ml_logger

Release 0.7

Shagun Sodhani

Jun 18, 2022

GETTING STARTED

CHAPTER

ONE

WHY ML_LOGGER

People use different tools for logging experimental results - Tensorboard, Wandb etc to name a few. Working with different collaborators, I will have to switch my logging tool with each new project. So I made this simple tool that provides a common interface to logging results to different loggers.

CHAPTER

TWO

INSTALLATION

- `pip install "mllogger[all]"`

If you want to use only the filesystem logger, use `pip install "mllogger"`

Install from source

- `git clone git@github.com:shagunsodhani/ml-logger.git`
- `cd ml-logger`
- `pip install ".[all]"`

Alternatively, `pip install "git+https://git@github.com/shagunsodhani/ml-logger.git@master#egg=ml_logger[all]"`

If you want to use only the filesystem logger, use `pip install .` or `pip install "git+https://git@github.com/shagunsodhani/ml-logger.git@master#egg=ml_logger"`.

CHAPTER

THREE

USE

- Make a `logbook_config`:

```
from ml_logger import logbook as ml_logbook
logbook_config = ml_logbook.make_config(
    logger_dir = <path to write logs>,
    wandb_config = <wandb config or None>,
    tensorboard_config = <tensorboard config or None>,
    mlflow_config = <mlflow config or None>)
```

  The API for `make_config` can be accessed here.

- Make a `LogBook` instance:

```
logbook = ml_logbook.LogBook(config = logbook_config)
```

- Use the `logbook` instance:

```
log = {
    "epoch": 1,
    "loss": 0.1,
    "accuracy": 0.2
}
logbook.write_metric(log)
```

  The API for `write_metric` can be accessed here.

CHAPTER

FOUR

NOTE

- If you are writing to wandb, the `log` must have a key called `step`. If your `log` already captures the `step` but as a different key (say `epoch`), you can pass the `wandb_key_map` argument (set as `{epoch: step}`). For more details, refer the documentation here.
- If you are writing to mlflow, the `log` must have a key called `step`. If your `log` already captures the `step` but as a different key (say `epoch`), you can pass the `mlflow_key_map` argument (set as `{epoch: step}`). For more details, refer the documentation here.
- If you are writing to tensorboard, the `log` must have a key called `main_tag` or `tag` which acts as the data Identifier and another key called `global_step`. These keys are described here. If your `log` already captures these values but as different key (say `mode` for `main_tag` and `epoch` for `global_step`), you can pass the `tensorboard_key_map` argument (set as `{mode: main_tag, epoch: global_step}`). For more details, refer the documentation here.

CHAPTER

FIVE

DEV SETUP

- `pip install -e ".[dev]"`
- Install pre-commit hooks `pre-commit install`
- The code is linted using:
  - `black`
  - `flake8`
  - `mypy`
  - `isort`
- Tests can be run locally using `nox`

CHAPTER

SIX

ACKNOWLEDGEMENTS

- Config for `circleci`, `pre-commit`, `mypy` etc are borrowed/modified from Hydra

CHAPTER

SEVEN

ML_LOGGER

7.1 ml_logger package

7.1.1 Subpackages

ml_logger.logger package

Submodules

ml_logger.logger.base module

Abstract logger class.

class ml_logger.logger.base.**Logger**(*config: Dict[str, Any]*)
Bases: object

Abstract Logger Class.

abstract write(*log: Dict[str, Any]*) → None
Interface to write the log.

Parameters **log** (*LogType*) – Log to write

ml_logger.logger.filesystem module

Functions to interface with the filesystem.

class ml_logger.logger.filesystem.**Logger**(*config: Dict[str, Any]*)
Bases: *ml_logger.logger.base.Logger*

Logger class that writes to the filesystem.

write(*log: Dict[str, Any]*) → None
Write the log to the filesystem.

Parameters **log** (*LogType*) – Log to write

ml_logger.logger.filesystem.**get_logger_file_path**(*logger_dir: str*, *filename: Optional[str]*, *filename_prefix: str*, *filename_suffix: str*) → str
Get path to the file (to write logs to).

ml_logger.logger.filesystem.**to_json_serializable**(*val: Any*) → Any
Serialize values as json.

ml_logger.logger.mlflow module

Logger class that writes to mlflow.

class ml_logger.logger.mlflow.**Logger**(*config: Dict[str, Any]*)
Bases: *ml_logger.logger.base.Logger*

Logger class that writes to mlflow.

write(*log: Dict[str, Any]*) → None
Write the log to mlflow.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write the config to mlflow.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to mlflow.

Parameters **metric** (*MetricType*) – Metric to write

ml_logger.logger.mongo module

Functions to interface with the mongodb.

class ml_logger.logger.mongo.**Logger**(*config: Dict[str, Any]*)
Bases: *ml_logger.logger.base.Logger*

Logger class that writes to the mongodb.

write(*log: Dict[str, Any]*) → None
Write the log to the filesystem.

Parameters **log** (*LogType*) – Log to write

ml_logger.logger.tensorboard module

Logger class that writes to tensorboard.

class ml_logger.logger.tensorboard.**Logger**(*config: Dict[str, Any]*)
Bases: *ml_logger.logger.base.Logger*

Logger class that writes to tensorboardX.

write(*log: Dict[str, Any]*) → None
Write the log to tensorboard.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write the config to tensorboard.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to tensorboard.

Parameters **metric** (*MetricType*) – Metric to write

ml_logger.logger.wandb module

Logger class that writes to wandb.

class ml_logger.logger.wandb.**Logger**(*config: Dict[str, Any]*)
Bases: *ml_logger.logger.base.Logger*

Logger class that writes to wandb.

write(*log: Dict[str, Any]*) → None
Write log to wandb.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write config to wandb.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to wandb.

Parameters **metric** (*MetricType*) – Metric to write

Module contents

ml_logger.parser package

Subpackages

ml_logger.parser.experiment package

Submodules

ml_logger.parser.experiment.experiment module

Container for the experiment data.

class ml_logger.parser.experiment.experiment.**Experiment**(*configs: List[Dict[str, Any]]*, *metrics: Dict[str, pandas.core.frame.DataFrame]*, *info: Optional[Dict[Any, Any]] = None*)
Bases: object

property config
Access the config property.

serialize(*dir_path: str*) → None
Serialize the experiment data and store at *dir_path*.

- configs are stored as jsonl (since there are only a few configs per experiment) in a file called *config.jsonl*.
- metrics are stored in [*feather* format](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_feather.html).

- info is stored in the gzip format.

class ml_logger.parser.experiment.experiment.**ExperimentSequence**(*experiments: List[*ml_logger.parser.experiment.exper

Bases: collections.UserList

aggregate(*aggregate_configs: Callable[[List[List[Dict[str, Any]]]], List[Dict[str, Any]]] = <function return_first_config>, aggregate_metrics: Callable[[List[Dict[str, pandas.core.frame.DataFrame]]], Dict[str, pandas.core.frame.DataFrame]] = <function concat_metrics>, aggregate_infos: Callable[[List[Dict[Any, Any]]], Dict[Any, Any]] = <function return_first_infos>*) → *ml_logger.parser.experiment.experiment.Experiment*

Aggregate a sequence of experiments into a single experiment.

Parameters

- **aggregate_configs** (*Callable[[List[List[ConfigType]]], List[ConfigType]], optional*) – Function to aggregate the configs. Defaults to return_first_config.
- **aggregate_metrics** (*Callable[[List[ExperimentMetricType]], ExperimentMetricType], optional*) – Function to aggregate the metrics. Defaults to concat_metrics.
- **aggregate_infos** (*Callable[[List[ExperimentInfoType]], ExperimentInfoType], optional*) – Function to aggregate the information. Defaults to return_first_infos.

Returns Aggregated Experiment.

Return type *Experiment*

filter(*filter_fn: Callable[[*ml_logger.parser.experiment.experiment.Experiment*], bool]*) → *ml_logger.parser.experiment.experiment.ExperimentSequence*

Filter experiments in the sequence.

Parameters **filter_fn** – Function to filter an experiment

Returns A sequence of experiments for which the filter condition is true

Return type *ExperimentSequence*

groupby(*group_fn: Callable[[*ml_logger.parser.experiment.experiment.Experiment*], str]*) → Dict[str, *ml_logger.parser.experiment.experiment.ExperimentSequence*]

Group experiments in the sequence.

Parameters **group_fn** – Function to assign a string group id to the experiment

Returns A dictionary mapping the sring group id to a sequence of experiments

Return type Dict[str, *ExperimentSequence*]

ml_logger.parser.experiment.experiment.**concat_metrics**(*metric_list: List[Dict[str, pandas.core.frame.DataFrame]]*) → Dict[str, pandas.core.frame.DataFrame]

Concatenate the metrics.

Parameters **metric_list** (*List[ExperimentMetricType]*) –

Returns ExperimentMetricType

ml_logger.parser.experiment.experiment.**deserialize**(*dir_path: str*) → *ml_logger.parser.experiment.experiment.Experiment*

Deserialize the experiment data stored at *dir_path* and return an Experiment object.

ml_logger.parser.experiment.experiment.**return_first_config**(*config_lists: List[List[Dict[str, Any]]]*) → List[Dict[str, Any]]

Return the first config list, from a list of list of configs, else return empty list.

Parameters **config_lists** (*List[List[ConfigType]]*) –

Returns List[ConfigType]

ml_logger.parser.experiment.experiment.**return_first_infos**(*info_list: List[Dict[Any, Any]]*) → Dict[Any, Any]

Return the first info, from a list of infos. Otherwise return empty info.

Parameters **info_list** (*List[ExperimentInfoType]*) –

Returns ExperimentInfoType

ml_logger.parser.experiment.parser module

Implementation of Parser to parse experiment from the logs.

class ml_logger.parser.experiment.parser.**Parser**(*parse_config_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>, parse_metric_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>, parse_info_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: *ml_logger.parser.base.Parser*

Class to parse an experiment from the log dir.

parse(*filepath_pattern: Union[str, pathlib.Path]*) → *ml_logger.parser.experiment.experiment.Experiment*

Load one experiment from the log dir.

Parameters **filepath_pattern** (*Union[str, Path]*) – filepath pattern to glob or instance of Path (directory) object.

Returns Experiment

Module contents

Module to interact with the experiment data.

Submodules

ml_logger.parser.base module

Base class that all parsers extend.

class ml_logger.parser.base.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: abc.ABC

Base class that all parsers extend.

ml_logger.parser.config module

Implementation of Parser to parse config from logs.

class ml_logger.parser.config.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>*)

Bases: *ml_logger.parser.log.Parser*

Class to parse config from the logs.

ml_logger.parser.config.**parse_json_and_match_value**(*line: str*) → Optional[Dict[str, Any]]

Parse a line as JSON log and check if it a valid config log.

ml_logger.parser.log module

Implementation of Parser to parse the logs.

class ml_logger.parser.log.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: *ml_logger.parser.base.Parser*

Class to parse the log files.

parse(*filepath_pattern: str*) → Iterator[Dict[str, Any]]

Open a log file, parse its contents and return *logs*.

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns Iterator over the logs

Return type Iterator[LogType]

Yields *Iterator[LogType]* – Iterator over the logs

parse_first_log(*filepath_pattern: str*) → Optional[Dict[str, Any]]

Return the first log from a file.

The method will return after finding the first log. Unlike *parse()* method, it will not iterate over the entire log file (thus saving memory and time).

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns First instance of a log

Return type LogType

parse_last_log (*filepath_pattern: str*) → Optional[Dict[str, Any]]
Return the last log from a file.

Like *parse()* method, it will iterate over the entire log file but will not keep all the logs in memory (thus saving memory).

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns Last instance of a log

Return type LogType

ml_logger.parser.log.**parse_json_and_match_value** (*line: str, value: str*) → Optional[Dict[str, Any]]
Parse a line as JSON log and check if it a valid log.

ml_logger.parser.metric module

Implementation of Parser to parse metrics from logs.

class ml_logger.parser.metric.**Parser** (*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>*)
Bases: *ml_logger.parser.log.Parser*

Class to parse the metrics from the logs.

parse_as_df (*filepath_pattern: str, group_metrics: Callable[[List[Dict[str, Any]]], Dict[str, List[Dict[str, Any]]]] = <function group_metrics>, aggregate_metrics: Callable[[List[Dict[str, Any]]], List[Dict[str, Any]]] = <function aggregate_metrics>*) → Dict[str, pandas.core.frame.DataFrame]
Create a dict of (metric_name, dataframe).

Method that: (i) reads metrics from the filesystem (ii) groups metrics (iii) aggregates all the metrics within a group, (iv) converts the aggregate metrics into dataframes and returns a dictionary of dataframes

Parameters

- **filepath_pattern** (*str*) – filepath pattern to glob
- **group_metrics** (*Callable[[List[LogType]], Dict[str, List[LogType]]], optional*) – Function to group a list of metrics into a dictionary of (key, list of grouped metrics). Defaults to group_metrics.
- **aggregate_metrics** (*Callable[[List[LogType]], List[LogType]], optional*) – Function to aggregate a list of metrics. Defaults to aggregate_metrics.

ml_logger.parser.metric.**aggregate_metrics** (*metrics: List[Dict[str, Any]]*) → List[Dict[str, Any]]
Aggregate a list of metrics.

Parameters **metrics** (*List[MetricType]*) – List of metrics to aggregate

Returns List of aggregated metrics

Return type List[MetricType]

ml_logger.parser.metric.**group_metrics** (*metrics: List[Dict[str, Any]]*) → Dict[str, List[Dict[str, Any]]]
Group a list of metrics.

Group a list of metrics into a dictionary of (key, list of grouped metrics)

Parameters **metrics** (*List[MetricType]*) – List of metrics to group

Returns

Dictionary of (key, list of grouped metrics)

Return type Dict[str, List[MetricType]]

ml_logger.parser.metric.**metrics_to_df**(*metric_logs: List[Dict[str, Any]], group_metrics: Callable[[List[Dict[str, Any]]], Dict[str, List[Dict[str, Any]]]] = <function group_metrics>, aggregate_metrics: Callable[[List[Dict[str, Any]]], List[Dict[str, Any]]] = <function aggregate_metrics>*) → Dict[str, pandas.core.frame.DataFrame]

Create a dict of (metric_name, dataframe).

Method that: (i) groups metrics (ii) aggregates all the metrics within a group, (iii) converts the aggregate metrics into dataframes and returns a dictionary of dataframes

Parameters

- **metric_logs** (*List[LogType]*) – List of metrics
- **group_metrics** (*Callable[[List[LogType]], Dict[str, List[LogType]]], optional*) – Function to group a list of metrics into a dictionary of (key, list of grouped metrics). Defaults to group_metrics.
- **aggregate_metrics** (*Callable[[List[LogType]], List[LogType]], optional*) – Function to aggregate a list of metrics. Defaults to aggregate_metrics.

Returns [description]

Return type Dict[str, pd.DataFrame]

ml_logger.parser.metric.**parse_json_and_match_value**(*line: str*) → Optional[Dict[str, Any]]

Parse a line as JSON log and check if it a valid metric log.

ml_logger.parser.utils module

Utility functions for the parser module.

ml_logger.parser.utils.**compare_logs**(*first_log: Dict[str, Any], second_log: Dict[str, Any], verbose: bool = False*) → Tuple[List[str], List[str], List[str]]

Compare two logs.

Return list of keys that are either missing or have different valus in the two logs.

Parameters

- **first_log** (*LogType*) – First Log
- **second_log** (*LogType*) – Second Log
- **verbose** (*bool*) – Defaults to False

Returns

Tuple of [list of keys with different values, list of keys with values missing in first log, list of keys with values missing in the second log,]

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

ml_logger.parser.utils.**flatten_log**(*d: Dict[str, Any]*, *parent_key: str = ''*, *sep: str = '#'*) → Dict[str, Any]

Flatten a log using a separator.

Taken from https://stackoverflow.com/a/6027615/1353861

Parameters

- **d** (*LogType*) – [description]
- **parent_key** (*str, optional*) – [description]. Defaults to "".
- **sep** (*str, optional*) – [description]. Defaults to "#".

Returns [description]

Return type LogType

ml_logger.parser.utils.**parse_json**(*line: str*) → Optional[Dict[str, Any]]

Parse a line as JSON string.

Module contents

7.1.2 Submodules

7.1.3 ml_logger.logbook module

Implementation of the LogBook class.

LogBook class provides an interface to persist the logs on the filesystem, tensorboard, remote backends, etc.

class ml_logger.logbook.**LogBook**(*config: Dict[str, Any]*)

Bases: object

This class provides an interface to persist the logs on the filesystem, tensorboard, remote backends, etc.

write(*log: Dict[str, Any]*, *log_type: str = 'metric'*) → None

Write log to loggers.

Parameters

- **log** (*LogType*) – Log to write
- **log_type** (*str, optional*) – Type of this log. Defaults to "metric".

write_config(*config: Dict[str, Any]*) → None

Write config to loggers.

Parameters **[ConfigType]** (*config*) – Config to write.

write_message(*message: Any*, *log_type: str = 'info'*) → None

Write message string to loggers.

Parameters

- **message** (*Any*) – Message string to write
- **log_type** (*str, optional*) – Type of this message (log). Defaults to "info".

write_metadata(*metadata: Dict[str, Any]*) → None

Write metadata to loggers.

Parameters **metadata** (*LogType*) – Metadata to wite

write_metric(*metric: Dict[str, Any]*) → None
Write metric to loggers.

Parameters **metric** (*MetricType*) – Metric to write

ml_logger.logbook.**make_config**(*id: str = '0'*, *name: str = 'default_logger'*, *write_to_console: bool = True*, *logger_dir: Optional[str] = None*, *filename: Optional[str] = None*, *filename_prefix: str = ''*, *create_multiple_log_files: bool = True*, *wandb_config: Optional[Dict[str, Any]] = None*, *wandb_key_map: Optional[Dict[str, str]] = None*, *wandb_prefix_key: Optional[str] = None*, *tensorboard_config: Optional[Dict[str, Any]] = None*, *tensorboard_key_map: Optional[Dict[str, str]] = None*, *tensorboard_prefix_key: Optional[str] = None*, *mlflow_config: Optional[Dict[str, Any]] = None*, *mlflow_key_map: Optional[Dict[str, str]] = None*, *mlflow_prefix_key: Optional[str] = None*, *mongo_config: Optional[Dict[str, Any]] = None*) → Dict[str, Any]
Make the config that can be passed to the LogBook constructor.

Parameters

- **id** (*str, optional*) – Id of the current LogBook instance. Defaults to "0".
- **name** (*str, optional*) – Name of the logger. Defaults to "default_logger".
- **write_to_console** (*bool, optional*) – Should write the logs to console. Defaults to True
- **logger_dir** (*str, optional*) – Path where the logs will be written. If None is passed, logs are not written to the filesystem. LogBook creates the directory, if it does not exist. Defaults to None.
- **filename** (*str, optional*) – Name to assign to the log file (eg log.jsonl). If None is passed, this argument is ignored. If the value is set, *filename_prefix* and *create_multiple_log_files* arguments are ignored. Defaults to None.
- **filename_prefix** (*str*) – String to prefix before the name of the log files. Eg if filename_prefix is "dummy", name of log files are dummymetric.jsonl, dummylog.jsonl etc. This argument is ignored if *filename* is set. Defaults to "".
- **create_multiple_log_files** (*bool, optional*) – Should multiple log files be created - for config, metric, metadata and message logs. If True, the files are named as config_log.jsonl, metric_log.jsonl etc. If False, only one file log.jsonl is created. This argument is ignored if *filename* is set. Defaults to True.
- **wandb_config** (*Optional[ConfigType], optional*) – Config for the wandb logger. If None, wandb logger is not created. The config can have any parameters that wandb.init() methods accepts (https://docs.wandb.com/library/init). Note that the wandb_config is passed as keyword arguments to the wandb.init() method. This provides a lot of flexibility to the users to configure wandb. This also means that the config should not have any parameters that wandb.init() would not accept. Defaults to None.
- **wandb_key_map** (*Optional[KeyMapType], optional*) – When using wandb logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* to be written) with the keys that wandb logger needs. For instance, wandb logger needs a *step* key in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, set *wandb_key_map* as *{epoch: step}*. This argument is ignored if set to None. Defaults to None.
- **wandb_prefix_key** (*Optional[str], optional*) – When a metric is logged to wandb, prefix the value (corresponding to the key) to all the remaining keys before values

are logged in the wandb logger. This argument is ignored if set to None. Defaults to None.

- **tensorboard_config** (*Optional[ConfigType], optional*) – config to initialise the tensorboardX logger. The config can have any parameters that [tensorboardX.SummaryWriter() method](https://tensorboardx.readthedocs.io/en/latest/tensorboard.html#tensorboardX.SummaryWriter) accepts. Note that the config is passed as keyword arguments to the tensorboardX.SummaryWriter() method. This provides a lot of flexibility to the users to configure tensorboard. This also means that config should not have any parameters that tensorboardX.SummaryWriter() would not accept. Defaults to None.
- **tensorboard_key_map** (*Optional[KeyMapType], optional*) – When using tensorboard logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* (to be written) with the keys that tensorboard logger needs. For instance, tensorboard logger needs a *main_tag* key and a *global_step* in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, and a key called *mode* that you want to use as *main_tag*, set *tensorboard_key_map* as *{epoch: global_step, mode: main_tag}*. This argument is ignored if set to None. Defaults to None.
- **tensorboard_prefix_key** (*Optional[str], optional*) – When a metric is logged to tensorboard, prefix the value (corresponding to the key) to all the remaining keys before values are logged in the tensorboard logger. This argument is ignored if set to None. Defaults to None.
- **mlflow_config** (*Optional[ConfigType], optional*) – config to initialise an mlflow experiment. The config can have any parameters that [mlflow.create_experiment() method](https://mlflow.org/docs/latest/python_api/mlflow.html#mlflow.create_experiment) accepts. Note that the config is passed as keyword arguments to the mlflow.create_experiment() method. This provides a lot of flexibility to the users to configure mlflow. This also means that config should not have any parameters that mlflow.create_experiment would not accept. Defaults to None.
- **mlflow_key_map** (*Optional[KeyMapType], optional*) – When using mlflow logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* (to be written) with the keys that mlflow logger needs. For instance, mlflow logger needs a *step* key in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, set *mlflow_key_map* as *{epoch: step}*. This argument is ignored if set to None. Defaults to None.
- **mlflow_prefix_key** (*Optional[str], optional*) – When a metric is logged to mlflow, prefix the value (corresponding to the key) to all the remaining keys before values are logged in the mlflow logger. This argument is ignored if set to None. Defaults to None.
- **mongo_config** (*Optional[ConfigType], optional*) – config to initialise connection to a collection in mongodb. The config supports the following keys:

  (1) host: host where mongodb is running.

  (2) port: port on which mongodb is running.

  (3) db: name of the db to use.

  (4) collection: name of the collection to use.

  Defaults to None.

Returns config to construct the LogBook

Return type ConfigType

7.1.4 ml_logger.metrics module

Implementation of different type of metrics.

class ml_logger.metrics.**AverageMetric**(*name: str*)
Bases: *ml_logger.metrics.BaseMetric*

Metric to track the average value.

This is generally used for logging strings

Parameters **BaseMetric** – Base metric class

get_val() → float
Get the current average value.

reset() → None
Reset Metric.

update(*val: Union[int, float]*, *n: int = 1*) → None
Update the metric.

Update the metric using the current average value and the number of samples used to compute the average value

Parameters

- **val** (*NumType*) – current average value
- **n** (*int, optional*) – Number of samples used to compute the average. Defaults to 1

class ml_logger.metrics.**BaseMetric**(*name: str*)
Bases: object

Base Metric class. This class is not to be used directly.

get_val() → Union[str, int, float]
Get the current value of the metric.

reset() → None
Reset the metric to the default value.

update(*val: Any*) → None
Update the metric using the current val.

Parameters **val** (*Any*) – Current value. This value is used to update the metric

class ml_logger.metrics.**ComparisonMetric**(*name: str*, *default_val: Union[str, int, float]*, *comparison_op: Callable[[Union[str, int, float], Union[str, int, float]], bool]*)
Bases: *ml_logger.metrics.BaseMetric*

Metric to track the min/max value.

This is generally used for logging best accuracy, least loss, etc.

Parameters **BaseMetric** – Base metric class

reset() → None
Reset the metric to the default value.

update(*val: Union[str, int, float]*) → None
Use the comparison operator to decide which value to keep.

If the output of self.comparison_op(val, self)

Parameters **val** (*ValueType*) – Value to compare the current value with. If comparison_op(current_val, new_val) is true, we update the current value.

class ml_logger.metrics.**ConstantMetric**(*name: str*, *val: Union[str, int, float]*)
Bases: *ml_logger.metrics.BaseMetric*

Metric to track one fixed value.

This is generally used for logging strings

Parameters **BaseMetric** – Base metric class

reset() → None
Do nothing for the constant metrics.

update(*val: Optional[Union[str, int, float]] = None*) → None
Do nothing for the constant metrics.

Parameters **val** (*Any*) – This value is ignored

class ml_logger.metrics.**CurrentMetric**(*name: str*)
Bases: *ml_logger.metrics.BaseMetric*

Metric to track only the most recent value.

Parameters **BaseMetric** – Base metric class

update(*val: Union[str, int, float]*) → None
Update the metric using the current val.

Parameters **val** (*Any*) – Current value. The metric value is set to this value

class ml_logger.metrics.**MaxMetric**(*name: str*)
Bases: *ml_logger.metrics.ComparisonMetric*

Metric to track the max value.

This is generally used for logging best accuracy, etc.

Parameters **ComparisonMetric** – Comparison metric class

class ml_logger.metrics.**MetricDict**(*metric_list: Iterable[*ml_logger.metrics.BaseMetric*]*)
Bases: object

Class that wraps over a collection of metrics.

reset() → None
Reset all the metrics to default values.

to_dict() → Dict[str, Any]
Convert the metrics into a dictionary for *LogBook*.

Returns Metric data in as a dictionary

Return type LogType

update(*metrics_dict: Union[Dict[str, Any],* ml_logger.metrics.MetricDict*]*) → None
Update all the metrics using the current values.

Parameters **metrics_dict** (*Union[LogType,* MetricDict*]*) – Current value of metrics

class ml_logger.metrics.**MinMetric**(*name: str*)
Bases: *ml_logger.metrics.ComparisonMetric*

Metric to track the min value.

This is generally used for logging least loss, etc.

Parameters **ComparisonMetric** – Comparison metric class

class ml_logger.metrics.**SumMetric**(*name: str*)
Bases: *ml_logger.metrics.AverageMetric*

Metric to track the sum value.

Parameters **BaseMetric** – Base metric class

get_val() → float
Get the current sum value.

7.1.5 ml_logger.types module

Types used in the package.

7.1.6 ml_logger.utils module

Utility Methods.

ml_logger.utils.**compare_keys_in_dict**(*dict1: Dict[Any, Any]*, *dict2: Dict[Any, Any]*) → bool
Check that the two dicts have the same set of keys.

ml_logger.utils.**flatten_dict**(*d: Dict[str, Any]*, *parent_key: str = ''*, *sep: str = '#'*) → Dict[str, Any]
Flatten a given dict using the given seperator.

Taken from https://stackoverflow.com/a/6027615/1353861

Parameters

- **d** (*Dict[str, Any]*) – dictionary to flatten
- **parent_key** (*str, optional*) – Keep track of the higher level key Defaults to "".
- **sep** (*str, optional*) – string for concatenating the keys. Defaults to "#"

Returns [description]

Return type Dict[str, Any]

ml_logger.utils.**make_dir**(*path: str*) → None
Make dir, if not exists.

Parameters **path** (*str*) – dir to make

7.1.7 Module contents

CHAPTER

EIGHT

COMMUNITY

- If you have questions, open an Issue
- Or, use Github Discussions
- To contribute, open a Pull Request (PR)

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

P

R

S

T

U

W