dymaxionlabs Documentation

Release unknown

Dymaxion Labs

Aug 13, 2020

Contents

1 Features **3**

2 Install **5**

3 Authentication **7**

4 Tutorial **9**

5 Contents **11**
- 5.1 License . . . 11
- 5.2 Contributors . . . 14
- 5.3 Changelog . . . 14
- 5.4 dymaxionlabs . . . 14

6 Indices and tables **23**

Python Module Index **25**

Index **27**

This is the Python package for accessing the Dymaxion Labs Platform.

CHAPTER 1

Features

The Dymaxion Labs Platform allows you:

- Download and upload satellite and drone images.
- Train machine learning models for object detection, segmentation, change detection, and more.
- Work your data using a REST API and Python.

This is a publicly installable package. However, if you want access to our full Platform, you will need to create a Dymaxion Labs account.

CHAPTER 2

Install

The package is currently in beta, so we recommend to install it via pip from the GitHub repository:

```
pip install git+https://github.com/dymaxionlabs/python-sdk.git
```

To install the latest stable version:

```
pip install dymaxionlabs
```

CHAPTER 3

Authentication

Sign up at https://app.dymaxionlabs.com/signup if you do not have a user yet, otherwise log in.

Enter the API Key section, create a new API key and copy the generated key.

You need to set the API key using an environment variable, like this:

```
export DYM_API_KEY=...
```

You can also do this from Python:

```
import os

os.environ["DYM_API_KEY"] = "insert-api-key"
```

From now on, you have full access to the Dymaxion Labs API from Python.

CHAPTER 4

Tutorial

Suppose you want to detect pools in a residential area. First, you need to create an Estimator. In this case, you want an *"object_detection"* type of model, and there is only one class of object.

```
from dymaxionlabs.models import Estimator

pools_detector = Estimator.create(name="Pools detector",
                                  type="object_detection",
                                  classes=["pool"])
```

Now, you should upload the images you want to use for training, add them to your estimator, and create the tiles from the image.

```
from dymaxionlabs.files import File

img = File.upload("pools-2020-02-01.tif", "pools/images/")
pools_detector.add_image(img)

tiling_task = img.tiling(output_path="pools/tiles/")
tiling_task.is_running()
#=> True
```

The tiling process generates tiles of 500x500 by default, but you can adjust the tile size with the *tile_size* parameter.

```
# Tile image in 250x250
tiling_task = img.tiling(output_path="pools/tiles-250/", tile_size=250)
```

Next step is to upload your labels file (GeoJSON file) and add them to your estimator. The labels file must be a GeoJSON of polygons for a specific class. If you have more than one class, you have to separate your labels in different files for each class.

```
labels = File.upload("labels.geojson", 'pools/labels/')
pools_detector.add_labels_for(labels, img, "pool")
```

Now you are ready to train the model. Training might take a few hours to finish, so the *train()* method returns a *Task* instance, that represents the current training task.

```
train_task = pools_detector.train()
train_task.is_running()
#=> True
```

You can adjust a few parameters when training by updating the *configuration* dictionary:

```
# Adjust configuration
pools_detector.configuration.update(epochs=25, steps=500)
# Re-train
train_task = pools_detector.train()
```

Currently there are two training parameters:

- `epochs`: Number of training epochs (default=15)
- `steps`: Number of steps per epoch (default=1000)

When the task finishes, your model will be ready to be used for prediction.

Unless you want to predict over the same image you used for training, you should upload another image and again, create the tiles for that image.

```
predict_img = File.upload("pools.tif", 'pools/predict-images/')
predict_tiles_folder = 'pools/predict-tiles/'
tiling_task = predict_img.tiling(output_path=predict_tiles_folder)
tiling_task.is_running()
#=> True
```

You are now ready to start a prediction task. This process might take a few minutes.

```
prediction_task = pools_detector.predict_files([predict_tiles_folder])
prediction_task.is_running()
#=> True
```

You can download the results when the prediction task has completed. The prediction task will generate the results as _output **artifacts_**, which you can download or export to your storage:

```
prediction_task.list_artifacts()
#=> ["pools.tif/results.csv", "pools.tif/results.geojson"]
prediction_task.download_artifacts("results/")
```

CHAPTER 5

Contents

5.1 License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying

the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

5.2 Contributors

- Gessica Paniagua <gessica@dymaxionlabs.com>
- Alan Toris <alan@dymaxionlabs.com>
- Damián Silvani <damian@dymaxionlabs.com>

5.3 Changelog

5.3.1 Version 0.1

- Upload and download files
- Predict using a trained estimator

5.4 dymaxionlabs

5.4.1 dymaxionlabs package

Submodules

dymaxionlabs.files module

class dymaxionlabs.files.**File**(*name*, *path*, *metadata*, ***extra_attributes*)
Bases: object

The File class represents files stored in Dymaxion Labs.

Files are owned by the authenticated user.

Parameters

- **name** (*str*) – file name
- **path** (*str*) – file path in storage
- **metadata** (*dict*) – file metadata
- **extra_attributes** (*dict*) – extra attributes from API endpoint

classmethod all(*path=''*)
Fetches all files found in path.

Glob patterns are allowed, to search recursively in directories:

```
File.all("foo/b*/images/*.tif")
#=> [<dymaxionlabs.file.File name="foo/bar/images/01.tif">, ...]
```

Parameters **path** (*str*) – path glob pattern (default: "")

Returns a list of *File* of files found in path

Return type list

base_path = '/storage'

delete()
Deletes the file in storage.

Returns `True` if file was succesfully deleted

Return type bool

download(*output_dir='.'*)
Downloads the file and stores it on `output_dir`.

If `output_dir` does not exist, it will be created.

Parameters **output_dir** (*str*) – directory path where file will be stored

classmethod get(*path*, *raise_error=True*)
Gets a specific file in `path`.

Parameters

- **path** (*str*) – file path
- **raise_error** (*bool*) – If False, do not raise NotFoundError if file not found

Return type *File*

tiling(*output_path*, *tile_size=500*)
Starts an image tiling job on the current file.

By default, tiles are 500x500, but you can set the `tile_size` to a different size.

`output_path` should be a directory path in storage where tiles will be generated.

Parameters

- **output_path** (*str*) – tiles output directory
- **tile_size** (*int*) – tile size (default: 500)

Returns a Task with info about the new tiling job

Raises RuntimeError if output path is None

Return type *Task*

classmethod upload(*input_path*, *storage_path=''*, *chunk_size=None*)
Uploads a file to storage

Parameters

- **input_path** (*str*) – path of local file to upload
- **storage_path** (*str*) – destination path in storage
- **chunk_size** (*int*) – size (in MB) of chunks for resumable uploading

Raises FileNotFoundError

Returns uploaded file

Return type *File*

dymaxionlabs.models module

class dymaxionlabs.models.**Estimator**(*, *uuid*, *name*, *classes*, *estimator_type*, *metadata*, *image_files*, *configuration*, *training_tasks=[]*, *prediction_tasks=[]*, ***extra_attributes*)

Bases: object

The Estimator class represents a Model that can be trained to solve different kinds of tasks, like object detection or classification.

To instance an Estimator, you should use one of the following class methods: *all()*, *get()*, or *create()*.

Parameters

- **uuid** (*str*) – internal id
- **name** (*str*) – name
- **classes** (*list*) – list of labels/classes
- **estimator_type** (*str*) – type of estimator (i.e.)
- **image_files** (*list*) – list of associated image files used for training
- **metadata** (*dict*) – user metadata
- **configuration** (*dict*) – estimator configuration
- **extra_attributes** (*dict*) – extra attributes from API endpoint

TYPES = {'object_detection': 'OD'}

add_image(**images*)

Adds an image `File` to the estimator, for training.

Parameters **images** – one or multiple image `File` instances

Returns itself

add_labels_for(*vector_file*, *image_file*, *label*)

Adds labels from an already uploaded `vector_file` related to `image_file` and tags these labels like `label`.

Parameters

- **vector_file** (`File`) – a GeoJSON file with polygons representing annotated objects
- **image_file** (`File`) – the corresponding image file to be annotated.
- **label** (*str*) – the class/label to use for the annotations

Returns itself

classmethod all()

Fetches all estimators in the current project.

base_path = '/estimators'

clone()

Clone an estimator.

Return type *Estimator*

classmethod create(*, *name*, *type*, *classes*, *metadata=None*, *configuration={}*)

Creates a new Estimator named `name` of `type` with `classes` as labels with `training_hours` hour of training job.

A metadata dictionary can be added via the `metadata` parameter. A configuration dictionary can be added via the `configuration` parameter.

Parameters

- **str** (*type*) – A name to identify the estimator
- **str** – Type of estimator ("object_detection")
- **list** (*classes*) – List of classes/labels of objects
- **dict** (*configuration*) – Optional metadata dictionary to store extra attributes
- **dict** – Optional configuration dictionary for training

Return type *Estimator*

delete()
Deletes the estimator.

Returns `True` if estimator was succesfully deleted.

classmethod get(*uuid*)
Gets an estimator identified by `uuid`.

Parameters **str** (*uuid*) – Estimator UUID

Return type *Estimator*

latest_prediction_task
Returns the most recent prediction task.

Returns a Task instance with information about the latest prediction task

Return type *Task*

latest_training_task
Returns the most recent training task.

Returns a Task instance with information about the latest training task

Return type *Task*

predict_files(*tile_dirs*, *confidence=0.2*)
Starts a prediction job with the tile images stored in `tile_dirs`.

Results are filtered through a `confidence` threshold.

Parameters

- **tile_dirs** (*list*) – list of directories with tiles to predict
- **confidence** (*float*) – confidence value for results

Returns a dict with info about the new `PredictionJob`

save()
Updates the estimator.

Returns itself

train()
Start a training job using this estimator.

It will build a custom model based on all the images from the estimator and the associates annotations.

Returns a dict with info about the new TrainingJob

dymaxionlabs.tasks module

class dymaxionlabs.tasks.**Task**(*, *id*, *state*, *name*, *args*, *kwargs*, *created_at*, *updated_at*, *finished_at*, *metadata*, *duration*, *estimated_duration*, *error*, ***extra_attributes*)

Bases: object

A Task represents a long running job.

Currently, Tasks are used to query about the status of a model training or prediction job (see `Estimator.train()` and `Estimator.predict_files()`), or an image tiling process (see `File.tiling()`).

Parameters

- **id** (*int*) – internal id
- **state** (*str*) – job state
- **name** (*str*) – task name
- **args** (*list*) – args
- **kwargs** (*dict*) – kwargs
- **created_at** (*datetime*) – created datetime
- **updated_at** (*datetime*) – updated datetime
- **finished_at** (*datetime*) – finished datetime
- **metadata** (*dict*) – job metadata
- **duration** (*int*) – duration (in seconds)
- **estimated_duration** (*int*) – estimated duration (in seconds)
- **error** (*str*) – latest error message (if task failed)
- **extra_attributes** (*dict*) – extra attributes from API response

classmethod all(*path='*'*)

Fetches all tasks.

Returns a list of *Task*

Return type list

base_path = '/tasks'

cancel()

Cancel the task if it is possible.

Returns itself

Return type *Task*

download_artifacts(*output_dir='.'*)

Downloads output artifacts in a compressed Zip file, and stores it on `output_dir`.

If `output_dir` does not exist, it will be created.

Parameters **output_dir** (*str*) – directory path where file will be stored

Returns path to the artifacts zip file

Return type str

export_artifacts(*storage_dir*)

Stores output artifacts in `storage_dir`.

Parameters **storage_dir** (*str*) – directory path where file will be stored

classmethod get(*id*)
Gets an task identified by id.

Parameters **int** (*id*) – Task id

Returns the specified *Task* instance

Return type *Task*

has_artifacts()
Checks if completed task has generated output artifacts.

Returns True if it has output artifacts, False if not.

Return type bool

is_running()
Decides whether a task is running or not, and update the task attributes if is necesary.

Return type bool

list_artifacts()
Returns a list of the generated output artifacts.

Returns list of file paths (strings)

Rtpe list

refresh()
Refreshes attributes of the task.

Returns itself

Return type *Task*

dymaxionlabs.upload module

class dymaxionlabs.upload.**CustomResumableUpload**(*upload_url*, *chunk_size*, *headers=None*)
Bases: google.resumable_media.requests.upload.ResumableUpload

initiate(*stream*, *metadata*, *content_type*, *resumable_url*, *total_bytes=None*, *stream_final=True*)
Initiate a resumable upload.

By default, this method assumes your stream is in a "final" state ready to transmit. However, stream_final=False can be used to indicate that the size of the resource is not known. This can happen if bytes are being dynamically fed into stream, e.g. if the stream is attached to application logs.

If stream_final=False is used, chunk_size bytes will be read from the stream every time *transmit_next_chunk()* is called. If one of those reads produces strictly fewer bites than the chunk size, the upload will be concluded.

Parameters

- **transport** (*Session*) – A requests object which can make authenticated requests.
- **stream** (*IO[bytes]*) – The stream (i.e. file-like object) that will be uploaded. The stream **must** be at the beginning (i.e. stream.tell() == 0).
- **metadata** (*Mapping[str, str]*) – The resource metadata, such as an ACL list.
- **content_type** (*str*) – The content type of the resource, e.g. a JPEG image has content type image/jpeg.

- **total_bytes** (*Optional[int]*) – The total number of bytes to be uploaded. If specified, the upload size **will not** be determined from the stream (even if `stream_final=True`).
- **stream_final** (*Optional[bool]*) – Indicates if the `stream` is "final" (i.e. no more bytes will be added to it). In this case we determine the upload size from the size of the stream. If `total_bytes` is passed, this argument will be ignored.

Returns The HTTP response returned by `transport`.

Return type Response

transmit_next_chunk()

Transmit the next chunk of the resource to be uploaded.

If the current upload was initiated with `stream_final=False`, this method will dynamically determine if the upload has completed. The upload will be considered complete if the stream produces fewer than `chunk_size` bytes when a chunk is read from it.

In the case of failure, an exception is thrown that preserves the failed response:

```
>>> error = None
>>> try:
...     upload.transmit_next_chunk(transport)
... except resumable_media.InvalidResponse as caught_exc:
...     error = caught_exc
...
>>> error
InvalidResponse('Request failed with status code', 400,
                'Expected one of', <HTTPStatus.OK: 200>, 308)
>>> error.response
<Response [400]>
```

Parameters **transport** (*Session*) – A `requests` object which can make authenticated requests.

Returns The HTTP response returned by `transport`.

Return type Response

Raises `InvalidResponse` – If the status code is not 200 or 308.

dymaxionlabs.utils module

exception dymaxionlabs.utils.**BadRequestError**

Bases: `Exception`

exception dymaxionlabs.utils.**InternalServerError**

Bases: `Exception`

exception dymaxionlabs.utils.**NotFoundError**

Bases: `Exception`

class dymaxionlabs.utils.**TimeoutHTTPAdapter**(**args*, ***kwargs*)

Bases: `requests.adapters.HTTPAdapter`

send(*request*, ***kwargs*)

Sends PreparedRequest object. Returns Response object.

Parameters

- **request** – The `PreparedRequest` being sent.
- **stream** – (optional) Whether to stream the request content.
- **timeout** (*float or tuple or urllib3 Timeout object*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** – (optional) Any user-provided SSL certificate to be trusted.
- **proxies** – (optional) The proxies dictionary to apply to the request.

Return type requests.Response

`dymaxionlabs.utils.`**`fetch_from_list_request`**(*path*, *params={}*)
Fetches all entities from a paginated result

`dymaxionlabs.utils.`**`get_api_key`**()
Get current API Key from environment

`dymaxionlabs.utils.`**`get_api_url`**()
Get current API URL from environment

`dymaxionlabs.utils.`**`request`**(*method*, *path*, *body=None*, *files=None*, *params={}*, *headers={}*, *binary=False*, *parse_response=True*)
Makes an HTTP request to the API

Module contents

Package to integrate the DymaxionLabs's funcionality:

- Upload images
- Predict imagenes based in object detection models
- Download results

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

dymaxionlabs, 21
dymaxionlabs.files, 14
dymaxionlabs.models, 16
dymaxionlabs.tasks, 18
dymaxionlabs.upload, 19
dymaxionlabs.utils, 20

Index

A

add_image() (*dymaxionlabs.models.Estimator method*), 16
add_labels_for() (*dymaxionlabs.models.Estimator method*), 16
all() (*dymaxionlabs.files.File class method*), 14
all() (*dymaxionlabs.models.Estimator class method*), 16
all() (*dymaxionlabs.tasks.Task class method*), 18

B

BadRequestError, 20
base_path (*dymaxionlabs.files.File attribute*), 15
base_path (*dymaxionlabs.models.Estimator attribute*), 16
base_path (*dymaxionlabs.tasks.Task attribute*), 18

C

cancel() (*dymaxionlabs.tasks.Task method*), 18
clone() (*dymaxionlabs.models.Estimator method*), 16
create() (*dymaxionlabs.models.Estimator class method*), 16
CustomResumableUpload (*class in dymaxionlabs.upload*), 19

D

delete() (*dymaxionlabs.files.File method*), 15
delete() (*dymaxionlabs.models.Estimator method*), 17
download() (*dymaxionlabs.files.File method*), 15
download_artifacts() (*dymaxionlabs.tasks.Task method*), 18
dymaxionlabs (*module*), 21
dymaxionlabs.files (*module*), 14
dymaxionlabs.models (*module*), 16
dymaxionlabs.tasks (*module*), 18
dymaxionlabs.upload (*module*), 19
dymaxionlabs.utils (*module*), 20

E

Estimator (*class in dymaxionlabs.models*), 16
export_artifacts() (*dymaxionlabs.tasks.Task method*), 18

F

fetch_from_list_request() (*in module dymaxionlabs.utils*), 21
File (*class in dymaxionlabs.files*), 14

G

get() (*dymaxionlabs.files.File class method*), 15
get() (*dymaxionlabs.models.Estimator class method*), 17
get() (*dymaxionlabs.tasks.Task class method*), 19
get_api_key() (*in module dymaxionlabs.utils*), 21
get_api_url() (*in module dymaxionlabs.utils*), 21

H

has_artifacts() (*dymaxionlabs.tasks.Task method*), 19

I

initiate() (*dymaxionlabs.upload.CustomResumableUpload method*), 19
InternalServerError, 20
is_running() (*dymaxionlabs.tasks.Task method*), 19

L

latest_prediction_task (*dymaxionlabs.models.Estimator attribute*), 17
latest_training_task (*dymaxionlabs.models.Estimator attribute*), 17
list_artifacts() (*dymaxionlabs.tasks.Task method*), 19

N

NotFoundError, 20

P

predict_files() (*dymaxionlabs.models.Estimator method*), 17

R

refresh() (*dymaxionlabs.tasks.Task method*), 19
request() (*in module dymaxionlabs.utils*), 21

S

save() (*dymaxionlabs.models.Estimator method*), 17
send() (*dymaxionlabs.utils.TimeoutHTTPAdapter method*), 20

T

Task (*class in dymaxionlabs.tasks*), 18
tiling() (*dymaxionlabs.files.File method*), 15
TimeoutHTTPAdapter (*class in dymaxionlabs.utils*), 20
train() (*dymaxionlabs.models.Estimator method*), 17
transmit_next_chunk() (*dymaxionlabs.upload.CustomResumableUpload method*), 20
TYPES (*dymaxionlabs.models.Estimator attribute*), 16

U

upload() (*dymaxionlabs.files.File class method*), 15