Gordo Documentation

Erik Parmann, Kristian Flikka, Miles Granger, Natalie C

Dec 20, 2021

PROJECT RESOURCES:

CHAPTER

ONE

OVERVIEW

Gordo is a collection of tools to create a distributed ML service represented by a specific pipeline. Generally, any `sklearn.pipeline.Pipeline` object can be defined within a config file and deployed as a REST API on Kubernetes.

1.1 Quick start

The concept of `Gordo` is to (as of now) process, only, *timeseries* datasets which are comprised of sensors/tag identifies. The workflow launches the collection of these tags, building of a defined model and subsequent deployment of a ML Server which acts as a REST interface in front of the model.

A typical config file might look like this:

```
apiVersion: equinor.com/v1
kind: Gordo
metadata:
  name: test-project
spec:
  deploy-version: 0.39.0
  config:

    machines:

      # This machine specifies all keys, and will train a model on one␣
↪month
      # worth of data, as shown in its train_start/end_date dataset␣
↪keys.
      - name: some-name-here
        dataset:
          train_start_date: 2018-01-01T00:00:00Z
          train_end_date: 2018-02-01T00:00:00Z
          resolution: 2T  # Resample timeseries at 2min intervals␣
↪(pandas freq strings)
```

(continues on next page)

(continued from previous page)

```
        tags:
          - tag-1
          - tag-2
      model:
        sklearn.pipeline.Pipeline:
          steps:
            - sklearn.preprocessing.MinMaxScaler
            - gordo.model.models.KerasAutoEncoder:
                kind: feedforward_hourglass
      metadata:
        key1: some-value

    # This machine does NOT specify all keys, it is missing 'model' but will
    # have the 'model' under 'globals' inserted as its default.
    # And will train a model on one month as well.
    - name: some-name-here
      dataset:
        train_start_date: 2018-01-01T00:00:00Z
        train_end_date: 2018-02-01T00:00:00Z
        resolution: 2T  # Resample timeseries at 2min intervals (pandas freq strings)
        tags:
          - tag-1
          - tag-2
      metadata:
        key1: some-different-value-if-you-want
        nested-keys-allowed:
          - correct: true

  globals:
    model:
      sklearn.pipeline.Pipeline:
        steps:
          - sklearn.preprocessing.MinMaxScaler
          - gordo.model.models.KerasAutoEncoder:
              kind: feedforward_model

    metadata:
      what-does-this-do: "This metadata will get mapped to every machine's metadata!"
```

One can experiment locally with Gordo through the Jupyter Notebooks provided in the examples directory of the repository.

1.2 Architecture

`Gordo` is based on parsing a config file written in `yaml` that is converted into an `Argo` workflow. This is deployed with `ArgoCD` onto a `Kubernetes` cluster. The main interface after building the models is a set of `REST` APIs

For illustrating the architecture, we use the C4 approach.

1.3 Endpoints

1.3.1 Project index page

Going to the base path of the project, ie. `/gordo/v0/my-project/` will return the project level index, with returns a collection of the metadata surrounding the models currently deployed and their status. Each `endpoint` key has an associated `endpoint-metadata` key which is the direct transferal of metadata returned from the ML servers at their */metadata/* route.

This returns *a lot* of metadata data, so we'll show a small screen-shot of some of the data you might expect to get:

```
endpoint:                                  "/gordo/v0/test-milesg3/trolla-rowfilter-model"
endpoint-metadata:
  env:
    MODEL_COLLECTION_DIR:                  "/gordo/models/test-milesg3/1573224406946"
  gordo-server-version:                    "0.39.1.dev12+gbe3bbd3"
  metadata:
    dataset:
      asset:                               null
      data_provider:                       {…}
      filter:                              "(`TRA-35TT8566.PV` > 30) & (`TRA-35TT8566.PV` < 40)"
      resolution:                          "10T"
      row_filter_buffer_size:              0
      tag_list:                            […]
      target_tag_list:                     […]
      train_end_date:                      "2019-03-01 00:00:00+00:00"
      train_start_date:                    "2019-01-01 00:00:00+00:00"
    model:
      base_estimator:                      "TransformedTargetRegressor(check_inverse=True, func=None, inverse_func=None,\n
                                           feature_range=(0,\n
                                           verbose=False),\n                          transformer=MinMaxScaler(copy=True,\n
      cross-validation:
        cv-duration-sec:                   23.922932863235474
        scores:
          explained-variance-score:
            fold-1:                        0.9756029673831756
            fold-2:                        0.9879222541420302
            fold-3:                        0.8414066428008284
            fold-max:                      0.9879222541420302
            fold-mean:                     0.9349772881086782
            fold-min:                      0.8414066428008284
            fold-std:                      0.06635530852442416
          explained-variance-score-TRA-35TT8566.PV:   {…}
          explained-variance-score-TRA-35TT8567.PV:   {…}
          explained-variance-score-TRA-35TT8568.PV:   {…}
          explained-variance-score-TRA-35TT8569.PV:   {…}
          mean-absolute-error:             {…}
          mean-absolute-error-TRA-35TT8566.PV:        {…}
          mean-absolute-error-TRA-35TT8567.PV:        {…}
          mean-absolute-error-TRA-35TT8568.PV:        {…}
          mean-absolute-error-TRA-35TT8569.PV:        {…}
          mean-squared-error:              {…}
          mean-squared-error-TRA-35TT8566.PV:         {…}
          mean-squared-error-TRA-35TT8567.PV:         {…}
          mean-squared-error-TRA-35TT8568.PV:         {…}
          mean-squared-error-TRA-35TT8569.PV:         {…}
          r2-score:                        {…}
          r2-score-TRA-35TT8566.PV:        {…}
          r2-score-TRA-35TT8567.PV:        {…}
          r2-score-TRA-35TT8568.PV:        {…}
          r2-score-TRA-35TT8569.PV:        {…}
        data-query-duration-sec:           45.346792221069336
      history:                             {…}
      model-builder-version:               "0.39.1.dev12+gbe3bbd3"
      model-config:                        {…}
      model-creation-date:                 "2019-11-08 14:49:32.714039+00:00"
      model-offset:                        0
      model-training-duration-sec:         4.223615884780884
      scaler:                              "RobustScaler(copy=True, …     with_scaling=True)"
    name:                                  "trolla-rowfilter-model"
  user-defined:                            {…}
healthy:                                   true
last-checked:                              "2019-11-11T14:09:49.652247+00:00"
last-seen:                                 "2019-11-11T14:09:49.652261+00:00"
target:                                    "trolla-rowfilter-model"
```

1.3.2 Machine Learning Server Routes

When a model is deployed from a config file, it results in a ML server capable of the following paths:

Under normal Equinor deployments, paths listed below should be prefixed with `/gordo/v0/<project-name>/<model-name>`. Otherwise, the paths listed below are the raw exposed endpoints from the server's perspective.

1.3.2.1 /

This is the Swagger UI for the given model. Allows for manual testing of endpoints via a GUI interface.

1.3.2.2 /prediction/

The `/prediction` endpoint will return the basic values a model is capable of returning. Namely, this will be:

- **`model-output`:**
  - The raw model output, after calling `.predict` on the model or pipeline or `.transform` if the pipeline/model does not have a `.predict` method.
- **`original-input`:**
  - Represents the data supplied to the Pipeline, the raw untransformed values.

Sample response:

```
{'data': {'end': {'end': {'0': None, '1': None}},
      'model-input': {'TAG-1': {'0': 0.7149938815135232,
                                '1': 0.5804863352453888},
                      'TAG-2': {'0': 0.724091483437877,
                                '1': 0.9307866320901698},
                      'TAG-3': {'0': 0.018676439423681468,
                                '1': 0.3389969016787632},
                      'TAG-4': {'0': 0.285813103358881,
                                '1': 0.12008312306966606}},
      'model-output': {'TARGET-TAG-1': {'0': 31.12387466430664,
                                        '1': 31.12371063232422},
                       'TARGET-TAG-2': {'0': 30.122753143310547,
                                        '1': 30.122438430786133},
                       'TARGET-TAG-3': {'0': 20.38254737854004,
```

(continues on next page)

(continued from previous page)

```
                                          '1': 20.382972717285156}},
      'start': {'start': {'0': None, '1': None}}}}
```

The endpoint only accepts POST requests.

`POST` requests take raw data:

```
>>> import requests
>>>
>>> # Single sample:
>>> requests.post("https://my-server.io/prediction", json={"X": [1, 2,␣
↪3, 4]})
>>>
>>> # Multiple samples:
>>> requests.post("https://my-server.io/prediction", json={"X": [[1, 2,
↪ 3, 4], [5, 6, 7, 8]]})
```

NOTE: The client must provide the correct number of input features, ie. if the model was trained on 4 features, the client should provide 4 feature sample(s).

You may also supply a dataframe using *gordo.server.utils.dataframe_to_dict()*:

```
>>> import requests
>>> import pprint
>>> from gordo.server import utils
>>> import pandas as pd
>>> X = pd.DataFrame({"TAG-1": range(4),
...                   "TAG-2": range(4),
...                   "TAG-3": range(4),
...                   "TAG-4": range(4)},
...                   index=pd.date_range('2019-01-01', '2019-01-02',␣
↪periods=4)
... )
>>> resp = requests.post("https://my-server.io/gordo/v0/project-name/
↪model-name/prediction",
...                       json={"X": utils.dataframe_to_dict(X)}
... )
>>> pprint.pprint(resp.json())
{'data': {'end': {'end': {'2019-01-01 00:00:00': None,
                          '2019-01-01 08:00:00': None,
                          '2019-01-01 16:00:00': None,
                          '2019-01-02 00:00:00': None}},
      'model-input': {'TAG-1': {'2019-01-01 00:00:00': 0,
                                '2019-01-01 08:00:00': 1,
                                '2019-01-01 16:00:00': 2,
                                '2019-01-02 00:00:00': 3},
```

(continues on next page)

(continued from previous page)

```
                    'TAG-2': {'2019-01-01 00:00:00': 0,
                              '2019-01-01 08:00:00': 1,
                              '2019-01-01 16:00:00': 2,
                              '2019-01-02 00:00:00': 3},
                    'TAG-3': {'2019-01-01 00:00:00': 0,
                              '2019-01-01 08:00:00': 1,
                              '2019-01-01 16:00:00': 2,
                              '2019-01-02 00:00:00': 3},
                    'TAG-4': {'2019-01-01 00:00:00': 0,
                              '2019-01-01 08:00:00': 1,
                              '2019-01-01 16:00:00': 2,
                              '2019-01-02 00:00:00': 3}},
     'model-output': {'TARGET-TAG-1': {'2019-01-01 00:00:00': 31.
↪123781204223633,
                                       '2019-01-01 08:00:00': 31.
↪122915267944336,
                                       '2019-01-01 16:00:00': 31.
↪12187385559082,
                                       '2019-01-02 00:00:00': 31.
↪120620727539062},
                      'TARGET-TAG-2': {'2019-01-01 00:00:00': 30.
↪122575759887695,
                                       '2019-01-01 08:00:00': 30.
↪120899200439453,
                                       '2019-01-01 16:00:00': 30.
↪11887550354004,
                                       '2019-01-02 00:00:00': 30.
↪116445541381836},
                      'TARGET-TAG-3': {'2019-01-01 00:00:00': 20.
↪382783889770508,
                                       '2019-01-01 08:00:00': 20.
↪385055541992188,
                                       '2019-01-01 16:00:00': 20.
↪38779640197754,
                                       '2019-01-02 00:00:00': 20.
↪391088485717773}},
     'start': {'start': {'2019-01-01 00:00:00': '2019-01-01T00:00:00',
                         '2019-01-01 08:00:00': '2019-01-01T08:00:00',
                         '2019-01-01 16:00:00': '2019-01-01T16:00:00',
                         '2019-01-02 00:00:00': '2019-01-02T00:00:00'}
↪}}}
>>> # Alternatively, you can convert the json back into a dataframe␣
↪with:
>>> df = utils.dataframe_from_dict(resp.json())
```

Furthermore, you can increase efficiency by instead converting your data to parquet with the fol-

lowing:

```
>>> resp = requests.post("https://my-server.io/gordo/v0/project-name/
↪model-name/prediction?format=parquet",  # <- note the '?
↪format=parquet'
...                       files={"X": utils.dataframe_into_parquet_
↪bytes(X)}
... )
>>> resp.ok
True
>>> df = utils.dataframe_from_parquet_bytes(resp.content)
```

1.3.2.3 /anomaly/prediction/

The `/anomaly/prediction` endpoint will return the data supplied by the `/prediction` endpoint but reserved for models which inherit from `gordo.model.anomaly.base.AnomalyDetectorBase`

By this restriction, additional _features_ are calculated and returned (depending on the *Anomaly-Detector* model being served.

For example, the `gordo.model.anomaly.diff.DiffBasedAnomalyDetector` will return the following:

- **`tag-anomaly-scaled` & `tag-anomaly-unscaled`:**
  - Anomaly per feature/tag calculated from the expected tag input (y) and the model's output for those tags (yhat), using scaled and unscaled values.
- **`total-anomaly-scaled` & `total-anomaly-unscaled`:**
  - This is the total anomaly for the given point as calculated by the model, using scaled and unscaled values.

Sample response:

```
{'data': {'end': {'end': {'2019-01-01 00:00:00': '2019-01-01T00:10:00',
                          '2019-01-01 08:00:00': '2019-01-01T08:10:00',
                          '2019-01-01 16:00:00': '2019-01-01T16:10:00',
                          '2019-01-02 00:00:00': '2019-01-02T00:10:00'}
↪},
      'model-input': {'TAG-1': {'2019-01-01 00:00:00': 0,
                                '2019-01-01 08:00:00': 1,
                                '2019-01-01 16:00:00': 2,
                                '2019-01-02 00:00:00': 3},
                      'TAG-2': {'2019-01-01 00:00:00': 0,
                                '2019-01-01 08:00:00': 1,
```

(continues on next page)

(continued from previous page)

```
                                '2019-01-01 16:00:00': 2,
                                '2019-01-02 00:00:00': 3},
                    'TAG-3': {'2019-01-01 00:00:00': 0,
                              '2019-01-01 08:00:00': 1,
                              '2019-01-01 16:00:00': 2,
                              '2019-01-02 00:00:00': 3},
                    'TAG-4': {'2019-01-01 00:00:00': 0,
                              '2019-01-01 08:00:00': 1,
                              '2019-01-01 16:00:00': 2,
                              '2019-01-02 00:00:00': 3}},
     'model-output': {'TARGET-TAG-1': {'2019-01-01 00:00:00': 31.
↪123781204223633,
                                       '2019-01-01 08:00:00': 31.
↪122915267944336,
                                       '2019-01-01 16:00:00': 31.
↪12187385559082,
                                       '2019-01-02 00:00:00': 31.
↪120620727539062},
                      'TARGET-TAG-2': {'2019-01-01 00:00:00': 30.
↪122575759887695,
                                       '2019-01-01 08:00:00': 30.
↪120899200439453,
                                       '2019-01-01 16:00:00': 30.
↪11887550354004,
                                       '2019-01-02 00:00:00': 30.
↪116445541381836},
                      'TARGET-TAG-3': {'2019-01-01 00:00:00': 20.
↪382783889770508,
                                       '2019-01-01 08:00:00': 20.
↪385055541992188,
                                       '2019-01-01 16:00:00': 20.
↪38779640197754,
                                       '2019-01-02 00:00:00': 20.
↪391088485717773}},
     'start': {'start': {'2019-01-01 00:00:00': '2019-01-01T00:00:00',
                         '2019-01-01 08:00:00': '2019-01-01T08:00:00',
                         '2019-01-01 16:00:00': '2019-01-01T16:00:00',
                         '2019-01-02 00:00:00': '2019-01-02T00:00:00'}
↪},
     'tag-anomaly-scaled': {'TARGET-TAG-1': {'2019-01-01 00:00:00':␣
↪43.9791088965509,
                                             '2019-01-01 08:00:00':␣
↪42.564846544761124,
                                             '2019-01-01 16:00:00':␣
↪41.15033623847873,
```

(continues on next page)

(continued from previous page)

```
                                                    '2019-01-02 00:00:00':␣
↪39.73552676971069},
                                'TARGET-TAG-2': {'2019-01-01 00:00:00':␣
↪42.73147969197182,
                                                    '2019-01-01 08:00:00':␣
↪41.310514834943056,
                                                    '2019-01-01 16:00:00':␣
↪39.88905753340811,
                                                    '2019-01-02 00:00:00':␣
↪38.46702390945659},
                                'TARGET-TAG-3': {'2019-01-01 00:00:00':␣
↪26.2922285259887,
                                                    '2019-01-01 08:00:00':␣
↪25.005235450434874,
                                                    '2019-01-01 16:00:00':␣
↪23.71884761692332,
                                                    '2019-01-02 00:00:00':␣
↪22.43317081979476}},
     'total-anomaly-scaled': {'total-anomaly-scaled': {'2019-01-01␣
↪00:00:00': 66.71898273252445,
                                                       '2019-01-01␣
↪08:00:00': 64.37069672792737,
                                                       '2019-01-01␣
↪16:00:00': 62.024759698996235,
                                                       '2019-01-02␣
↪00:00:00': 59.68141393388054}}},
'time-seconds': '0.1623'}
```

This endpoint accepts only `POST` requests. Model requests are exactly the same as */prediction/*, but will require a `y` to compare the anomaly against.

1.3.2.4 /download-model/

Returns the current model being served. Loadable via `gordo.serializer.loads(downloaded_bytes)`

1.3.2.5 /metadata/

Various metadata surrounding the current model and environment.

1.4 Machine

A Machine is the central unity of a model, dataset, metadata and everything needed to create and build a ML model to be served by a deployment.

An example of a `Machine` in the context of a YAML config, could be the following:

```
- name: ct-23-0001
  dataset:
    tags:
      - TAG 1
      - TAG 2
      - TAG 3
    train_start_date: 2016-11-07T09:11:30+01:00
    train_end_date: 2018-09-15T03:01:00+01:00
  metadata:
    arbitrary-key: arbitrary-value
  model:
    gordo.machine.model.anomaly.diff.DiffBasedAnomalyDetector:
      base_estimator:
        sklearn.pipeline.Pipeline:
          steps:
            - sklearn.preprocessing.MinMaxScaler
            - gordo.machine.model.models.KerasAutoEncoder:
                kind: feedforward_hourglass
```

And to construct this into a python object:

```
>>> from gordo.machine import Machine
>>> # `config` is the result of the parsed and loaded yaml element␣
↪above
>>> machine = Machine.from_config(config, project_name='test-proj')
>>> machine.name
ct-23-0001
```

class gordo.machine.machine.**Machine**(*name: str, model: dict, dataset: Union[gordo_dataset.base.GordoBaseDataset, dict], project_name: str, evaluation: Optional[dict] = None, metadata: Union[dict, gordo.machine.metadata.metadata.Metadata, None] = None, runtime=None*)

Bases: `object`

Represents a single machine in a config file

dataset
Descriptor for attributes requiring type `gordo.workflow.config_elements.Dataset`

classmethod from_config(*config: Dict[str, Any]*, *project_name: str*, *config_globals=None*)
Construct an instance from a block of YAML config file which represents a single Machine; loaded as a `dict`.

Parameters

- **config** (*dict*) – The loaded block of config which represents a 'Machine' in YAML
- **project_name** (*str*) – Name of the project this Machine belongs to.
- **config_globals** – The block of config within the YAML file within *globals*

Returns

Return type *Machine*

classmethod from_dict(*d: dict*) → gordo.machine.machine.Machine
Get an instance from a dict taken from *to_dict()*

host
Descriptor for use in objects which require valid URL values. Where 'valid URL values' is Gordo's version: alphanumeric with dashes.

Use:

```
class MySpecialClass:

    url_attribute = ValidUrlString()

    ...

myspecialclass = MySpecialClass()

myspecialclass.url_attribute = 'this-is-ok'
myspecialclass.url_attribute = 'this will r@ise a ValueError'
```

metadata
Descriptor for attributes requiring type Optional[dict]

model
Descriptor for attributes requiring type Union[dict, str]

name
Descriptor for use in objects which require valid URL values. Where 'valid URL values' is Gordo's version: alphanumeric with dashes.

Use:

```
class MySpecialClass:

    url_attribute = ValidUrlString()

    ...

myspecialclass = MySpecialClass()

myspecialclass.url_attribute = 'this-is-ok'
myspecialclass.url_attribute = 'this will r@ise a ValueError'
```

normalize_sensor_tags(*tag_list: List[Union[Dict, List, str, gordo_dataset.sensor_tag.SensorTag]]*) → List[gordo_dataset.sensor_tag.SensorTag]
Finding assets for all of the tags according to information from the dataset metadata

Parameters **tag_list** (*TagsList*) –

Returns

Return type List[SensorTag]

project_name
Descriptor for use in objects which require valid URL values. Where 'valid URL values' is Gordo's version: alphanumeric with dashes.

Use:

```
class MySpecialClass:

    url_attribute = ValidUrlString()

    ...

myspecialclass = MySpecialClass()

myspecialclass.url_attribute = 'this-is-ok'
myspecialclass.url_attribute = 'this will r@ise a ValueError'
```

report()
Run any reporters in the machine's runtime for the current state.

Reporters implement the `gordo.reporters.base.BaseReporter` and can be specified in a config file of the machine for example:

```
runtime:
  reporters:
    - gordo.reporters.postgres.PostgresReporter:
        host: my-special-host
```

runtime
Descriptor for runtime dict in a machine object. Must be a valid runtime, but also must contain server.resources.limits/requests.memory/cpu to be valid.

to_dict()
Convert to a `dict` representation along with all attributes which can also be converted to a `dict`. Can reload with *from_dict()*

class gordo.machine.machine.**MachineEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

A JSONEncoder for machine objects, handling datetime.datetime objects as strings and handles any numpy numeric instances; both of which common in the `dict` representation of a `Machine`

Example

```
>>> from pytz import UTC
>>> s = json.dumps({"now":datetime.now(tz=UTC)},␣
↪cls=MachineEncoder, indent=4)
>>> s = '{"now": "2019-11-22 08:34:41.636356+"}'
```

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

1.4.1 Descriptors

Collection of descriptors to verify types and conditions of the Machine attributes when loading.

And example of which is if the machine name is set to a value which isn't a valid URL string, thus causing early failure before k8s itself discovers that the name isn't valid. (See: *gordo.machine.validators.ValidUrlString*)

class gordo.machine.validators.**BaseDescriptor**

Bases: `object`

Base descriptor class

New object should override __set__(self, instance, value) method to check if 'value' meets required needs.

class gordo.machine.validators.**ValidDataProvider**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for DataProvider

class gordo.machine.validators.**ValidDataset**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring type `gordo.workflow.config_elements.Dataset`

class gordo.machine.validators.**ValidDatasetKwargs**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring type `gordo.workflow.config_elements.Dataset`

class gordo.machine.validators.**ValidDatetime**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring valid datetime.datetime attribute

class gordo.machine.validators.**ValidMachineRuntime**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for runtime dict in a machine object. Must be a valid runtime, but also must contain server.resources.limits/requests.memory/cpu to be valid.

class gordo.machine.validators.**ValidMetadata**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring type Optional[dict]

class gordo.machine.validators.**ValidModel**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring type Union[dict, str]

class gordo.machine.validators.**ValidTagList**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for attributes requiring a non-empty list of strings

class gordo.machine.validators.**ValidUrlString**
Bases: *gordo.machine.validators.BaseDescriptor*

Descriptor for use in objects which require valid URL values. Where 'valid URL values' is Gordo's version: alphanumeric with dashes.

Use:

```
class MySpecialClass:
```

(continues on next page)

(continued from previous page)

```
    url_attribute = ValidUrlString()

    ...

myspecialclass = MySpecialClass()

myspecialclass.url_attribute = 'this-is-ok'
myspecialclass.url_attribute = 'this will r@ise a ValueError'
```

static valid_url_string(*string: str*) → bool

What we (Gordo) deem to be a suitable URL is the same as kubernetes lowercase alphanumeric with dashes but not ending or starting with a dash

Parameters **string** (*str - String to check*) –

Returns

Return type bool

gordo.machine.validators.**fix_resource_limits**(*resources: dict*) → dict

Resource limitations must be higher or equal to resource requests, if they are both specified. This bumps any limits to the corresponding request if they are both set.

Parameters **resources** (*dict*) – Dictionary with possible requests/limits

Examples

```
>>> fix_resource_limits({"requests": {"cpu": 10}, "limits":{"cpu
↪":9}})
{'requests': {'cpu': 10}, 'limits': {'cpu': 10}}
>>> fix_resource_limits({"requests": {"cpu": 10}})
{'requests': {'cpu': 10}}
```

Returns A copy of *resource_dict* with the any limits bumped to the corresponding request if they are both set.

Return type dict

gordo.machine.validators.**fix_runtime**(*runtime_dict*)

A valid runtime description must satisfy that any resource description must have that limit >= requests. This function will bump any limits that is too low.

1.4.2 Models

Models are a collection of Scikit-Learn like models, built specifically to fulfill a need. One example of which is the `KerasAutoEncoder`.

Other scikit-learn compliant models can be used within the config files without any additional configuration.

1.4.2.1 Base Model

The base model is designed to be inherited from any other models which need to be implemented within Gordo due to special model requirements. ie. PyTorch, Keras, etc.

class gordo.machine.model.base.**GordoBase**(***kwargs*)

Bases: abc.ABC

Initialize the model

abstract get_metadata()

Get model specific metadata, if any

abstract get_params(*deep=False*)

Return a dict containing all parameters used to initialized object

abstract score(*X: Union[numpy.ndarray, pandas.core.frame.DataFrame], y: Union[numpy.ndarray, pandas.core.frame.DataFrame], sample_weight: Optional[numpy.ndarray] = None*)

Score the model; must implement the correct default scorer based on model type

1.4.2.2 Custom Gordo models

This group of models are already implemented and ready to be used within config files, by simply specifying their full path. For example: `gordo.machine.model.models.KerasAutoEncoder`

class gordo.machine.model.models.**KerasAutoEncoder**(*kind: Union[str, Callable[[int, Dict[str, Any]], tensorflow.keras.models.Model]], **kwargs*)

Bases: *gordo.machine.model.models.KerasBaseEstimator*, sklearn.base.TransformerMixin

Subclass of the KerasBaseEstimator to allow fitting to just X without requiring y.

Initialized a Scikit-Learn API compatitble Keras model with a pre-registered function or a builder function directly.

Parameters

- **kind** (*Union[callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo_compontents.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*
- **kwargs** (*dict*) – Any additional args which are passed to the factory building function and/or any additional args to be passed to Keras' fit() method

score(*X: Union[numpy.ndarray, pandas.core.frame.DataFrame], y: Union[numpy.ndarray, pandas.core.frame.DataFrame], sample_weight: Optional[numpy.ndarray] = None*) → float

Returns the explained variance score between auto encoder's input vs output

Parameters

- **X** (*Union[np.ndarray, pd.DataFrame]*) – Input data to the model
- **y** (*Union[np.ndarray, pd.DataFrame]*) – Target
- **sample_weight** (*Optional[np.ndarray]*) – sample weights

Returns **score** – Returns the explained variance score

Return type float

class gordo.machine.model.models.**KerasBaseEstimator**(*kind: Union[str, Callable[[int, Dict[str, Any]], tensorflow.keras.models.Model]], **kwargs*)

Bases: tensorflow.keras.wrappers.scikit_learn.KerasRegressor, *gordo.machine.model.base.GordoBase*, sklearn.base.BaseEstimator

Initialized a Scikit-Learn API compatitble Keras model with a pre-registered function or a builder function directly.

Parameters

- **kind** (*Union[callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo_compontents.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such

a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*

- **kwargs** (*dict*) – Any additional args which are passed to the factory building function and/or any additional args to be passed to Keras' fit() method

classmethod extract_supported_fit_args(*kwargs*)
Filtering only `fit` related kwargs

Parameters **kwargs** (*dict*) –

fit(*X: Union[numpy.ndarray, pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], y: Union[numpy.ndarray, pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], **kwargs*)
Fit the model to X given y.

Parameters

- **X** (*Union[np.ndarray, pd.DataFrame, xr.Dataset]*) – numpy array or pandas dataframe
- **y** (*Union[np.ndarray, pd.DataFrame, xr.Dataset]*) – numpy array or pandas dataframe
- **sample_weight** (*np.ndarray*) – array like - weight to assign to samples
- **kwargs** – Any additional kwargs to supply to keras fit method.

Returns 'KerasAutoEncoder'

Return type self

classmethod from_definition(*definition: dict*)
Handler for `gordo.serializer.from_definition`

Parameters **definition** (*dict*) –

get_metadata()
Get metadata for the KerasBaseEstimator. Includes a dictionary with key "history". The key's value is a a dictionary with a key "params" pointing another dictionary with various parameters. The metrics are defined in the params dictionary under "metrics". For each of the metrics there is a key who's value is a list of values for this metric per epoch.

Returns Metadata dictionary, including a history object if present

Return type Dict

static get_n_features(*X: Union[numpy.ndarray, pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]*) → Union[int, tuple]

static get_n_features_out(*y: Union[numpy.ndarray, pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]*) → Union[int, tuple]

get_params(***params*)

Gets the parameters for this estimator

Parameters **params** – ignored (exists for API compatibility).

Returns Parameters used in this estimator

Return type Dict[str, Any]

into_definition() → dict

Handler for `gordo.serializer.into_definition`

Returns

Return type dict

load_kind(*kind*)

static parse_module_path(*module_path*) → Tuple[Optional[str], str]

predict(*X: numpy.ndarray*, ***kwargs*) → numpy.ndarray

Parameters

- **X** (*np.ndarray*) – Input data
- **kwargs** (*dict*) – kwargs which are passed to Kera's `predict` method

Returns np.ndarray

Return type results

property sk_params

Parameters used for scikit learn kwargs

supported_fit_args = ['batch_size', 'epochs', 'verbose', 'callbacks', '

class gordo.machine.model.models.**KerasLSTMAutoEncoder**(*kind: Union[Callable, str], lookback_window: int = 1, batch_size: int = 32, **kwargs*)

Bases: *gordo.machine.model.models.KerasLSTMBaseEstimator*

Parameters

- **kind** (*Union[Callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo.machine.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*.
- **lookback_window** (*int*) – Number of timestamps (lags) used to train the model.
- **batch_size** (*int*) – Number of training examples used in one epoch.
- **epochs** (*int*) – Number of epochs to train the model. An epoch is an iteration over the entire data provided.
- **verbose** (*int*) – Verbosity mode. Possible values are 0, 1, or 2 where 0 = silent, 1 = progress bar, 2 = one line per epoch.
- **kwargs** (*dict*) – Any arguments which are passed to the factory building function and/or any additional args to be passed to the intermediate fit method.

property lookahead

Steps ahead in y the model should target

class gordo.machine.model.models.**KerasLSTMBaseEstimator**(*kind: Union[Callable, str], lookback_window: int = 1, batch_size: int = 32, **kwargs*)

Bases: *gordo.machine.model.models.KerasBaseEstimator*, sklearn.base.TransformerMixin

Abstract Base Class to allow to train a many-one LSTM autoencoder and an LSTM 1 step forecast

Parameters

- **kind** (*Union[Callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo.machine.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*.
- **lookback_window** (*int*) – Number of timestamps (lags) used to train the model.
- **batch_size** (*int*) – Number of training examples used in one epoch.
- **epochs** (*int*) – Number of epochs to train the model. An epoch is an iteration over the entire data provided.
- **verbose** (*int*) – Verbosity mode. Possible values are 0, 1, or 2 where 0 = silent, 1 = progress bar, 2 = one line per epoch.
- **kwargs** (*dict*) – Any arguments which are passed to the factory building function and/or any additional args to be passed to the intermediate fit method.

fit (*X: numpy.ndarray*, *y: numpy.ndarray*, ***kwargs*) → gordo.machine.model.models.KerasLSTMForecast

This fits a one step forecast LSTM architecture.

Parameters

- **X** (*np.ndarray*) – 2D numpy array of dimension n_samples x n_features. Input data to train.
- **y** (*np.ndarray*) – 2D numpy array representing the target
- **kwargs** (*dict*) – Any additional args to be passed to Keras *fit_generator* method.

Returns KerasLSTMForecast

Return type class

get_metadata ()

Add number of forecast steps to metadata

Returns **metadata** – Metadata dictionary, including forecast steps.

Return type dict

abstract property lookahead

Steps ahead in y the model should target

predict (*X: numpy.ndarray*, ***kwargs*) → numpy.ndarray

Parameters **X** (*np.ndarray*) – Data to predict/transform. 2D numpy array of dimension *n_samples x n_features* where *n_samples* must be > lookback_window.

Returns results – 2D numpy array of dimension *(n_samples - lookback_window) x 2*n_features*. The first half of the array *(results[:, :n_features])* corresponds to X offset by *lookback_window+1* (i.e., *X[lookback_window:,:]*) whereas the second half corresponds to the predicted values of *X[lookback_window:,:]*.

Return type np.ndarray

Example

```
>>> import numpy as np
>>> from gordo.machine.model.factories.lstm_autoencoder import␣
↪lstm_model
>>> from gordo.machine.model.models import KerasLSTMForecast
>>> #Define train/test data
>>> X_train = np.array([[1, 1], [2, 3], [0.5, 0.6], [0.3, 1],␣
↪[0.6, 0.7]])
>>> X_test = np.array([[2, 3], [1, 1], [0.1, 1], [0.5, 2]])
>>> #Initiate model, fit and transform
>>> lstm_ae = KerasLSTMForecast(kind="lstm_model",
...                             lookback_window=2,
...                             verbose=0)
>>> model_fit = lstm_ae.fit(X_train, y=X_train.copy())
>>> model_transform = lstm_ae.predict(X_test)
>>> model_transform.shape
(2, 2)
```

score (*X: Union[numpy.ndarray, pandas.core.frame.DataFrame]*, *y: Union[numpy.ndarray, pandas.core.frame.DataFrame]*, *sample_weight: Optional[numpy.ndarray] = None*) → float

Returns the explained variance score between 1 step forecasted input and true input at next time step (note: for LSTM X is offset by *lookback_window*).

Parameters

- **X** (*Union[np.ndarray, pd.DataFrame]*) – Input data to the model.
- **y** (*Union[np.ndarray, pd.DataFrame]*) – Target
- **sample_weight** (*Optional[np.ndarray]*) – Sample weights

Returns **score** – Returns the explained variance score.

Return type float

class gordo.machine.model.models.**KerasLSTMForecast**(*kind: Union[Callable, str], lookback_window: int = 1, batch_size: int = 32, **kwargs*)

Bases: *gordo.machine.model.models.KerasLSTMBaseEstimator*

Parameters

- **kind** (*Union[Callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo.machine.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*.
- **lookback_window** (*int*) – Number of timestamps (lags) used to train the model.
- **batch_size** (*int*) – Number of training examples used in one epoch.
- **epochs** (*int*) – Number of epochs to train the model. An epoch is an iteration over the entire data provided.
- **verbose** (*int*) – Verbosity mode. Possible values are 0, 1, or 2 where 0 = silent, 1 = progress bar, 2 = one line per epoch.
- **kwargs** (*dict*) – Any arguments which are passed to the factory building function and/or any additional args to be passed to the intermediate fit method.

property lookahead

Steps ahead in y the model should target

class gordo.machine.model.models.**KerasRawModelRegressor**(*kind: Union[str, Callable[[int, Dict[str, Any]], tensorflow.keras.models.Model]], **kwargs*)

Bases: *gordo.machine.model.models.KerasAutoEncoder*

Create a scikit-learn like model with an underlying tensorflow.keras model from a raw config.

.. rubric:: Examples

```
>>> import yaml
>>> import numpy as np
>>> config_str = '''
...   # Arguments to the .compile() method
...   compile:
...     loss: mse
...     optimizer: adam
...
...   # The architecture of the model itself.
...   spec:
...     tensorflow.keras.models.Sequential:
...       layers:
...         - tensorflow.keras.layers.Dense:
...             units: 4
...         - tensorflow.keras.layers.Dense:
...             units: 1
... '''
>>> config = yaml.safe_load(config_str)
>>> model = KerasRawModelRegressor(kind=config)
>>>
>>> X, y = np.random.random((10, 4)), np.random.random((10, 1))
>>> model.fit(X, y, verbose=0)
KerasRawModelRegressor(kind: {'compile': {'loss': 'mse', 'optimizer
↪': 'adam'},
 'spec': {'tensorflow.keras.models.Sequential': {'layers': [{
↪'tensorflow.keras.layers.Dense': {'units': 4}},
                                                            {
↪'tensorflow.keras.layers.Dense': {'units': 1}}]}}})
>>> out = model.predict(X)
```

Initialized a Scikit-Learn API compatitble Keras model with a pre-registered function or a builder function directly.

Parameters

- **kind** (*Union[callable, str]*) – The structure of the model to build. As designated by any registered builder functions, registered with *gordo_compontents.model.register.register_model_builder*. Alternatively, one may pass a builder function directly to this argument. Such a function should accept *n_features* as it's first argument, and pass any additional parameters to ***kwargs*
- **kwargs** (*dict*) – Any additional args which are passed to the factory building function and/or any additional args to be passed to Keras' fit() method

load_kind(*kind*)

gordo.machine.model.models.**create_keras_timeseriesgenerator**(*X: numpy.ndarray, y: Optional[numpy.ndarray] batch_size: int, lookback_window: int, lookahead: int*) → tensorflow.keras.preprocessin

Provides a *keras.preprocessing.sequence.TimeseriesGenerator* for use with LSTM's, but with the added ability to specify the lookahead of the target in y.

If lookahead==0 then the generated samples in X will have as their last element the same as the corresponding Y. If lookahead is 1 then the values in Y is shifted so it is one step in the future compared to the last value in the samples in X, and similar for larger values.

Parameters

- **X** (*np.ndarray*) – 2d array of values, each row being one sample.
- **y** (*Optional[np.ndarray]*) – array representing the target.
- **batch_size** (*int*) – How big should the generated batches be?
- **lookback_window** (*int*) – How far back should each sample see. 1 means that it contains a single measurement
- **lookahead** (*int*) – How much is Y shifted relative to X

Returns 3d matrix with a list of batchX-batchY pairs, where batchX is a batch of X-values, and correspondingly for batchY. A batch consist of *batch_size* nr of pairs of samples (or y-values), and each sample is a list of length *lookback_window*.

Return type TimeseriesGenerator

Examples

```
>>> import numpy as np
>>> X, y = np.random.rand(100,2), np.random.rand(100, 2)
>>> gen = create_keras_timeseriesgenerator(X, y,
...                                        batch_size=10,
...                                        lookback_window=20,
...                                        lookahead=0)
>>> len(gen) # 9 = (100-20+1)/10
9
>>> len(gen[0]) # batchX and batchY
2
>>> len(gen[0][0]) # batch_size=10
10
>>> len(gen[0][0][0]) # a single sample, lookback_window = 20,
20
>>> len(gen[0][0][0][0]) # n_features = 2
2
```

Model factories

Model factories are stand alone functions which take an arbitrary number of primitive parameters (int, float, list, dict, etc) and return a model which can then be used in the `kind` parameter of some Scikit-Learn like wrapper model.

An example of this is `KerasAutoEncoder` which accepts a `kind` argument (as all custom gordo models do) and can be given *feedforward_model*. Meaning that function will be used to create the underlying Keras model for `KerasAutoEncoder`

feedforward factories

gordo.machine.model.factories.feedforward_autoencoder.**feedforward_hourglass**

deeper into the encoder network and increasing number of neurons as one gets out of the decoder network.

Parameters

- **n_features** (*int*) – Number of input and output neurons.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **encoding_layers** (*int*) – Number of layers from the input layer (exclusive) to the narrowest layer (inclusive). Must be > 0. The total nr of layers including input and output layer will be 2*encoding_layers + 1.
- **compression_factor** (*float*) – How small the smallest layer is as a ratio of n_features (smallest layer is rounded up to nearest integer). Must satisfy 0 <= compression_factor <= 1.
- **func** (*str*) – Activation function for the internal layers
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimization_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Notes

The resulting model will look like this when n_features = 10, encoding_layers= 3, and compression_factor = 0.3:

```
* * * * * * * * * * *
  * * * * * * * * *
     * * * * * *
        * * *
        * * *
     * * * * * *
  * * * * * * * * *
* * * * * * * * * * *
```

Returns

Return type keras.models.Sequential

Examples

```
>>> model = feedforward_hourglass(10)
>>> len(model.layers)
7
>>> [model.layers[i].units for i in range(len(model.layers))]
[8, 7, 5, 5, 7, 8, 10]
>>> model = feedforward_hourglass(5)
>>> [model.layers[i].units for i in range(len(model.layers))]
[4, 4, 3, 3, 4, 4, 5]
>>> model = feedforward_hourglass(10, compression_factor=0.2)
>>> [model.layers[i].units for i in range(len(model.layers))]
[7, 5, 2, 2, 5, 7, 10]
>>> model = feedforward_hourglass(10, encoding_layers=1)
>>> [model.layers[i].units for i in range(len(model.layers))]
[5, 5, 10]
```

gordo.machine.model.factories.feedforward_autoencoder.**feedforward_model**(*n_fea*
int,
n_fea
int
=
None
en-
cod-
ing_d
Tu-
ple[i
...]
=
(256,
128,
64),
en-
cod-
ing_f
Tu-
ple[s
...]
=
('tanh
'tanh
'tanh
de-
cod-
ing_d
Tu-
ple[i
...]
=
(64,
128,
256),
de-
cod-
ing_f
Tu-
ple[s
...]
=
('tanh
'tanh
'tanh
out_f
str
=
'lin-

Builds a customized keras neural network auto-encoder based on a config dict

Parameters

- **n_features** (*int*) – Number of features the dataset X will contain.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **encoding_dim** (*tuple*) – Tuple of numbers with the number of neurons in the encoding part.
- **decoding_dim** (*tuple*) – Tuple of numbers with the number of neurons in the decoding part.
- **encoding_func** (*tuple*) – Activation functions for the encoder part.
- **decoding_func** (*tuple*) – Activation functions for the decoder part.
- **out_func** (*str*) – Activation function for the output layer
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimize_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Returns

Return type keras.models.Sequential

gordo.machine.model.factories.feedforward_autoencoder.**feedforward_symmetric**

Builds a symmetrical feedforward model

Parameters

- **n_features** (*int*) – Number of input and output neurons.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **dim** (*List[int]*) – Number of neurons per layers for the encoder, reversed for the decoder. Must have len > 0.
- **funcs** (*List[str]*) – Activation functions for the internal layers
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimization_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Returns

Return type keras.models.Sequential

lstm factories

gordo.machine.model.factories.lstm_autoencoder.**lstm_hourglass**(*n_features: int*, *n_features_out: int = None*, *lookback_window: int = 1*, *encoding_layers: int = 3*, *compression_factor: float = 0.5*, *func: str = 'tanh'*, *out_func: str = 'linear'*, *optimizer: Union[str, tensorflow.keras.optimizer = 'Adam'*, *optimizer_kwargs: Dict[str, Any] =*

deeper into the encoder network and increasing number of neurons as one gets out of the decoder network.

Parameters

- **n_features** (*int*) – Number of input and output neurons.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **encoding_layers** (*int*) –

  Number of layers from the input layer (exclusive) to the narrowest layer (inclusive). Must be > 0. The total nr of layers including input and output layer will be 2*encoding_layers + 1.

  compression_factor: float How small the smallest layer is as a ratio of n_features (smallest layer is rounded up to nearest integer). Must satisfy 0 <= compression_factor <= 1.
- **func** (*str*) – Activation function for the internal layers.
- **out_func** (*str*) – Activation function for the output Dense layer.
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimization_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Returns

Return type keras.models.Sequential

Examples

```
>>> model = lstm_hourglass(10)
>>> len(model.layers)
7
>>> [model.layers[i].units for i in range(len(model.layers))]
[8, 7, 5, 5, 7, 8, 10]
>>> model = lstm_hourglass(5)
>>> [model.layers[i].units for i in range(len(model.layers))]
```

(continues on next page)

(continued from previous page)

```
[4, 4, 3, 3, 4, 4, 5]
>>> model = lstm_hourglass(10, compression_factor=0.2)
>>> [model.layers[i].units for i in range(len(model.layers))]
[7, 5, 2, 2, 5, 7, 10]
>>> model = lstm_hourglass(10, encoding_layers=1)
>>> [model.layers[i].units for i in range(len(model.layers))]
[5, 5, 10]
```

gordo.machine.model.factories.lstm_autoencoder.**lstm_model**(*n_features: int*, *n_features_out: int = None*, *lookback_window: int = 1*, *encoding_dim: Tuple[int, ...] = (256, 128, 64)*, *encoding_func: Tuple[str, ...] = ('tanh', 'tanh', 'tanh')*, *decoding_dim: Tuple[int, ...] = (64, 128, 256)*, *decoding_func: Tuple[str, ...] = ('tanh', 'tanh', 'tanh')*

Builds a customized Keras LSTM neural network auto-encoder based on a config dict.

Parameters

- **n_features** (*int*) – Number of features the dataset X will contain.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **lookback_window** (*int*) – Number of timesteps used to train the model. One timestep = current observation in the sample. Two timesteps = current observation + previous observation in the sample. ...
- **encoding_dim** (*tuple*) – Tuple of numbers with the number of neurons in the encoding part.
- **decoding_dim** (*tuple*) – Tuple of numbers with the number of neurons in the decoding part.
- **encoding_func** (*tuple*) – Activation functions for the encoder part.
- **decoding_func** (*tuple*) – Activation functions for the decoder part.
- **out_func** (*str*) – Activation function for the output Dense layer.
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimize_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Returns Returns Keras sequential model.

Return type keras.models.Sequential

gordo.machine.model.factories.lstm_autoencoder.**lstm_symmetric**(*n_features: int*, *n_features_out: int = None*, *lookback_window: int = 1*, *dims: Tuple[int, ...] = (256, 128, 64)*, *funcs: Tuple[str, ...] = ('tanh', 'tanh', 'tanh')*, *out_func: str = 'linear'*, *optimizer: Union[str, tensorflow.keras.optimizer = 'Adam'*, *optimizer_kwargs: Dict[str, Any] = {}*, *compile_kwargs:*

Builds a symmetrical lstm model

Parameters

- **n_features** (*int*) – Number of input and output neurons.
- **n_features_out** (*Optional[int]*) – Number of features the model will output, default to `n_features`.
- **lookback_window** (*int*) – Number of timesteps used to train the model. One timestep = sample contains current observation. Two timesteps = sample contains current and previous observation. ...
- **dims** (*Tuple[int, ...]*) – Number of neurons per layers for the encoder, reversed for the decoder. Must have len > 0
- **funcs** (*List[str]*) – Activation functions for the internal layers.
- **out_func** (*str*) – Activation function for the output Dense layer.
- **optimizer** (*Union[str, Optimizer]*) – If str then the name of the optimizer must be provided (e.x. "Adam"). The arguments of the optimizer can be supplied in optimization_kwargs. If a Keras optimizer call the instance of the respective class (e.x. Adam(lr=0.01,beta_1=0.9, beta_2=0.999)). If no arguments are provided Keras default values will be set.
- **optimizer_kwargs** (*Dict[str, Any]*) – The arguments for the chosen optimizer. If not provided Keras' default values will be used.
- **compile_kwargs** (*Dict[str, Any]*) – Parameters to pass to `keras.Model.compile`.

Returns Returns Keras sequential model.

Return type keras.models.Sequential

Transformer Functions

A collection of functions which can be referenced within the `sklearn.preprocessing.FunctionTransformer` transformer.

General

Functions to be used within sklearn's FunctionTransformer https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html

Each function SHALL take an X, and optionally a y.

Functions CAN take additional arguments which should be given during the initialization of the FunctionTransformer

Example:

```
>>> from sklearn.preprocessing import FunctionTransformer
>>> import numpy as np
>>> def my_function(X, another_arg):
...     # Some fancy X manipulation...
...     return X
>>> transformer = FunctionTransformer(func=my_function, kw_args={
↪'another_arg': 'this thing'})
>>> out = transformer.fit_transform(np.random.random(100).reshape(10,␣
↪10))
```

gordo.machine.model.transformer_funcs.general.**multiply_by**(*X*, *factor*)

Multiplies X by a given factor

Transformers

Specialized transformers to address Gordo specific problems. This function just like Scikit-Learn's transformers and thus can be inserted into `Pipeline` objects.

Imputers

class gordo.machine.model.transformers.imputer.**InfImputer**(*inf_fill_value=None*, *neg_inf_fill_value=None*, *strategy='minmax'*, *delta: float = 2.0*)

Bases: `sklearn.base.TransformerMixin`

Fill inf/-inf values of a 2d array/dataframe with imputed or provided values By default it will find the min and max of each feature/column and fill -infs/infs with those values +/- `delta`

Parameters

- **inf_fill_value** (*numeric*) – Value to fill 'inf' values
- **neg_inf_fill_value** (*numeric*) – Value to fill '-inf' values
- **strategy** (*str*) – How to fill values, irrelevant if fill value is provided. choices: 'extremes', 'minmax' -'extremes' will use the min and max values for the current datatype. such that 'inf' in a float32 dataset will have float32's largest value inserted. - 'minmax' will look at the min and max values in the feature where the -inf / inf appears and fill with the max/min found in that feature.
- **delta** (*float*) – Only applicable if `strategy='minmax'` Will add/subtract the max/min value, by feature, by this delta. If the max value in a feature was 10 and `delta=2` any inf value will be filled with 12. Likewise, if the min feature was -10 any -inf will be filled with -12.

fit(*X: Union[pandas.core.frame.DataFrame, numpy.ndarray], y=None*)

get_params(*deep=True*)

transform(*X: Union[pandas.core.frame.DataFrame, numpy.ndarray], y=None*)

Anomaly Models

Models which implment a `.anomaly(X, y)` and can be served under the model server `/anomaly/prediction` endpoint.

AnomalyDetectorBase

The base class for all other anomaly detector models

class gordo.machine.model.anomaly.base.**AnomalyDetectorBase**(***kwargs*)

Bases: `sklearn.base.BaseEstimator`, *gordo.machine.model.base.GordoBase*

Initialize the model

abstract anomaly(*X: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], y: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], frequency: Optional[datetime.timedelta] = None*) → Union[pandas.core.frame.DataFrame, xarray.core.dataset.Dataset]

Take X, y and optionally frequency; returning a dataframe containing anomaly score(s)

DiffBasedAnomalyDetector

Calculates the absolute value prediction differences between y and yhat as well as the absolute difference error between both matrices via `numpy.linalg.norm(..., axis=1)`

class gordo.machine.model.anomaly.diff.**DiffBasedAnomalyDetector**(*base_estimator: sklearn.base.Bas = ten- sor- flow.keras.wrapp scaler: sklearn.base.Tra = Min- MaxS- caler(), re- quire_thresholds bool = True, shuf- fle: bool = False, win- dow: Op- tional[int] = None, smooth- ing_method: Op- tional[str] = None*)

Bases: *gordo.machine.model.anomaly.base.AnomalyDetectorBase*

Estimator which wraps a `base_estimator` and provides a diff error based approach to anomaly detection.

It trains a `scaler` to the target **after** training, purely for error calculations. The underlying `base_estimator` is trained with the original, unscaled, `y`.

Threshold calculation is based on a rolling statistic of the validation errors on the last fold of cross-validation.

Parameters

- **base_estimator** (*sklearn.base.BaseEstimator*) – The model to which normal `.fit`, `.predict` methods will be used. defaults to py:class:*gordo.machine.model.models.KerasAutoEncoder* with `kind='feedforward_hourglass`
- **scaler** (*sklearn.base.TransformerMixin*) – Defaults to `sklearn.preprocessing.RobustScaler` Used for transforming model output and the original `y` to calculate the difference/error in model output vs expected.
- **require_thresholds** (*bool*) – Requires calculating `thresholds_` via a call to *cross_validate()*. If this is set (default True), but *cross_validate()* was not called before calling *anomaly()* an `AttributeError` will be raised.
- **shuffle** (*bool*) – Flag to shuffle or not data in `.fit` so that the model, if relevant, will be trained on a sample of data accross the time range and not just the last elements according to model arg `validation_split`.
- **window** (*int*) – Window size for smoothed thresholds
- **smoothing_method** (*str*) – Method to be used together with `window` to smooth metrics. Must be one of: 'smm': simple moving median, 'sma': simple moving average or 'ewma': exponential weighted moving average.

anomaly(*X: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], y: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray], frequency: Optional[datetime.timedelta] = None*) → Union[pandas.core.frame.DataFrame, xarray.core.dataset.Dataset]

Create an anomaly dataframe from the base provided dataframe.

Parameters

- **X** (*pd.DataFrame*) – Dataframe representing the data to go into the model.
- **y** (*pd.DataFrame*) – Dataframe representing the target output of the model.

Returns A superset of the original base dataframe with added anomaly specific features

Return type pd.DataFrame

cross_validate(*, *X: Union[pandas.core.frame.DataFrame, numpy.ndarray]*, *y: Union[pandas.core.frame.DataFrame, numpy.ndarray]*, *cv=TimeSeriesSplit(max_train_size=None, n_splits=3)*, ***kwargs*)

Run TimeSeries cross validation on the model, and will update the model's threshold values based on the cross validation folds.

Parameters

- **X** (*Union[pd.DataFrame, np.ndarray]*) – Input data to the model
- **y** (*Union[pd.DataFrame, np.ndarray]*) – Target data
- **kwargs** (*dict*) – Any additional kwargs to be passed to `sklearn.model_selection.cross_validate()`

Returns

Return type dict

fit(*X: numpy.ndarray*, *y: numpy.ndarray*)

get_metadata()

Generates model metadata.

Returns

Return type dict

get_params(*deep=True*)

Get parameters for this estimator.

Returns

Return type dict

score(*X: Union[numpy.ndarray, pandas.core.frame.DataFrame]*, *y: Union[numpy.ndarray, pandas.core.frame.DataFrame]*, *sample_weight: Optional[numpy.ndarray] = None*) → float

Score the model; must implement the correct default scorer based on model type

class gordo.machine.model.anomaly.diff.**DiffBasedKFCVAnomalyDetector**(*base_estim*
sklearn.bas
=
ten-
sor-
flow.keras.
scaler:
sklearn.bas
=
Min-
MaxS-
caler(),
re-
quire_thres
bool
=
True,
shuf-
fle:
bool
=
True,
win-
dow:
int
=
144,
smooth-
ing_metho
str
=
'smm',
thresh-
old_percen
float
=
0.99)

Bases: *gordo.machine.model.anomaly.diff.DiffBasedAnomalyDetector*

Estimator which wraps a `base_estimator` and provides a diff error based approach to anomaly detection.

It trains a `scaler` to the target **after** training, purely for error calculations. The underlying `base_estimator` is trained with the original, unscaled, `y`.

Threshold calculation is based on a percentile of the smoothed validation errors as calculated from cross-validation predictions.

Parameters

- **base_estimator** (*sklearn.base.BaseEstimator*) – The model to which normal `.fit`, `.predict` methods will be used. defaults to py:class:*gordo.machine.model.models.KerasAutoEncoder* with `kind='feedforward_hourglass`
- **scaler** (*sklearn.base.TransformerMixin*) – Defaults to `sklearn.preprocessing.RobustScaler` Used for transforming model output and the original `y` to calculate the difference/error in model output vs expected.
- **require_thresholds** (*bool*) – Requires calculating `thresholds_` via a call to *cross_validate()*. If this is set (default True), but *cross_validate()* was not called before calling *anomaly()* an `AttributeError` will be raised.
- **shuffle** (*bool*) – Flag to shuffle or not data in `.fit` so that the model, if relevant, will be trained on a sample of data accross the time range and not just the last elements according to model arg `validation_split`.
- **window** (*int*) – Window size for smooth metrics and threshold calculation.
- **smoothing_method** (*str*) – Method to be used together with `window` to smooth metrics. Must be one of: 'smm': simple moving median, 'sma': simple moving average or 'ewma': exponential weighted moving average.
- **threshold_percentile** (*float*) – Percentile of the validation data to be used to calculate the threshold.

cross_validate(**, X: Union[pandas.core.frame.DataFrame, numpy.ndarray], y: Union[pandas.core.frame.DataFrame, numpy.ndarray], cv=KFold(n_splits=5, random_state=0, shuffle=True), **kwargs*)

Run Kfold cross validation on the model, and will update the model's threshold values based on a percentile of the validation metrics.

Parameters

- **X** (*Union[pd.DataFrame, np.ndarray]*) – Input data to the model
- **y** (*Union[pd.DataFrame, np.ndarray]*) – Target data

- **kwargs** (*dict*) – Any additional kwargs to be passed to `sklearn.model_selection.cross_validate()`

Returns

Return type dict

get_metadata()
Generates model metadata.

Returns

Return type dict

get_params(*deep=True*)
Get parameters for this estimator.

Returns

Return type dict

1.4.2.3 Utils

Shared utility functions used by models and other components interacting with the model's.

gordo.machine.model.utils.**make_base_dataframe**(*tags: Union[List[gordo_dataset.sensor_tag.Senso List[str]], model_input: numpy.ndarray, model_output: numpy.ndarray, target_tag_list: Union[List[gordo_dataset.sensor_tag.Senso List[str], None] = None, index: Optional[numpy.ndarray] = None, frequency: Optional[datetime.timedelta] = None*) → pandas.core.frame.DataFrame
Construct a dataframe which has a MultiIndex column consisting of top level keys 'model-input' and 'model-output'. Takes care of aligning model output if different than model input lengths, as setting column names based on passed tags and target_tag_list.

Parameters

- **tags** (*List[Union[str, SensorTag]]*) – Tags which will be assigned to `model-input` and/or `model-output` if the shapes

match.

- **model_input** (*np.ndarray*) – Original input given to the model
- **model_output** (*np.ndarray*) – Raw model output
- **target_tag_list** (*Optional[Union[List[SensorTag], List[str]]]*) – Tags to be assigned to `model-output` if not assinged but model output matches model input, `tags` will be used.
- **index** (*Optional[np.ndarray]*) – The index which should be assinged to the resulting dataframe, will be clipped to the length of `model_output`, should the model output less than its input.
- **frequency** (*Optional[datetime.timedelta]*) – The spacing of the time between points.

Returns

Return type pd.DataFrame

gordo.machine.model.utils.**metric_wrapper**(*metric*, *scaler: Optional[sklearn.base.TransformerMixin] = None*)

Ensures that a given metric works properly when the model itself returns a y which is shorter than the target y, and allows scaling the data before applying the metrics.

Parameters

- **metric** – Metric which must accept y_true and y_pred of the same length
- **scaler** (*Optional[TransformerMixin]*) – Transformer which will be applied on y and y_pred before the metrics is calculated. Must have method *transform*, so for most scalers it must already be fitted on *y*.

1.4.3 Metadata

Each Machine is entitled to have Metadata, which can be set at the Machine.metadata level inside the config, but will result in a standardized output of metadata under `user_defined` and `build_metadata`. Where `user_defined` can go arbitrarily deep, depending on the amount of metadata the user wishes to enter.

`build_metadata` is more predictable. During the course of building a Machine the system will insert certain metadata given about the build time, and model metrics (depending on configuration).

class gordo.machine.metadata.metadata.**Metadata**(*user_defined: Dict[str, Any] = <factory>, build_metadata: gordo.machine.metadata.metadata.BuildM = <factory>*)

Bases: object

build_metadata: BuildMetadata = None

classmethod from_dict(*kvs: Union[dict, list, str, int, float, bool, None], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → dataclasses_json.mm.SchemaF[~A][A]

to_dict(*encode_json=False*) → Dict[str, Union[dict, list, str, int, float, bool, None]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Union[int, str, None] = None*, *separators: Tuple[str, str] = None*, *default: Callable = None*, *sort_keys: bool = False*, ***kw*) → str

user_defined: Dict[str, Any] = None

class gordo.machine.metadata.metadata.**BuildMetadata**(*model: gordo.machine.metadata.metadata. = <factory>, dataset: gordo.machine.metadata.metadata. = <factory>*)

Bases: object

dataset: DatasetBuildMetadata = None

classmethod from_dict(*kvs: Union[dict, list, str, int, float, bool, None], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

model: ModelBuildMetadata = None

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → dataclasses_json.mm.SchemaF[~A][A]

to_dict(*encode_json=False*) → Dict[str, Union[dict, list, str, int, float, bool, None]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Union[int, str, None] = None*, *separators: Tuple[str, str] = None*, *default: Callable = None*, *sort_keys: bool = False*, ***kw*) → str

class gordo.machine.metadata.metadata.**ModelBuildMetadata**(*model_offset: int = 0*, *model_creation_date: Union[str, NoneType] = None*, *model_builder_version: str = '1.10.5'*, *cross_validation: gordo.machine.metadata.m = <factory>*, *model_training_duration_s Union[float, NoneType] = None*, *model_meta: Dict[str, Any] = <factory>*)

Bases: object

cross_validation: CrossValidationMetaData = None

classmethod from_dict(*kvs: Union[dict, list, str, int, float, bool, None], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

model_builder_version: str = '1.10.5'

model_creation_date: Optional[str] = None

model_meta: Dict[str, Any] = None

model_offset: int = 0

model_training_duration_sec: Optional[float] = None

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → dataclasses_json.mm.SchemaF[~A][A]

to_dict(*encode_json=False*) → Dict[str, Union[dict, list, str, int, float, bool, None]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Union[int, str, None] = None*, *separators: Tuple[str, str] = None*, *default: Callable = None*, *sort_keys: bool = False*, **kw*) → str

class gordo.machine.metadata.metadata.**CrossValidationMetaData**(*scores: Dict[str, Any] = <factory>*, *cv_duration_sec: Union[float, NoneType] = None*, *splits: Dict[str, Any] = <factory>*)

Bases: object

cv_duration_sec: Optional[float] = None

classmethod from_dict(*kvs: Union[dict, list, str, int, float, bool, None], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → dataclasses_json.mm.SchemaF[~A][A]

scores: Dict[str, Any] = None

splits: Dict[str, Any] = None

to_dict(*encode_json=False*) → Dict[str, Union[dict, list, str, int, float, bool, None]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Union[int, str, None] = None*, *separators: Tuple[str, str] = None*, *default: Callable = None*, *sort_keys: bool = False*, **kw*) → str

class gordo.machine.metadata.metadata.**DatasetBuildMetadata**(*query_duration_sec: Union[float, NoneType] = None*, *dataset_meta: Dict[str, Any] = <factory>*)

Bases: object

dataset_meta: Dict[str, Any] = None

classmethod from_dict(*kvs: Union[dict, list, str, int, float, bool, None], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

query_duration_sec: Optional[float] = None

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → dataclasses_json.mm.SchemaF[~A][A]

to_dict(*encode_json=False*) → Dict[str, Union[dict, list, str, int, float, bool, None]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Union[int, str, None] = None*, *separators: Tuple[str, str] = None*, *default: Callable = None*, *sort_keys: bool = False*, **kw*) → str

1.5 Builder

1.5.1 Model builder

class gordo.builder.build_model.**ModelBuilder**(*machine: gordo.machine.machine.Machine*)

Bases: object

Build a model for a given gordo.workflow.config_elements.machine.Machine

Parameters **machine** (Machine) –

Example

```
>>> from gordo_dataset.sensor_tag import SensorTag
>>> from gordo.machine import Machine
>>> from gordo.dependencies import configure_once
>>> configure_once()
>>> machine = Machine(
...     name="special-model-name",
...     model={"sklearn.decomposition.PCA": {"svd_solver": "auto"}}
 ↪,
...     dataset={
...         "type": "RandomDataset",
...         "train_start_date": "2017-12-25 06:00:00Z",
...         "train_end_date": "2017-12-30 06:00:00Z",
...         "tag_list": [SensorTag("Tag 1", None), SensorTag("Tag 2
 ↪", None)],
...         "target_tag_list": [SensorTag("Tag 3", None),␣
 ↪SensorTag("Tag 4", None)]
...     },
...     project_name='test-proj',
... )
>>> builder = ModelBuilder(machine=machine)
>>> model, machine = builder.build()
```

build(*output_dir: Union[os.PathLike, str, None] = None*, *model_register_dir: Union[os.PathLike, str, None] = None*, *replace_cache=False*) → Tuple[sklearn.base.BaseEstimator, gordo.machine.machine.Machine]

Always return a model and its metadata.

If `output_dir` is supplied, it will save the model there. `model_register_dir` points to the model cache directory which it will attempt to read the model from. Supplying both will then have the effect of both; reading from the cache and saving that cached model to the new output directory.

Parameters

- **output_dir** (*Optional[Union[os.PathLike, str]]*) – A path to where the model will be deposited.
- **model_register_dir** (*Optional[Union[os.PathLike, str]]*) – A path to a register, see *:func:gordo.util.disk_registry*. If this is None then always build the model, otherwise try to resolve the model from the registry.
- **replace_cache** (*bool*) – Forces a rebuild of the model, and replaces the entry in the cache with the new model.

Returns Built model and an updated `Machine`

Return type Tuple[sklearn.base.BaseEstimator, *Machine*]

static build_metrics_dict(*metrics_list: list*, *y: pandas.core.frame.DataFrame*, *scaler: Union[sklearn.base.TransformerMixin, str, None] = None*) → dict

Given a list of metrics that accept a true_y and pred_y as inputs this returns a dictionary with keys in the form '{score}-{tag_name}' for each given target tag and '{score}' for the average score across all target tags and folds, and values being the callable make_scorer(metric_wrapper(score)). Note: score in {score}-{tag_name} is a sklearn's score function name with '_' replaced by '-' and tag_name corresponds to given target tag name with ' ' replaced by '-'.

Parameters

- **metrics_list** (*list*) – List of sklearn score functions
- **y** (*pd.DataFrame*) – Target data
- **scaler** (*Optional[Union[TransformerMixin, str]]*) – Scaler which will be fitted on y, and used to transform the data before scoring. Useful when the metrics are sensitive to the amplitude of the data, and you have multiple targets.

Returns

Return type dict

static build_split_dict(*X: pandas.core.frame.DataFrame*, *split_obj: Type[sklearn.model_selection._split.BaseCrossValidator]*) → dict

Get dictionary of cross-validation training dataset split metadata

Parameters

- **X** (*pd.DataFrame*) – The training dataset that will be split during cross-validation.
- **split_obj** (*Type[sklearn.model_selection.BaseCrossValidator]*) – The cross-validation object that returns train, test indices for splitting.

Returns **split_metadata** – Dictionary of cross-validation train/test split metadata

Return type Dict[str,Any]

property cache_key

property cached_model_path

static calculate_cache_key(*machine: gordo.machine.machine.Machine*) → str

Calculates a hash-key from the model and data-config.

Returns A 512 byte hex value as a string based on the content of the parameters.

Return type str

Examples

```
>>> from gordo.machine import Machine
>>> from gordo_dataset.sensor_tag import SensorTag
>>> from gordo.dependencies import configure_once
>>> configure_once()
>>> machine = Machine(
...     name="special-model-name",
...     model={"sklearn.decomposition.PCA": {"svd_solver":
↪"auto"}},
...     dataset={
...         "type": "RandomDataset",
...         "train_start_date": "2017-12-25 06:00:00Z",
...         "train_end_date": "2017-12-30 06:00:00Z",
...         "tag_list": [SensorTag("Tag 1", None), SensorTag(
↪"Tag 2", None)],
...         "target_tag_list": [SensorTag("Tag 3", None),␣
↪SensorTag("Tag 4", None)]
...     },
...     project_name='test-proj'
```

(continues on next page)

(continued from previous page)

```
... )
>>> builder = ModelBuilder(machine)
>>> len(builder.cache_key)
128
```

check_cache(*model_register_dir: Union[os.PathLike, str]*)

Checks if the model is cached, and returns its path if it exists.

Parameters

- **model_register_dir** (*[os.PathLike, None]*) – The register dir where the model lies.
- **cache_key** (*str*) – A 512 byte hex value as a string based on the content of the parameters.

Returns

- **-------** –
- **None]** (*Union[os.PathLike,]*) – The path to the cached model, or None if it does not exist.

static metrics_from_list(*metric_list: Optional[List[str]] = None*) → List[Callable]

Given a list of metric function paths. ie. sklearn.metrics.r2_score or simple function names which are expected to be in the `sklearn.metrics` module, this will return a list of those loaded functions.

Parameters **metrics** (*Optional[List[str]]*) – List of function paths to use as metrics for the model Defaults to those specified in `gordo.workflow.config_components.NormalizedConfig` sklearn.metrics.explained_variance_score, sklearn.metrics.r2_score, sklearn.metrics.mean_squared_error, sklearn.metrics.mean_absolute_error

Returns A list of the functions loaded

Return type List[Callable]

Raises **AttributeError:** – If the function cannot be loaded.

set_seed(*seed: int*)

1.5.2 Local Model builder

This is meant to provide a good way to validate a configuration file as well as to enable creating and testing models locally with little overhead.

gordo.builder.local_build.**local_build**(*config_str: str*) → Iterable[Tuple[Optional[sklearn.base.BaseEstimator], gordo.machine.machine.Machine]]

Build model(s) from a bare Gordo config file locally.

This is very similar to the same steps as the normal workflow generation and subsequent Gordo deployment process makes. Should help developing locally, as well as giving a good indication that your config is valid for deployment with Gordo.

Parameters **config_str** (*str*) – The raw yaml config file in string format.

Examples

```
>>> import numpy as np
>>> from gordo.dependencies import configure_once
>>> configure_once()
>>> config = '''
... machines:
...       - dataset:
...           tags:
...             - SOME-TAG1
...             - SOME-TAG2
...           target_tag_list:
...             - SOME-TAG3
...             - SOME-TAG4
...           train_end_date: '2019-03-01T00:00:00+00:00'
...           train_start_date: '2019-01-01T00:00:00+00:00'
...           asset: asgb
...           data_provider:
...             type: RandomDataProvider
...         metadata:
...           information: Some sweet information about the model
...         model:
...           gordo.machine.model.anomaly.diff.
↪DiffBasedAnomalyDetector:
...             base_estimator:
...               sklearn.pipeline.Pipeline:
...                 steps:
...                 - sklearn.decomposition.PCA
...                 - sklearn.multioutput.MultiOutputRegressor:
...                     estimator: sklearn.linear_model.
↪LinearRegression
```

(continues on next page)

(continued from previous page)

```
...         name: crazy-sweet-name
... '''
>>> models_n_metadata = local_build(config)
>>> assert len(list(models_n_metadata)) == 1
```

Returns A generator yielding tuples of models and their metadata.

Return type Iterable[Tuple[Union[BaseEstimator, None], *Machine*]]

1.6 Serializer

The serializer is the core component used in the conversion of a Gordo config file into Python objects which interact in order to construct a full ML model capable of being served on Kubernetes.

Things like the `dataset` and `model` keys within the YAML config represents objects which will be (de)serialized by the serializer to complete this goal.

`gordo.serializer.serializer.`**`dump`**(*obj: object, dest_dir: Union[os.PathLike, str], metadata: dict = None*)

Serialize an object into a directory, the object must be pickle-able.

Parameters

- **`obj`** – The object to dump. Must be pickle-able.
- **`dest_dir`** (*`Union[os.PathLike, str]`*) – The directory to which to save the model metadata: dict - any additional metadata to be saved alongside this model if it exists, will be returned from the corresponding "load" function
- **`metadata`** (*`Optional dict of metadata which will be serialized to a file together`*) – with the model, and loaded again by *`load_metadata()`*.

Returns

Return type None

Example

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.decomposition import PCA
>>> from gordo.machine.model.models import KerasAutoEncoder
>>> from gordo import serializer
>>> from tempfile import TemporaryDirectory
>>> pipe = Pipeline([
...     ('pca', PCA(3)),
...     ('model', KerasAutoEncoder(kind='feedforward_hourglass'))])
>>> with TemporaryDirectory() as tmp:
...     serializer.dump(obj=pipe, dest_dir=tmp)
...     pipe_clone = serializer.load(source_dir=tmp)
```

gordo.serializer.serializer.**dumps**(*model: Union[sklearn.pipeline.Pipeline, gordo.machine.model.base.GordoBase]*) → bytes

Dump a model into a bytes representation suitable for loading from `gordo.serializer.loads`

Parameters **model** (*Union[Pipeline, GordoBase]*) – A gordo model/pipeline

Returns Serialized model which supports loading via `serializer.loads()`

Return type bytes

Example

```
>>> from gordo.machine.model.models import KerasAutoEncoder
>>> from gordo import serializer
>>>
>>> model = KerasAutoEncoder('feedforward_symmetric')
>>> serialized = serializer.dumps(model)
>>> assert isinstance(serialized, bytes)
>>>
>>> model_clone = serializer.loads(serialized)
>>> assert isinstance(model_clone, KerasAutoEncoder)
```

gordo.serializer.serializer.**load**(*source_dir: Union[os.PathLike, str]*) → Any

Load an object from a directory, saved by `gordo.serializer.pipeline_serializer.dump`

This take a directory, which is either top-level, meaning it contains a sub directory in the naming scheme: "n_step=<int>-class=<path.to.Class>" or the aforementioned naming

scheme directory directly. Will return that unsterilized object.

Parameters **source_dir** (*Union[os.PathLike, str]*) – Location of the top level dir the pipeline was saved

Returns

Return type Union[*GordoBase*, Pipeline, BaseEstimator]

gordo.serializer.serializer.**load_metadata**(*source_dir: Union[os.PathLike, str]*) → dict

Load the given metadata.json which was saved during the `serializer.dump` will return the loaded metadata as a dict, or empty dict if no file was found

Parameters **source_dir** (*Union[os.PathLike, str]*) – Directory of the saved model, As with serializer.load(source_dir) this source_dir can be the top level, or the first dir into the serialized model.

Returns

Return type dict

Raises **FileNotFoundError** – If a 'metadata.json' file isn't found in or above the supplied `source_dir`

gordo.serializer.serializer.**loads**(*bytes_object: bytes*) → gordo.machine.model.base.GordoBase

Load a GordoBase model from bytes dumped from `gordo.serializer.dumps`

Parameters **bytes_object** (*bytes*) – Bytes to be loaded, should be the result of *serializer.dumps(model)*

Returns Custom gordo model, scikit learn pipeline or other scikit learn like object.

Return type Union[*GordoBase*, Pipeline, BaseEstimator]

1.6.1 From Definition

The ability to take a 'raw' representation of an object in `dict` form and load it into a Python object.

gordo.serializer.from_definition.**from_definition**(*pipe_definition: Union[str, Dict[str, Dict[str, Any]]]*) → Union[sklearn.pipeline.FeatureUnion, sklearn.pipeline.Pipeline]

Construct a Pipeline or FeatureUnion from a definition.

Example

```
>>> import yaml
>>> from gordo import serializer
>>> raw_config = '''
... sklearn.pipeline.Pipeline:
...         steps:
...             - sklearn.decomposition.PCA:
...                 n_components: 3
...             - sklearn.pipeline.FeatureUnion:
...                 - sklearn.decomposition.PCA:
...                     n_components: 3
...                 - sklearn.pipeline.Pipeline:
...                     - sklearn.preprocessing.MinMaxScaler
...                     - sklearn.decomposition.TruncatedSVD:
...                         n_components: 2
...             - sklearn.ensemble.RandomForestClassifier:
...                 max_depth: 3
... '''
>>> config = yaml.safe_load(raw_config)
>>> scikit_learn_pipeline = serializer.from_definition(config)
```

Parameters

- **pipe_definition** – List of steps for the Pipeline / FeatureUnion
- **constructor_class** – What to place the list of transformers into, either sklearn.pipeline.Pipeline/FeatureUnion

Returns pipeline

Return type sklearn.pipeline.Pipeline

gordo.serializer.from_definition.**import_locate**(*import_path: str*) → Any

gordo.serializer.from_definition.**load_params_from_definition**(*definition: dict*) → dict

Deserialize each value from a dictionary. Could be used for preparing kwargs for methods

Parameters **definition** (*dict*) –

1.6.2 Into Definitiion

The ability to take a Python object, such as a scikit-learn pipeline and convert it into a primitive `dict`, which can then be inserted into a YAML config file.

`gordo.serializer.into_definition.`**`into_definition`**(*pipeline: sklearn.pipeline.Pipeline*, *prune_default_params: bool = False*) → dict

Convert an instance of `sklearn.pipeline.Pipeline` into a dict definition capable of being reconstructed with `gordo.serializer.from_definition`

Parameters

- **pipeline** (*sklearn.pipeline.Pipeline*) – Instance of pipeline to decompose
- **prune_default_params** (*bool*) – Whether to prune the default parameters found in current instance of the transformers vs what their default params are.

Returns definitions for the pipeline, compatible to be reconstructed with *gordo.serializer.from_definition()*

Return type dict

Example

```
>>> import yaml
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.decomposition import PCA
>>> from gordo.machine.model.models import KerasAutoEncoder
>>>
>>> pipe = Pipeline([('pca', PCA(4)), ('ae', KerasAutoEncoder(kind=
↪'feedforward_model'))])
>>> pipe_definition = into_definition(pipe)  # It is now a␣
↪standard python dict of primitives.
>>> print(yaml.dump(pipe_definition))
sklearn.pipeline.Pipeline:
  memory: null
  steps:
  - sklearn.decomposition._pca.PCA:
      copy: true
      iterated_power: auto
      n_components: 4
      random_state: null
```

(continues on next page)

(continued from previous page)

```
      svd_solver: auto
      tol: 0.0
      whiten: false
  - gordo.machine.model.models.KerasAutoEncoder:
      kind: feedforward_model
  verbose: false
```

gordo.serializer.into_definition.**load_definition_from_params**(*params: dict*) → dict

Recursively decomposing each of values from params into the definition

Parameters **params** (*dict*) –

Returns

Return type dict

1.7 ML Server

The ML Server is responsible for giving different "views" into the model being served.

1.7.1 Server

This module contains code for generating the Gordo server Flask application.

Running this module will run the application using Flask's development webserver. Gunicorn can be used to run the application as *gevent* async workers by using the *run_server()* function.

class gordo.server.server.**Config**

Bases: object

Server config

gordo.server.server.**adapt_proxy_deployment**(*wsgi_app: Callable*) → Callable

Decorator specific to fixing behind-proxy-issues when on Kubernetes and using Envoy proxy.

Parameters **wsgi_app** (*typing.Callable*) – The underlying WSGI application of a flask app, for example

Notes

Special note about deploying behind Ambassador, or prefixed proxy paths in general:

When deployed on kubernetes/ambassador there is a prefix in-front of the server. ie:

```
/gordo/v0/some-project-name/some-target
```

The server itself only knows about routes to the right of such a prefix: such as `/metadata` or `/predictions` when in reality, the full path is:

```
/gordo/v0/some-project-name/some-target/metadata
```

This is solved by getting the current application's assigned prefix, where `HTTP_X_ENVOY_ORIGINAL_PATH` is the *full* path, including the prefix. and `PATH_INFO` is the actual relative path the server knows about.

This function wraps the WSGI app itself to map the current full path to the assigned route function.

ie. `/metadata` -> metadata route function, by default, but updates `/gordo/v0/some-project-name/some-target/metadata` -> metadata route function

Returns

Return type Callable

Example

```
>>> app = Flask(__name__)
>>> app.wsgi_app = adapt_proxy_deployment(app.wsgi_app)
```

gordo.server.server.**build_app**(*config: Optional[Dict[str, Any]] = None*, *prometheus_registry: Optional[prometheus_client.registry.CollectorRegistry] = None*)

Build app and any associated routes

gordo.server.server.**create_prometheus_metrics**(*project: Optional[str] = None*, *registry: Optional[prometheus_client.registry.CollectorR = None*) → gordo.server.prometheus.metrics.GordoServ

gordo.server.server.**enable_prometheus**()

gordo.server.server.**run_cmd**(*cmd*)

Run a shell command and handle CalledProcessError and OSError types

Note: This function is abstracted from *run_server()* in order to test the calling of commands that would allow the subprocess call to break, depending on how it is parameterized. For example, calling this without sending *stderr* to stdout will cause a segmentation fault when calling an executable that does not exist.

gordo.server.server.**run_server**(*host: str*, *port: int*, *workers: int*, *log_level: str*, *config_module: Optional[str] = None*, *worker_connections: Optional[int] = None*, *threads: Optional[int] = None*, *worker_class: str = 'gthread'*, *server_app: str = 'gordo.server.server:build_app()'*)

Run application with Gunicorn server using Gevent Async workers

Parameters

- **host** (*str*) – The host to run the server on.
- **port** (*int*) – The port to run the server on.
- **workers** (*int*) – The number of worker processes for handling requests.
- **log_level** (*str*) – The log level for the *gunicorn* webserver. Valid log level names can be found in the [gunicorn documentation](http://docs.gunicorn.org/en/stable/settings.html#loglevel).
- **config_module** (*str*) – The config module. Will be passed with *python:* [prefix](https://docs.gunicorn.org/en/stable/settings.html#config).
- **worker_connections** (*int*) – The maximum number of simultaneous clients per worker process.
- **threads** (*str*) – The number of worker threads for handling requests.
- **worker_class** (*str*) – The type of workers to use.
- **server_app** (*str*) – The application to run

1.7.2 Views

A collection of implemented views into the Model being served.

1.7.2.1 Base

Provides the most basic view into the model. This view will simply apply the model to the provided data and return the `model-output` along with the `model-output`

class gordo.server.views.base.**BaseModelView**(*api=None*, **args*, ***kwargs*)

Bases: `flask_restplus.resource.Resource`

The base model view.

X: pandas.core.frame.DataFrame = None

endpoint = 'base_model_view'

property frequency

The frequency the model was trained with in the dataset

static load_build_dataset_metadata()

mediatypes()

methods = ['POST']

post()

Process a POST request by using provided user data

A typical response might look like this

```
{
    'data': [
        {
            'end': ['2016-01-01T00:10:00+00:00'],
            'model-output': [0.0005317790200933814,
                             -0.0001525811239844188,
                             0.0008310950361192226,
                             0.0015755111817270517],
            'original-input': [0.9135588550070414,
                               0.3472517774179448,
                               0.8994921857179736,
                               0.11982773108991263],
            'start': ['2016-01-01T00:00:00+00:00'],
        },
        ...
    ],

    'tags': [
        {'asset': None, 'name': 'tag-0'},
        {'asset': None, 'name': 'tag-1'},
        {'asset': None, 'name': 'tag-2'},
        {'asset': None, 'name': 'tag-3'}
```

(continues on next page)

(continued from previous page)

```
    ],
    'time-seconds': '0.1937'
}
```

property tags
The input tags for this model

Returns

Return type typing.List[SensorTag]

property target_tags
The target tags for this model

Returns

Return type typing.List[SensorTag]

y: pandas.core.frame.DataFrame = None

class gordo.server.views.base.**DownloadModel**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

Download the trained model

suitable for reloading via *gordo.serializer.serializer.loads()*

endpoint = 'download_model'

get()
Responds with a serialized copy of the current model being served.

Returns Results from gordo.serializer.dumps()

Return type bytes

mediatypes()

methods = {'GET'}

class gordo.server.views.base.**ExpectedModels**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

endpoint = 'expected_models'

get(*gordo_project: str*)

mediatypes()

methods = {'GET'}

class gordo.server.views.base.**MetaDataView**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

Serve model / server metadata

endpoint = 'meta_data_view'

get()

Get metadata about this endpoint, also serves as /healthcheck endpoint

mediatypes()

methods = {'GET'}

class gordo.server.views.base.**ModelListView**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

List the current models capable of being served by the server

endpoint = 'model_list_view'

get(*gordo_project: str*)

mediatypes()

methods = {'GET'}

class gordo.server.views.base.**RevisionListView**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

List the available revisions the model can serve.

endpoint = 'revision_list_view'

get(*gordo_project: str*)

mediatypes()

methods = {'GET'}

1.7.2.2 Anomaly

The anomaly view into the model. Expects that the model being served when accessing this route implements the `anomaly()` method in order to calculate the anomaly key(s) for the response.

class gordo.server.views.anomaly.**AnomalyView**(*api=None*, **args*, ***kwargs*)

Bases: *gordo.server.views.base.BaseModelView*

Serve model predictions via POST method.

Gives back predictions looking something like this (depending on anomaly model being served):

```
   {
   'data': [
       {
      'end': ['2016-01-01T00:10:00+00:00'],
      'tag-anomaly-scaled': [0.913027075986948,
                             0.3474043585419292,
                             0.8986610906818544,
                             0.118252219908185 57],
      'tag-anomaly-unscaled': [10.2335327305725986948,
                             4.234343958392+3293,
                             10.379394390232232,
                             3.32093438982743929],
      'model-output': [0.0005317790200933814,
                       -0.0001525811239844188,
                       0.0008310950361192226,
                       0.0015755111817270517],
      'original-input': [0.9135588550070414,
                         0.3472517774179448,
                         0.8994921857179736,
                         0.11982773108991263],
      'start': ['2016-01-01T00:00:00+00:00'],
      'total-anomaly-unscaled': [1.3326228173185086],
      'total-anomaly-scaled': [0.30203283280023 92],
       },
       ...
   ],

'tags': [{'asset': None, 'name': 'tag-0'},
         {'asset': None, 'name': 'tag-1'},
         {'asset': None, 'name': 'tag-2'},
         {'asset': None, 'name': 'tag-3'}],
'time-seconds': '0.1937'}
```

endpoint = 'anomaly_view'

mediatypes()

methods = ['POST']

post()

Process a POST request by using provided user data

A typical response might look like this

```
{
```

(continues on next page)

(continued from previous page)

```
    'data': [
        {
            'end': ['2016-01-01T00:10:00+00:00'],
            'model-output': [0.0005317790200933814,
                             -0.0001525811239844188,
                             0.0008310950361192226,
                             0.0015755111817270517],
            'original-input': [0.9135588550070414,
                               0.3472517774179448,
                               0.8994921857179736,
                               0.11982773108991263],
            'start': ['2016-01-01T00:00:00+00:00'],
        },
        ...
    ],

    'tags': [
        {'asset': None, 'name': 'tag-0'},
        {'asset': None, 'name': 'tag-1'},
        {'asset': None, 'name': 'tag-2'},
        {'asset': None, 'name': 'tag-3'}
    ],
    'time-seconds': '0.1937'
}
```

1.7.3 Utils

Shared utility functions and decorators which are used by the Views

gordo.server.utils.**dataframe_from_dict**(*data: dict*) → pandas.core.frame.DataFrame

The inverse procedure done by `multi_lvl_column_dataframe_from_dict()` Reconstructed a MultiIndex column dataframe from a previously serialized one.

Expects `data` to be a nested dictionary where each top level key has a value capable of being loaded from `pandas.core.DataFrame.from_dict()`

Parameters **data** (*dict*) – Data to be loaded into a MultiIndex column dataframe

Returns MultiIndex column dataframe.

Return type pandas.core.DataFrame

Examples

```
>>> serialized = {
... 'feature0': {'sub-feature-0': {'2019-01-01': 0, '2019-02-01':␣
↪4},
...              'sub-feature-1': {'2019-01-01': 1, '2019-02-01':␣
↪5}},
... 'feature1': {'sub-feature-0': {'2019-01-01': 2, '2019-02-01':␣
↪6},
...              'sub-feature-1': {'2019-01-01': 3, '2019-02-01':␣
↪7}}
... }
>>> dataframe_from_dict(serialized)
               feature0                    feature1
      sub-feature-0 sub-feature-1 sub-feature-0 sub-feature-1
2019-01-01             0             1             2             3
2019-02-01             4             5             6             7
```

gordo.server.utils.**dataframe_from_parquet_bytes**(*buf: bytes*) → pandas.core.frame.DataFrame

Convert bytes representing a parquet table into a pandas dataframe.

Parameters **buf** (*bytes*) – Bytes representing a parquet table. Can be the direct result from *func*::gordo.server.utils.dataframe_into_parquet_bytes

Returns

Return type pandas.DataFrame

gordo.server.utils.**dataframe_into_parquet_bytes**(*df: pandas.core.frame.DataFrame*, *compression: str = 'snappy'*) → bytes

Convert a dataframe into bytes representing a parquet table.

Parameters

- **df** (*pd.DataFrame*) – DataFrame to be compressed
- **compression** (*str*) – Compression to use, passed to `pyarrow.parquet.write_table()`

Returns

Return type bytes

gordo.server.utils.**dataframe_to_dict**(*df: pandas.core.frame.DataFrame*) → dict

Convert a dataframe can have a `pandas.MultiIndex` as columns into a dict where each key is the top level column name, and the value is the array of columns under the top level name. If it's a simple dataframe, `pandas.core.DataFrame.to_dict()` will be used.

This allows `json.dumps()` to be performed, where `pandas.DataFrame.to_dict()` would convert such a multi-level column dataframe into keys of `tuple` objects, which are not json serializable. However this ends up working with `pandas.DataFrame.from_dict()`

Parameters **df** (*pandas.DataFrame*) – Dataframe expected to have columns of type `pandas.MultiIndex` 2 levels deep.

Returns List of records representing the dataframe in a 'flattened' form.

Return type List[dict]

Examples

```
>>> import pprint
>>> import pandas as pd
>>> import numpy as np
>>> columns = pd.MultiIndex.from_tuples((f"feature{i}", f"sub-
↪feature-{ii}") for i in range(2) for ii in range(2))
>>> index = pd.date_range('2019-01-01', '2019-02-01', periods=2)
>>> df = pd.DataFrame(np.arange(8).reshape((2, 4)),␣
↪columns=columns, index=index)
>>> df
                feature0                    feature1
            sub-feature-0 sub-feature-1 sub-feature-0 sub-feature-1
2019-01-01             0             1             2             3
2019-02-01             4             5             6             7
>>> serialized = dataframe_to_dict(df)
>>> pprint.pprint(serialized)
{'feature0': {'sub-feature-0': {'2019-01-01': 0, '2019-02-01': 4},
              'sub-feature-1': {'2019-01-01': 1, '2019-02-01': 5}},
 'feature1': {'sub-feature-0': {'2019-01-01': 2, '2019-02-01': 6},
              'sub-feature-1': {'2019-01-01': 3, '2019-02-01': 7}}}
```

`gordo.server.utils.`**`extract_X_y`**(*method*)

For a given flask view, will attempt to extract an 'X' and 'y' from the request and assign it to flask's 'g' global request context

If it fails to extract 'X' and (optionally) 'y' from the request, it will **not** run the function but return a `BadRequest` response notifying the client of the failure.

Parameters **method** (*Callable*) – The flask route to decorate, and will return it's own response object and will want to use `flask.g.X` and/or `flask.g.y`

Returns Will either run a `flask.Response` with status code 400 if it fails to extract the X and optionally the y. Otherwise will run the decorated `method` which is also expected to return some sort of `flask.Response` object.

Return type flask.Response

gordo.server.utils.**find_path_in_dict**(*path: List[str], data: dict*) → Any
Find a path in *dict* recursively

Examples

```
>>> find_path_in_dict(["parent", "child"], {"parent": {"child": 42}
↪})
42
```

Parameters

- **path** (*List[str]*) –
- **data** (*dict*) –

gordo.server.utils.**load_metadata**(*directory: str, name: str*) → dict
Load metadata from a directory for a given model by name.

Parameters

- **directory** (*str*) – Directory to look for the model's metadata
- **name** (*str*) – Name of the model to load metadata for, this would be the sub directory within the directory parameter.

Returns

Return type dict

gordo.server.utils.**load_model**
Load a given model from the directory by name.

Parameters

- **directory** (*str*) – Directory to look for the model
- **name** (*str*) – Name of the model to load, this would be the sub directory within the directory parameter.

Returns

Return type BaseEstimator

gordo.server.utils.**metadata_required**(*f*)
Decorate a view which has `gordo_name` as a url parameter and will set `g.metadata` to that model's metadata

gordo.server.utils.**model_required**(*f*)
Decorate a view which has gordo_name as a url parameter and will set g.model to be the loaded model and g.metadata to that model's metadata

gordo.server.utils.**parse_iso_datetime**(*datetime_str: str*) → datetime.datetime

1.7.4 Model IO

The general model input/output operations applied by the views

gordo.server.model_io.**get_model_output**(*model: sklearn.pipeline.Pipeline*, *X: numpy.ndarray*) → numpy.ndarray
Get the raw output from the current model given X. Will try to *predict* and then *transform*, raising an error if both fail.

Parameters **X** (*np.ndarray*) – 2d array of sample(s)

Returns The raw output of the model in numpy array form.

Return type np.ndarray

1.8 CLI

1.8.1 gordo CLI

Available CLIs for Gordo:

1.8.1.1 gordo

The main entry point for the CLI interface

```
gordo [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--log-level <log_level>
Run workflow with custom log-level.

Environment variables

GORDO_LOG_LEVEL

Provide a default for *--log-level*

build

Build a model and deposit it into 'output_dir' given the appropriate config settings.

Parameters
———-
machine_config: dict
A dict loadable by `gordo.machine.Machine.from_config`
output_dir: str
Directory to save model & metadata to.
model_register_dir: path
Path to a directory which will index existing models and their locations, used for re-using old models instead of rebuilding them. If omitted then always rebuild
print_cv_scores: bool
Print cross validation scores to stdout
model_parameter: List[Tuple[str, Any]
List of model key-values, wheres the values will be injected into the model config wherever there is a jinja variable with the key.
exceptions_reporter_file: str
JSON output file for exception information
exceptions_report_level: str
Details level for exception reporting

```
gordo build [OPTIONS] MACHINE_CONFIG [OUTPUT_DIR]
```

Options

--model-register-dir <model_register_dir>

--print-cv-scores
Prints CV scores to stdout

--model-parameter <model_parameter>
Key-Value pair for a model parameter and its value, may use this option multiple times. Separate key,valye by a comma. ie: –model-parameter key,val –model-parameter some_key,some_value

--exceptions-reporter-file <exceptions_reporter_file>
JSON output file for exception information

--exceptions-report-level <exceptions_report_level>
Details level for exception reporting

Options EXIT_CODE | TYPE | MESSAGE | TRACEBACK

Arguments

MACHINE_CONFIG
Required argument

OUTPUT_DIR
Optional argument

Environment variables

MACHINE

Provide a default for *MACHINE_CONFIG*

OUTPUT_DIR

Provide a default for *OUTPUT_DIR*

MODEL_REGISTER_DIR

Provide a default for *--model-register-dir*

EXCEPTIONS_REPORTER_FILE

Provide a default for *--exceptions-reporter-file*

EXCEPTIONS_REPORT_LEVEL

Provide a default for *--exceptions-report-level*

run-server

Run the gordo server app with Gunicorn

```
gordo run-server [OPTIONS]
```

Options

--host <host>
The host to run the server on.

Default 0.0.0.0

--port <port>
The port to run the server on.

Default 5555

--workers <workers>
The number of worker processes for handling requests.

Default 2

--worker-connections <worker_connections>
The maximum number of simultaneous clients per worker process.

Default 50

--threads <threads>
The number of worker threads for handling requests.This argument only has affects with –worker-class=gthread. Default value is 8 (4 x $(NUM_CORES))

--worker-class <worker_class>
The type of workers to use.

Default gthread

--log-level <log_level>
The log level for the server.

Default debug

Options critical | error | warning | info | debug

--server-app <server_app>
The application to run

Default gordo.server.server:build_app()

--with-prometheus-config
Run with custom config for prometheus

Environment variables

GORDO_SERVER_HOST

Provide a default for *--host*

GORDO_SERVER_PORT

Provide a default for *--port*

GORDO_SERVER_WORKERS

Provide a default for *--workers*

GORDO_SERVER_WORKER_CONNECTIONS

Provide a default for *--worker-connections*

GORDO_SERVER_THREADS

Provide a default for *--threads*

GORDO_SERVER_WORKER_CLASS

Provide a default for *--worker-class*

GORDO_SERVER_LOG_LEVEL

Provide a default for *--log-level*

GORDO_SERVER_APP

Provide a default for *--server-app*

workflow

```
gordo workflow [OPTIONS] COMMAND [ARGS]...
```

generate

Machine Configuration to Argo Workflow

```
gordo workflow generate [OPTIONS]
```

Options

--machine-config <machine_config>
Required Machine configuration file

--workflow-template <workflow_template>
Template to expand

--owner-references <owner_references>
Kubernetes owner references to inject into all created resources. Should be a nonempty yaml/json list of owner-references, each owner-reference a dict containing at least the keys 'uid', 'name', 'kind', and 'apiVersion'

--gordo-version <gordo_version>
Version of gordo to use, if different than this one

--project-name <project_name>
Required Name of the project which own the workflow.

--project-revision <project_revision>
Revision of the project which own the workflow.

--output-file <output_file>
Optional file to render to

--namespace <namespace>
Which namespace to deploy services into

--split-workflows <split_workflows>
Split workflows containg more than this number of models into several workflows, where each workflow contains at most this nr of models. The workflows are outputted sequentially with '—' in between, which allows kubectl to apply them all at once.

--n-servers <n_servers>
Max number of ML Servers to use, defaults to N machines * 10

--docker-repository <docker_repository>
The docker repo to use for pulling component images from

--docker-registry <docker_registry>
The docker registry to use for pulling component images from

--retry-backoff-duration <retry_backoff_duration>
retryStrategy.backoff.duration for workflow steps

--retry-backoff-factor <retry_backoff_factor>
retryStrategy.backoff.factor for workflow steps

--gordo-server-workers <gordo_server_workers>
The number of worker processes for handling Gordo server requests.

--gordo-server-threads <gordo_server_threads>
The number of worker threads for handling requests.

--gordo-server-probe-timeout <gordo_server_probe_timeout>
timeoutSeconds value for livenessProbe and readinessProbe of Gordo server Deployment

--without-prometheus
Do not deploy Prometheus for Gordo servers monitoring

--prometheus-metrics-server-workers <prometheus_metrics_server_workers>
Number of workers for Prometheus metrics servers

--image-pull-policy <image_pull_policy>
Default imagePullPolicy for all gordo's images

--with-keda
Enable support for the KEDA autoscaler

--ml-server-hpa-type <ml_server_hpa_type>
HPA type for the ML server

Options none | k8s_cpu | keda

--custom-model-builder-envs <custom_model_builder_envs>
List of custom environment variables in

--prometheus-server-address <prometheus_server_address>
Prometheus url. Required for "–ml-server-hpa-type=keda"

--keda-prometheus-metric-name <keda_prometheus_metric_name>
metricName value for the KEDA prometheus scaler

--keda-prometheus-query <keda_prometheus_query>
query value for the KEDA prometheus scaler

--keda-prometheus-threshold <keda_prometheus_threshold>
threshold value for the KEDA prometheus scaler

--resources-labels <resources_labels>
Additional labels for resources. Have to be empty string or a dictionary in JSON format

--server-termination-grace-period <server_termination_grace_period>
terminationGracePeriodSeconds for the gordo server

--server-target-cpu-utilization-percentage <server_target_cpu_utilization_pe
targetCPUUtilizationPercentage for gordo-server's HPA

Environment variables

WORKFLOW_GENERATOR_MACHINE_CONFIG

Provide a default for *--machine-config*

WORKFLOW_GENERATOR_OWNER_REFERENCES

Provide a default for *--owner-references*

WORKFLOW_GENERATOR_GORDO_VERSION

Provide a default for *--gordo-version*

WORKFLOW_GENERATOR_PROJECT_NAME

Provide a default for *--project-name*

WORKFLOW_GENERATOR_PROJECT_REVISION

Provide a default for *--project-revision*

WORKFLOW_GENERATOR_OUTPUT_FILE

Provide a default for *--output-file*

WORKFLOW_GENERATOR_NAMESPACE

Provide a default for *--namespace*

WORKFLOW_GENERATOR_SPLIT_WORKFLOWS

Provide a default for *--split-workflows*

WORKFLOW_GENERATOR_N_SERVERS

Provide a default for *--n-servers*

WORKFLOW_GENERATOR_DOCKER_REPOSITORY

Provide a default for *--docker-repository*

WORKFLOW_GENERATOR_DOCKER_REGISTRY

Provide a default for *--docker-registry*

WORKFLOW_GENERATOR_RETRY_BACKOFF_DURATION

Provide a default for *--retry-backoff-duration*

WORKFLOW_GENERATOR_RETRY_BACKOFF_FACTOR

Provide a default for *--retry-backoff-factor*

WORKFLOW_GENERATOR_GORDO_SERVER_WORKERS

Provide a default for *--gordo-server-workers*

WORKFLOW_GENERATOR_GORDO_SERVER_THREADS

Provide a default for *--gordo-server-threads*

WORKFLOW_GENERATOR_GORDO_SERVER_PROBE_TIMEOUT

Provide a default for *--gordo-server-probe-timeout*

WORKFLOW_GENERATOR_WITHOUT_PROMETHEUS

Provide a default for *--without-prometheus*

WORKFLOW_GENERATOR_PROMETHEUS_METRICS_SERVER_WORKERS

Provide a default for *--prometheus-metrics-server-workers*

WORKFLOW_GENERATOR_IMAGE_PULL_POLICY

Provide a default for *--image-pull-policy*

WORKFLOW_GENERATOR_WITH_KEDA

Provide a default for *--with-keda*

WORKFLOW_GENERATOR_ML_SERVER_HPA_TYPE

Provide a default for *--ml-server-hpa-type*

WORKFLOW_GENERATOR_CUSTOM_MODEL_BUILDER_ENVS

Provide a default for *--custom-model-builder-envs*

WORKFLOW_GENERATOR_PROMETHEUS_SERVER_ADDRESS

Provide a default for *--prometheus-server-address*

WORKFLOW_GENERATOR_KEDA_PROMETHEUS_METRIC_NAME

Provide a default for *--keda-prometheus-metric-name*

WORKFLOW_GENERATOR_KEDA_PROMETHEUS_QUERY

Provide a default for *--keda-prometheus-query*

WORKFLOW_GENERATOR_KEDA_PROMETHEUS_THRESHOLD

Provide a default for *--keda-prometheus-threshold*

WORKFLOW_GENERATOR_RESOURCE_LABELS

Provide a default for *--resources-labels*

WORKFLOW_GENERATOR_SERVER_TERMINATION_GRACE_PERIOD

Provide a default for *--server-termination-grace-period*

WORKFLOW_GENERATOR_SERVER_TARGET_CPU_UTILIZATION_PERCENTAGE

Provide a default for *--server-target-cpu-utilization-percentage*

1.9 Workflow

The workflow component is responsible for converting a Gordo config into an Argo workflow which then runs the various components in order to build and serve the ML models.

1.9.1 Normalized Config

class gordo.workflow.config_elements.normalized_config.**NormalizedConfig**(*config: dict*, *project_name: str*, *gordo_version: Optional = None*, *model_builder_env: Optional = None*)

Bases: object

Handles the conversion of a single Machine representation in config format and updates it with any features which are 'left out' inside of globals key or the default config globals held here.

DEFAULT_CONFIG_GLOBALS: Dict[str, Any] = {'evaluation': {'cv_mode': '

SPLITED_DOCKER_IMAGES: Dict[str, Any] = {'runtime': {'builder': {'ima

UNIFIED_DOCKER_IMAGES: Dict[str, Any] = {'runtime': {'builder': {'ima

UNIFYING_GORDO_VERSION: str = '1.2.0'

classmethod get_default_globals(*gordo_version: str*) → dict

classmethod prepare_patched_globals(*patched_globals: dict*) → dict

static prepare_runtime(*runtime: dict*) → dict

1.9.2 Workflow Generator

Workflow loading/processing functionality to help the CLI 'workflow' sub-command.

gordo.workflow.workflow_generator.workflow_generator.**default_image_pull_pol**

gordo.workflow.workflow_generator.workflow_generator.**get_dict_from_yaml**(*confi*
Unio
_io.S
→
dict

Read a config file or file like object of YAML into a dict

gordo.workflow.workflow_generator.workflow_generator.**load_workflow_template**

Loads the Jinja2 Template from a specified path

Parameters **workflow_template** (*str*) – Path to a workflow template

Returns Loaded but non-rendered jinja2 template for the workflow

Return type jinja2.Template

gordo.workflow.workflow_generator.workflow_generator.**yaml_filter**(*data:*
Any)
→
str

1.9.3 Helpers

gordo.workflow.workflow_generator.helpers.**patch_dict**(*original_dict:*
dict,
patch_dictionary:
dict) →
dict

Patches a dict with another. Patching means that any path defines in the patch is either added (if it does not exist), or replaces the existing value (if it exists). Nothing is removed from the original dict, only added/replaced.

Parameters

- **original_dict** (*dict*) – Base dictionary which will get paths added/changed

- **patch_dictionary** (*dict*) – Dictionary which will be overlaid on top of original_dict

Examples

```
>>> patch_dict({"highKey":{"lowkey1":1, "lowkey2":2}}, {"highKey":{
↪"lowkey1":10}})
{'highKey': {'lowkey1': 10, 'lowkey2': 2}}
>>> patch_dict({"highKey":{"lowkey1":1, "lowkey2":2}}, {"highKey":{
↪"lowkey3":3}})
{'highKey': {'lowkey1': 1, 'lowkey2': 2, 'lowkey3': 3}}
>>> patch_dict({"highKey":{"lowkey1":1, "lowkey2":2}}, {"highKey2
↪":4})
{'highKey': {'lowkey1': 1, 'lowkey2': 2}, 'highKey2': 4}
```

Returns A new dictionary which is the result of overlaying *patch_dictionary* on top of *original_dict*

Return type dict

1.10 Util

Project helpers, and associated functionality which have no home.

1.10.1 Disk Registry

gordo.util.disk_registry.**delete_value**(*registry_dir: Union[os.PathLike, str], key: str*) → bool

Deletes the value with key reg_key from the registry, and returns True if it existed.

Parameters

- **registry_dir** (*Union[os.PathLike, str]*) – Path to the registry. Does not need to exist
- **key** (*str*) – Key to look up in the registry.

Returns True if the key existed, false otherwise

Return type bool

gordo.util.disk_registry.**get_value**(*registry_dir: Union[os.PathLike, str], key: str*) → Optional[AnyStr]

Retrieves the value with key reg_key from the registry, None if it does not exists.

Parameters

- **registry_dir** (*Union[os.PathLike, str]*) – Path to the registry. If it does not exist we return None
- **key** (*str*) – Key to look up in the registry.

Returns The value of *key* in the registry, None if no value is registered with that key in the registry.

Return type Optional[AnyStr]

gordo.util.disk_registry.**logger = <Logger gordo.util.disk_registry (WARNING)**

A simple file-based key/value registry. Each key gets a file with filename = key, and the content of the file is the value. No fancy. Why? Simple, and there is no problems with concurrent writes to different keys. Concurrent writes to the same key will break stuff.

gordo.util.disk_registry.**write_key**(*registry_dir: Union[os.PathLike, str], key: str, val: AnyStr*)

Registers a key-value combination into the register. Key must valid as a filename.

Parameters

- **registry_dir** (*Union[os.PathLike, str]*) – Path to the registry. If it does not exists it will be created, including any missing folders in the path.
- **key** (*str*) – Key to use for the key/value. Must be valid as a filename.
- **val** (*AnyStr*) – Value to write to the registry.

Examples

In the following example we use the temp directory as the registry >>> import tempfile >>> with tempfile.TemporaryDirectory() as tmpdir: … write_key(tmpdir, “akey”, “aval”) … get_value(tmpdir, “akey”) ‘aval’

1.10.2 Utils

gordo.util.utils.**capture_args**(*method: Callable*)

Decorator that captures args and kwargs passed to a given method. This assumes the decorated method has a self, which has a dict of kwargs assigned as an attribute named _params.

Parameters **method** (*Callable*) – Some method of an object, with ‘self’ as the first parameter.

Returns Returns whatever the original method would return

Return type Any

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P