attribute), 10

O

objExists() (in module *utility.databaseLogger*), 67
 objID (*dataManagementClasses.Detection* attribute), 47, 48
 objID (*dataManagementClasses.TrackedObject* attribute), 48, 55
 OneVSRestClassifierExtended (class in *classifier*), 30
 optics_clustering_on_nx4() (in module *clustering*), 42
 optics_dbscan_worker() (in module *clustering*), 42
 optics_on_featureVectors() (in module *clustering*), 43
 optics_worker() (in module *clustering*), 43
 output (*DetectionPipeline.DeepSORT* attribute), 10

P

parseRawObject2TrackedObject() (in module *dataManagementClasses*), 57
 partial_fit() (*classifier.OneVSRestClassifierExtended* method), 32
 plot_decision_tree() (in module *classification*), 27
 plot_misclassified() (in module *utility.plots*), 74
 plot_misclassified_feature_vectors() (in module *utility.plots*), 74
 plot_module() (in module *classification*), 27
 plot_one_cluster() (in module *utility.plots*), 74
 plot_one_prediction() (in module *utility.plots*), 75
 postprocess() (*DetectionPipeline.Yolov7* method), 16

predict() (*classifier.BinaryClassifier* method), 29
 predict() (*classifier.OneVSRestClassifierExtended* method), 32
 predict() (*DetectionPipeline.Detector* method), 13
 predict_proba() (*classifier.BinaryClassifier* method), 29
 predict_proba() (*classifier.OneVSRestClassifierExtended* method), 32
 prediction2float() (in module *utility.databaseLogger*), 67
 preprocess() (*DetectionPipeline.Yolov7* method), 16
 preprocess_database_data() (in module *dataManagementClasses*), 58
 preprocess_database_data_multiprocessed() (in module *dataManagementClasses*), 58
 preprocess_dataset_for_training() (in module *classification*), 27
 printConfig() (in module *database_metadata*), 59
 printSamplesToExcel() (in module *traffic_statistics*), 61

Q

queryLastObjID() (in module *utility.databaseLoader*), 63

R

run() (*DetectionPipeline.Detector* method), 12, 14

S

save_model() (in module *utility.models*), 73
 save_trajectories() (in module *utility.dataset*), 68
 savePlot() (in module *utility.plots*), 75
 search_min_max_coordinates() (in module *utility.preprocessing*), 77
 SGDClassification() (in module *classification*), 18
 shuffle_data() (in module *utility.preprocessing*), 77
 simple_dbscan_plotter() (in module *clustering*), 44
 simple_kmeans_plotter() (in module *clustering*), 44
 simple_optics_plotter() (in module *clustering*), 44
 simple_spectral_plotter() (in module *clustering*), 44
 source (*DetectionPipeline.DeepSORT* attribute), 9
 spectral_clustering_on_nx4() (in module *clustering*), 44
 spectral_on_featureVectors() (in module *clustering*), 45
 spectral_worker() (in module *clustering*), 45
 strfy_dict_params() (in module *utility.general*), 72
 stride (*DetectionPipeline.Yolov7* attribute), 14
 submodule_aoi_birch() (in module *clustering*), 45
 submodule_aoi_kmeans() (in module *clustering*), 45
 submodule_birch() (in module *clustering*), 45
 submodule_dbscan() (in module *clustering*), 45

submodule_kmeans() (in module clustering), 45
submodule_optics() (in module clustering), 45
SVMClassfictaion() (in module classification), 19

T

time_since_update (dataManagementClasses.TrackedObject attribute), 49, 55
TrackedObject (class in dataManagementClasses), 48
trackedObjectFactory() (in module dataManagementClasses), 58
tracker_factory() (DetectionPipeline.DeepSORT static method), 11
tracks (classifier.OneVSRestClassifierExtended attribute), 30
tracks2joblib() (in module dataManagementClasses), 58
traffic_statistics module, 60
trafficHistogram() (in module traffic_statistics), 61
trafficHistogramModule() (in module traffic_statistics), 62
trafficStatisticTable() (in module traffic_statistics), 62
train module, 62
train() (in module train), 62
train_binary_classifiers() (in module classification), 27
train_binary_classifiers_submodule() (in module classification), 27
true_class_under_threshold() (in module classification), 27

U

update() (dataManagementClasses.TrackedObject method), 55
update_accel() (dataManagementClasses.TrackedObject method), 55
update_history() (DetectionPipeline.DeepSORT method), 11
update_velocity() (dataManagementClasses.TrackedObject method), 55
updateAccel() (dataManagementClasses.TrackedObject method), 51, 55
upscale_cluster_centers() (in module clustering), 46
upscale_feature() (dataManagementClasses.TrackedObject method), 51
upscale_feature() (dataManagementClasses.TrackedObject static method), 56
utility module, 77
utility.databaseLoader module, 62

utility.databaseLogger module, 63
utility.dataset module, 67
utility.featurevector module, 69
utility.general module, 71
utility.logging module, 72
utility.models module, 73
utility.plots module, 73
utility.preprocessing module, 75
utility.training module, 77

V

validate() (classifier.BinaryClassifier method), 29
validate() (classifier.OneVSRestClassifierExtended method), 33
validate_models() (in module classification), 27
validate_predictions() (classifier.BinaryClassifier method), 29
validate_predictions() (classifier.OneVSRestClassifierExtended method), 33
ValidateClassification() (in module classification), 19
ValidateClassification_Probability() (in module classification), 19
VotingClassification() (in module classification), 19
VX (dataManagementClasses.Detection attribute), 47, 48
VX (dataManagementClasses.TrackedObject attribute), 50, 52
VY (dataManagementClasses.Detection attribute), 47, 48
VY (dataManagementClasses.TrackedObject attribute), 50, 52

W

warmup() (DetectionPipeline.Yolov7 method), 16
warn() (in module database_metadata), 59
Width (dataManagementClasses.Detection attribute), 47, 48

X

X (dataManagementClasses.Detection attribute), 47, 48
X (dataManagementClasses.TrackedObject attribute), 50, 52

Y

Y (*dataManagementClasses.Detection* attribute), [47](#), [48](#)

Y (*dataManagementClasses.TrackedObject* attribute), [50](#),
[52](#)

Yolov7 (class in *DetectionPipeline*), [14](#)