Datmo Documentation

Release 0.0.7-dev

Anand Sampat

May 22, 2018

Contents

Datmo is an open source model tracking tool for developers

CHAPTER 1

Why we built this

As data scientists, machine learning engineers, and deep learning engineers, we faced a number of issues keeping track of our work and maintaining versions that could be put into production quicker.

In order to solve this challenge, we found there are a few components that are critical to ensuring this is the case.

1. Source code should be managed with current source control management tools (of which git is the most popular currently)
2. Dependencies should be encoded in one place for your source code (e.g. requirements.txt in python and pre-built containers)
3. Large files that cannot be stored in source code like weights files, data files, etc should be stored separately
4. Configurations and hyperparameters that define your experiments (e.g. data split, alpha, beta, etc)
5. Performance metrics that evaluate your model (e.g. validation accuracy)

We've encapsulated these concepts in an object called a *snapshot*. A snapshot is a combination of all 5 of the above components and is the way that Datmo versions models for reproducibility and deployability. Our open source tool is an interface for developers to transform their current model projects into trackable models that can be used for transportability throughout the model building process.

We have used this internally to speed up our own iteration processes and are excited to share it with the community to continue improving. If you're interested in contributing check out the guidelines.

CHAPTER 2

Table of contents

2.1 Command Line Utility

The command line utility for datmo is to be used in tandem with the SDK and will typically be your first contact with the datmo system. If using Python, see *Python SDK*.

If you are working within a repository already, you will want to run the `datmo init` within your repository in order to create your datmo project.

From there, you can create snapshots or run tasks using either the SDK or the CLI. At any given point you can find out more about all of your snapshots using the `datmo snapshot ls` command and see the status of any of your tasks with the `datmo task ls` command.

Sessions are a way for you to group together tasks and snapshots, but are completely optional. For example, if you want to run a set of hyperparameter experiments modifying some subset of hyperparameters you might want to do them in a designated session. Then you might try another set of hyperparameter sweeps which you would like to group into another session. By default, you will always be in the "default" session unless otherwise specified.

You can delve through more of the commands and each of their parameters below to learn more about each entity and how you can create different versions of them. You can also look through the Getting Started section in the README.

```
usage: datmo [-h] {init,version,status,cleanup,session,snapshot,task} ...
```

2.1.1 commands

command Possible choices: init, version, status, cleanup, session, snapshot, task

2.1.2 Sub-commands:

init

initialize project

```
datmo init [-h] [--name NAME] [--description DESCRIPTION]
```

Named Arguments

--name

--description

version

datmo version

```
datmo version [-h]
```

status

project status

```
datmo status [-h]
```

cleanup

remove project

```
datmo cleanup [-h]
```

session

session module

```
datmo session [-h] {create,delete,ls,select} ...
```

subcommands

subcommand Possible choices: create, delete, ls, select

Sub-commands:

create

create session

```
datmo session create [-h] [--name NAME] [--current]
```

Named Arguments

--name, -m	session name Default: “”
--current	boolean if you want to switch to this session Default: True

delete

delete a session by id

```
datmo session delete [-h] [--name NAME]
```

Named Arguments

--name	name of session to delete

ls

list sessions

```
datmo session ls [-h]
```

select

select a session

```
datmo session select [-h] [--name NAME]
```

Named Arguments

--name	name of session to select

snapshot

Datmo snapshots allow you to save the state of your model and experiments by keeping track of your source code, environment, configuration, metrics and large files.

```
datmo snapshot [-h] {create,delete,ls,checkout,diff,inspect} ...
```

subcommands

subcommand	Possible choices: create, delete, ls, checkout, diff, inspect

Sub-commands:

create

Run snapshot create any time you want to save the results of your experiments. You can then view all snapshots with the *snapshot ls* command.

```
datmo snapshot create [-h] [--message MESSAGE] [--label LABEL]
                      [--session-id SESSION_ID] [--task-id TASK_ID]
                      [--code-id CODE_ID] [--commit-id COMMIT_ID]
                      [--environment-id ENVIRONMENT_ID]
                      [--environment-def ENVIRONMENT_DEFINITION_FILEPATH]
                      [--file-collection-id FILE_COLLECTION_ID]
                      [--filepaths FILEPATHS]
                      [--config-filename CONFIG_FILENAME]
                      [--config-filepath CONFIG_FILEPATH]
                      [--stats-filename STATS_FILENAME]
                      [--stats-filepath STATS_FILEPATH]
```

Named Arguments

--message, -m	message to describe snapshot
--label, -l	label snapshots with a category (e.g. best)
--session-id	user given session id
--task-id	specify task id to pull information from
--code-id	code id from code object
--commit-id	commit id from source control
--environment-id	environment id from environment object
--environment-def	absolute filepath to environment definition file (e.g. /path/to/Dockerfile)
--file-collection-id	file collection id for file collection object
--filepaths	absolute paths to files or folders to include within the files of the snapshot
--config-filename	filename to use to search for configuration JSON
--config-filepath	absolute filepath to use to search for configuration JSON
--stats-filename	filename to use to search for metrics JSON
--stats-filepath	absolute filepath to use to search for metrics JSON

delete

delete a snapshot by id

```
datmo snapshot delete [-h] [--id ID]
```

Named Arguments

--id snapshot id to delete

ls

list snapshots

```
datmo snapshot ls [-h] [--session-id SESSION_ID] [--all]
```

Named Arguments

--session-id session id to filter

--all, -a show detailed snapshot information

Default: False

checkout

checkout a snapshot by id

```
datmo snapshot checkout [-h] id
```

Positional Arguments

id snapshot id

diff

view diff between 2 snapshots

```
datmo snapshot diff [-h] id_1 id_2
```

Positional Arguments

id_1 snapshot id 1

id_2 snapshot id 2

inspect

inspect a snapshot by id

```
datmo snapshot inspect [-h] id
```

Positional Arguments

id snapshot id

task

task module

```
datmo task [-h] {run,ls,stop} ...
```

subcommands

subcommand Possible choices: run, ls, stop

Sub-commands:

run

run task

```
datmo task run [-h] [--gpu] [--ports PORTS]
               [--environment-def ENVIRONMENT_DEFINITION_FILEPATH]
               [--interactive]
               [cmd]
```

Positional Arguments

cmd command to run within environment

Named Arguments

--gpu boolean if you want to run using GPUs

Default: False

--ports, -p network port mapping during task (e.g. 8888:8888). Left is the host machine port and right is the environment port available during a run.

--environment-def absolute filepath to environment definition file (e.g. /path/to/Dockerfile)

--interactive run the environment in interactive mode (keeps STDIN open)

Default: False

ls

list tasks

```
datmo task ls [-h] [--session-id [SESSION_ID]]
```

Named Arguments

--session-id	pass in the session id to list the tasks in that session

stop

stop tasks

```
datmo task stop [-h] [--id ID] [--all]
```

Named Arguments

--id	task id to stop
--all, -a	stop all datmo tasks Default: False

2.2 Python SDK

Datmo's Python SDK is a way to create datmo snapshots and run tasks directly within your code. Although the SDK is not necessary for using datmo, it helps simplify the process of integrating your current code with current Python projects. If you aren't using Python, see *Command Line Utility*.

2.2.1 datmo.snapshot module

class datmo.snapshot.**Snapshot**(*snapshot_entity*, *home=None*)

Snapshot is an entity object to enable user access to properties

Parameters

- **snapshot_entity** (*datmo.core.entity.snapshot.Snapshot*) – core snapshot entity to reference
- **home** (*str, optional*) – root directory of the project (default is CWD, if not provided)

id

str – the id of the entity

model_id

str – the parent model id for the entity

session_id

str – id of session associated with task

id

str – the id of the entity

model_id

str – the parent model id for the entity

session_id

str – session id within which snapshot is created

message
str – long description of snapshot

code_id
str – code reference associated with the snapshot

environment_id
str – id for environment used to create snapshot

file_collection_id
str – file collection associated with the snapshot

config
dict – key, value pairs of configurations

stats
dict – key, value pairs of metrics and statistics

task_id
str – task id associated with snapshot

label
str – short description of snapshot

created_at
datetime.datetime

Raises `InvalidArgumentType`

files

get_files(*mode='r'*)
Returns a list of file objects for the snapshot

Parameters **mode** (*str*) – file object mode (default is "r" which signifies read mode)

Returns list of file objects associated with the snapshot

Return type list

datmo.snapshot.**create**(*message*, *label=None*, *home=None*, *task_id=None*, *commit_id=None*, *environment_id=None*, *filepaths=None*, *config=None*, *stats=None*)
Create a snapshot within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **message** (*str*) – a description of the snapshot for later reference
- **label** (*str, optional*) – a short description of the snapshot for later reference (default is None, which means a blank label is stored)
- **home** (*str, optional*) – absolute home path of the project (default is None, which will use the CWD as the project path)
- **task_id** (*str, optional*) – task object id to use to create snapshot if task id is passed then subsequent parameters would be ignored. when using task id, it will overwrite the following inputs

  commit_id: taken form the source code after the task is run

environment_id: used to run the task,

filepaths: this is the set of all files saved during the task

config: nothing is passed into this variable. the user may add something to the config by passing in a dict for the config

stats: the task.results are added into the stats variable of the snapshot.

- **commit_id** (*str, optional*) – provide the exact commit hash associated with the snapshot (default is None, which means it automatically creates a commit)
- **environment_id** (*str, optional*) – provide the environment object id to use with this snapshot (default is None, which means it creates a default environment)
- **filepaths** (*list, optional*) – provides a list of absolute filepaths to files or directories that are relevant (default is None, which means we have an empty
- **config** (*dict, optional*) – provide the dictionary of configurations (default is None, which means it is empty)
- **stats** (*dict, optional*) – provide the dictionary of relevant statistics or metrics (default is None, which means it is empty)

Returns returns a Snapshot entity as defined above

Return type *Snapshot*

Examples

You can use this function within a project repository to save snapshots for later use. Once you have created this, you will be able to view the snapshot with the *datmo snapshot ls* cli command

```
>>> import datmo
>>> datmo.snapshot.create(message="my first snapshot", filepaths=["/path/to/a/
↪large/file"], config={"test": 0.4, "test2": "string"}, stats={"accuracy": 0.94})
```

You can also use the result of a task run in order to create a snapshot

```
>>> datmo.snapshot.create(message="my first snapshot from task", task_id=
↪"1jfkshg049")
```

datmo.snapshot.**ls**(*session_id=None*, *filter=None*, *home=None*)

List snapshots within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **session_id** (*str, optional*) – session to filter output snapshots (default is None, which means no session filter is given)
- **filter** (*str, optional*) – a string to use to filter from message and label (default is to give all snapshots, unless provided a specific string. eg: best)
- **home** (*str, optional*) – absolute home path of the project (default is None, which will use the CWD as the project path)

Returns returns a list of Snapshot entities (as defined above)

Return type list

Examples

You can use this function within a project repository to list snapshots.

```
>>> import datmo
>>> snapshots = datmo.snapshot.ls()
```

2.2.2 datmo.task module

class datmo.task.**Task**(*task_entity*, *home=None*)

Task is an entity object to enable user access to properties

Parameters

- **task_entity** (*datmo.core.entity.task.Task*) – core task entity to reference
- **home** (*str, optional*) – root directory of the project (default is CWD, if not provided)

id

str – the id of the entity

model_id

str – the parent model id for the entity

session_id

str – id of session associated with task

command

str – command that is used by the task

status

str or None – status of the current task

start_time

datetime.datetime or None – timestamp for the beginning time of the task

end_time

datetime.datetime or None – timestamp for the end time of the task

duration

float or None – delta in seconds between start and end times

logs

str or None – string output of logs

results

dict or None – dictionary containing output results from the task

files

list – returns list of file objects for the task in read mode

get_files(*mode="r"*)

Returns a list of file objects for the task

Raises InvalidArgumentType

duration

end_time

files

get_files(*mode='r'*)
Returns a list of file objects for the task

Parameters **mode** (*str*) – file object mode (default is "r" which signifies read mode)

Returns list of file objects associated with the task

Return type list

logs

results

start_time

status

datmo.task.**ls**(*session_id=None*, *filter=None*, *home=None*)
List tasks within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **session_id** (*str, optional*) – session to filter output tasks (default is None, which means no session filter is given)
- **filter** (*str, optional*) – a string to use to filter from message and label (default is to give all snapshots, unless provided a specific string. eg: best)
- **home** (*str, optional*) – absolute home path of the project (default is None, which will use the CWD as the project path)

Returns returns a list of Task entities (as defined above)

Return type list

Examples

You can use this function within a project repository to list tasks.

```
>>> import datmo
>>> tasks = datmo.task.ls()
```

datmo.task.**run**(*command*, *env=None*, *home=None*, *gpu=False*)
Run the code or script inside

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **command** (*str or list*) – the command to be run in environment. this can be either a string or list

- **env** (*str, optional*) – the location for the environment definition path (default is None, which will defer to the environment to find a default environment, or will fail if not found)
- **home** (*str, optional*) – absolute home path of the project (default is None, which will use the CWD as the project path)
- **gpu** (*boolean*) – try to run task on GPU (if available)

Returns returns a Task entity as defined above

Return type *Task*

Examples

You can use this function within a project repository to run tasks in the following way.

```
>>> import datmo
>>> datmo.task.run(command="python script.py")
>>> datmo.task.run(command="python script.py", env='Dockerfile')
```

2.2.3 datmo.config module

class datmo.config.**Config**

Bases: object

Datmo Config properties

Parameters

- **home** (*string*) – project home directory
- **logging_level** (*int*) – logging level

Returns Config Singleton

Return type *Config*

static cache_setting(**args*, ***kwargs*)

instance = None

2.3 Examples

In order to run the examples, make sure that you have datmo properly installed with the latest stable or development version. You can install it with the following command:

```
$ pip install datmo
```

2.3.1 Using the Examples

CLI flow

See CLI flow examples for instructions

CLI + Python flow

See CLI + Python flow examples for instructions

CLI + Jupyter Notebook flow

See CLI + Jupyter Notebook flow examples for instructions

2.3.2 Examples

Listed below are actions you might want to take with Datmo. For each we have listed if there are any example for each type of flow. You can navigate to the specific flow folder to find the exact instructions for each example.

Creating a Snapshot

- CLI flow
  - snapshot_create_iris_sklearn
- CLI + Python flow
  - snapshot_create_iris_sklearn
- CLI + Jupyter Notebook flow
  - snapshot_create_iris_sklearn

Running a containerized task (with option to create Snapshot)

- CLI + Python flow
  - task_run_iris_sklearn_basic
  - task_run_iris_sklearn_compare

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index