datarun Documentation

Release 0.1

Camille

May 04, 2016

Contents:

HOW TO USE DATARUN?

The workflow to use datarun is the following:

1.1 1- Send data to datarun

The standard format of a data file excepted by datarun is a csv file whose first row contains the feature and target names, each line corresponds to a data sample.

Here is an example of a standard data file:

```
sepal length,sepal width,petal length,petal width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
```

If your data match the standard data file, you need to send: - the name of the data set (for instance if you use databoard, you can use the problem name) - your data file - the name of the target column - the workflow elements of the problem related to the dataset (for instance feature_extractor, classifier, ...)

If your data do not match the standard data file, you need to send in addition to above:

- a python file with three specific functions: `prepare_data(raw_data_path)`, `get_train_data(raw_data_path)`, and `get_test_data(raw_data_path)`. An example of such file is `test_files/variable_stars/variable_stars_datarun.py`.

- possibly other data files (if your data are split in different files).

In both cases, to send you data to datarun, you can use:

- a post request to `<master-host>/runapp/rawdata/` (cf *direct requests*, class runapp.views.RawDataList)

- the `post_data` function in the module test_files.post_api (cf *post_api module*)

Note for databoard users:

1.2 2- Split data into train and test dataset

If your data match the standard format, you can use:

- a post request to `<master-host>/runapp/rawdata/split/` (cf *direct requests*, class runapp.views.SplitTrainTest)

- the `post_split` function in the module test_files.post_api (cf *post_api module*)

If your data do not match the standard format, you can use:

- a post request to `<master-host>/runapp/rawdata/customsplit/` (cf *direct requests*, class runapp.views.CustomSplitTrainTest)

- the `custom_post_split` function in the module test_files.post_api (cf *post_api module*)

1.3 3- Send submission on cv fold to be trained on datarun

To send a submission on cv fold, you can use:

- a post request to `<master-host>/runapp/submissionfold/` (cf *direct requests*, class runapp.views.SubmissionFoldList)

- the `post_submission_fold` function in the module test_files.post_api (cf *post_api module*)

1.4 4- Get back your predictions

MODELS

class runapp.models.**RawData**(**args*, ***kwargs*)

> **Parameters**
>
> - **name** (*string*) – name of the data set
> - **files_path** (*string*) – path of file where data are saved
> - **workflow_elements** (*string*) – list of workflow elements used to solve the RAMP
> - **column** (*target*) – name of the target column

class runapp.models.**Submission**(**args*, ***kwargs*)

> **Parameters**
>
> - **databoard_s_id** (*IntegerField(primary_key=True)*) – id of the submission in the db of databoard
> - **files_path** (*CharField(max_length=200, null=True)*) – path of submitted files
> - **raw_data** (*ForeignKey(RawData, null=True, blank=True)*) – associated raw data

class runapp.models.**SubmissionFold**(**args*, ***kwargs*)

> **Parameters**
>
> - **databoard_sf_id** (*IntegerField(primary_key=True)*) – id of the submission on cv fold in databoard db
> - **databoard_s** (*ForeignKey(Submission, null=True, blank=True)*) – associated submission
> - **train_is** (*TextField*) – train indices
> - **test_is** (*TextField*) – test indices
> - **priority** (*CharField, choices.*) – priority to train-test the fold ('L' for low priority, 'H' for high priority)
> - **full_train_predictions** (*TextField*) – predictions of the entire train dataset
> - **test_predictions** (*TextField*) – predictions of the test dataset
> - **state** (*CharField, choices.*) – TODO, TRAINED, VALIDATED, TESTED, ERROR
> - **log_messages** (*TextField*) – logs recorded during train and test
> - **train_time** (*FloatField, default=0.*) – real clock training time

- **validation_time** (*FloatField, default=0.*) – real clock validation time

- **test_time** (*FloatField, default=0.*) – real clock testing time

- **train_cpu_time** (*FloatField, default=0.*) – training cpu time

- **train_memory** – peak memory usage during train and test (in kb)

- **test_cpu_time** – test cpu time

- **test_memory** (*FloatField, default=0.*) – peak memory usage durnig train and test (in kb)

- **new** (*BooleanField, default=True.*) – True when it has not already been sent by the API

REQUESTS

You can either make direct requests to the datarun API, or use the post_api function.

3.1 direct requests

class `runapp.views.`**CustomSplitTrainTest**(***kwargs*)

Split data set into train and test datasets for custom dataset (when a specific.py was submitted along with raw data

post(*request*, *format=None*)

Split raw data into train and test datasets for custom dataset

•Example with curl (on localhost):

curl -u username:password -H "Content-Type: application/json" -X POST -d '{"raw_data_id": 1}' http://127.0.0.1:8000/runapp/rawdata/customsplit/

Don't forget double quotes for the json, simple quotes do not work

•Example with the python package requests (on localhost):

requests.post('http://127.0.0.1:8000/runapp/raw_data/customsplit/', auth=('username', 'password'), json={'raw_data_id': 1})

— parameters:

•name: raw_data_id description: id of the raw dataset required: true type: integer paramType: form

class `runapp.views.`**GetTestPredictionList**(***kwargs*)

Get predictions of submissions on cv fold given their ids

post(*request*, *format=None*)

Retrieve predictions (on the test data set) of SubmissionFold instances among a list of id that have been trained and tested

•Example with curl (on localhost):

curl -u username:password -H "Content-Type: application/json" -X POST -d '{"list_submission_fold": [1, 2, 10]}' http://127.0.0.1:8000/runapp/testpredictions/list/

Don't forget double quotes for the json, simple quotes do not work

•Example with the python package requests (on localhost):

requests.post('http://127.0.0.1:8000/runapp/testpredictions/list/', auth=('username', 'password'), json={'list_submission_fold': [1, 2, 10]})

— parameters:

> •name: list_submission_fold description: list of submission on cv fold ids required: true type: list
>   paramType: form

> response_serializer: TestPredSubmissionFoldSerializer

class `runapp.views.`**`GetTestPredictionNew`**(*****kwargs*)
> Get predictions of submissions on cv fold that have not been requested

> **post**(*request*, *format=None*)
>> Retrieve predictions (on the test data set) of SubmissionFold instances that have been trained and tested
>> and not yet requested. You can specify a given data challenge by posting the raw_data id.

>> •Example with curl (on localhost):

>>> curl -u username:password -H "Content-Type: application/json" -X POST -d
>>> '{"raw_data_id": 1}' http://127.0.0.1:8000/runapp/testpredictions/new/

>>> Don't forget double quotes for the json, simple quotes do not work

>> •Example with the python package requests (on localhost):

>>> requests.post('http://127.0.0.1:8000/runapp/testpredictions/new/', auth=('username', 'pass-
>>> word'), json={'raw_data_id': 1})

>> — parameters:

>>> •name: raw_data_id description: id of the raw dataset from which to get predictions required: false
>>>   type: integer paramType: form

>> response_serializer: TestPredSubmissionFoldSerializer

class `runapp.views.`**`RawDataList`**(*****kwargs*)
> List all data set or submit a new one

> **get**(*request*, *format=None*)
>> List all raw dataset

>> •Example with curl (on localhost):

>> curl -u username:password GET http://127.0.0.1:8000/runapp/rawdata/

>> •Example with the python package requests (on localhost):

>> requests.get('http://127.0.0.1:8000/runapp/rawdata/', auth=('username', 'password'))

>> — response_serializer: RawDataSerializer

> **post**(*request*, *format=None*)
>> Create a new dataset

>> You have to post the name of the dataset, the target column, the workflow elements, and the raw data file. If
>> your data file does not match the format expected by datarun (a csv with a first row containing the feature
>> and target column name, and then a row for each sample), you can submit a python file containing three
>> functions: prepare_data(data_path), get_train_data(data_path), and get_test_data(data_path)

>> •Example with curl (on localhost):

>>> curl -u username:password -H "Content-Type: application/json" -X POST -d '{"name":
>>> "iris", "target_column": "species", "workflow_elements": "classifier", "files": {"iris.csv":
>>> 'blablabla', 'specific.py': 'bli'}}' http://127.0.0.1:8000/runapp/rawdata/

>>> Don't forget double quotes for the json, simple quotes don't work.

>> •Example with the python package requests (on localhost):

requests.post('http://127.0.0.1:8000/runapp/rawdata/', auth=('username', 'password'), json={'name': 'iris', 'target_column': 'species', 'workflow_elements': 'classifier', 'files': {'iris.csv': 'bla', 'specific.py': 'bli'}})

— request_serializer: RawDataSerializer response_serializer: RawDataSerializer

class runapp.views.**SplitTrainTest**(***kwargs*)

Split data set into train and test datasets for normal dataset

post(*request*, *format=None*)

Split raw data into train and test datasets for normal dataset

• Example with curl (on localhost):

curl -u username:password -H "Content-Type: application/json" -X POST -d '{"random_state": 42, "held_out_test": 0.7, "raw_data_id": 1}' http://127.0.0.1:8000/runapp/rawdata/split/

Don't forget double quotes for the json, simple quotes do not work

• Example with the python package requests (on localhost):

requests.post('http://127.0.0.1:8000/runapp/raw_data/split/', auth=('username', 'password'), json={'random_state': 42, 'held_out_test': 0.7, 'raw_data_id': 1})

— parameters:

• name: random_state description: random state used to split data required: false type: integer paramType: form

• name: held_out_test description: percentage of the dataset kept as test dataset required: true type: float paramType: form

• name: raw_data_id description: id of the raw dataset required: true type: integer paramType: form

class runapp.views.**SubmissionFoldDetail**(***kwargs*)

Get a submission on CV fold given its id

get(*request*, *pk*, *format=None*)

Retrieve a SubmissionFold instance to check its state

• Example with curl (on localhost):

curl -u username:password GET http://127.0.0.1:8000/runapp/submissionfold/10/

• Example with the python package requests (on localhost):

requests.get('http://127.0.0.1:8000/runapp/submissionfold/10/', auth=('username', 'password'))

— parameters:

• name : pk description: id of the submission on cv fold in the databoard db required: true type: interger paramType: path

response_serializer: SubmissionFoldSerializer

class runapp.views.**SubmissionFoldLightList**(***kwargs*)

To get main info about all submissions on CV fold

get(*request*, *format=None*)

List main info (id, submission id, state, new) about all submissions on CV fold

• Example with curl (on localhost):

curl -u username:password GET http://127.0.0.1:8000/runapp/submissionfold-light/

- Example with the python package requests (on localhost):

    requests.get('http://127.0.0.1:8000/runapp/submissionfold-light/', auth=('username', 'password'))

  — response_serializer: SubmissionFoldLightSerializer

class `runapp.views.` **`SubmissionFoldList`** (***kwargs*)

To get all submissions on CV fold

`get` (*request*, *format=None*)

List all submission on CV fold

- Example with curl (on localhost):

    curl -u username:password GET http://127.0.0.1:8000/runapp/submissionfold/

- Example with the python package requests (on localhost):

    requests.get('http://127.0.0.1:8000/runapp/submissionfold/', auth=('username', 'password'))

  — response_serializer: SubmissionFoldSerializer

`post` (*request*, *format=None*)

Create a submission on CV fold (and if necessary the associated submission)

- Example with curl (on localhost):

    curl -u username:password -H "Content-Type: application/json" -X POST -d '{"databoard_s_id": 1, "files": {"classifier.py": "import sklearn.."}, "train_is": "hgjhg", "raw_data":1, "databoard_sf_id": 11, "test_is": "kdjhLGf2", "priority": "L"}' http://127.0.0.1:8000/runapp/submissionfold/

    Don't forget double quotes for the json, simple quotes do not work

- Example with the python package requests (on localhost):

    requests.post('http://127.0.0.1:8000/runapp/submissionfold/', auth=('username', 'password'), json={'databoard_sf_id': 10, 'databoard_s_id': 24, 'raw_data': 8, 'train_is': 'GDHRFdfgfd', 'test_is': 'kdjhLGf2', 'priority': 'L' 'files': {'classifier.py': 'import skle...'}})

  Possible to force the submission and submission on CV fold (even if the ids already exist) by adding to the data dictionary "force": 'submission, submission_fold' to resubmit both, or "force": 'submission_fold' to resubmit only the submission on CV fold

  — request_serializer: SubmissionFoldSerializer response_serializer: SubmissionFoldSerializer

`runapp.views.` **`save_files`** (*dir_data*, *data*)

save files from data['files'] in directory dir_data

3.2 post_api module

`test_files.post_api.` **`custom_post_split`** (*host_url*, *username*, *password*, *raw_data_id*)

To split data between train and test on datarun using a specific prepare_data function sent by databoard

Parameters

- **`host_url`** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)

- **`username`** (*string*) – username to be used for authentication

- **password** (*string*) – password to be used for authentication

- **raw_data_id** (*integer*) – id of the raw dataset on datarun

`test_files.post_api.`**`get_prediction_list`**(*host_url*, *username*, *password*, *list_submission_fold_id*)

Get predictions given a list of submission on cv fold ids

> **Parameters**
>
> - **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
>
> - **username** (*string*) – username to be used for authentication
>
> - **password** (*string*) – password to be used for authentication
>
> - **list_submission_fold_id** (*list*) – list of submission on cv fold ids from which we want the predictions

`test_files.post_api.`**`get_prediction_new`**(*host_url*, *username*, *password*, *raw_data_id*)

Get all new predictions given a raw data id

> **Parameters**
>
> - **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
>
> - **username** (*string*) – username to be used for authentication
>
> - **password** (*string*) – password to be used for authentication
>
> - **raw_data_id** (*integer*) – id of a data set from which we want new predictions

`test_files.post_api.`**`get_raw_data`**(*host_url*, *username*, *password*)

Get all raw data sets

> **Parameters**
>
> - **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
>
> - **username** (*string*) – username to be used for authentication
>
> - **password** (*string*) – password to be used for authentication

`test_files.post_api.`**`get_submission_fold`**(*host_url*, *username*, *password*)

Get all submission on cv fold (all attributes)

> **Parameters**
>
> - **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
>
> - **username** (*string*) – username to be used for authentication
>
> - **password** (*string*) – password to be used for authentication

`test_files.post_api.`**`get_submission_fold_detail`**(*host_url*, *username*, *password*, *submission_fold_id*)

Get details about a submission on cv fold given its id

> **Parameters**
>
> - **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
>
> - **username** (*string*) – username to be used for authentication
>
> - **password** (*string*) – password to be used for authentication
>
> - **submission_fold_id** – id of the submission on cv fold
>
> - **submission_fold_id** – integer

`test_files.post_api.`**`get_submission_fold_light`**(*host_url*, *username*, *password*)
  Get all submissions on cv fold only main info: id, associated submission id, state, and new

    Parameters

      • **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)

      • **username** (*string*) – username to be used for authentication

      • **password** (*string*) – password to be used for authentication

`test_files.post_api.`**`post_data`**(*host_url*, *username*, *password*, *data_name*, *target_column*, *work-flow_elements*, *data_file*, *extra_files=None*)
  To post data to the datarun api. Data are compressed (with zlib) and base64-encoded before being posted.

    Parameters

      • **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)

      • **username** (*string*) – username to be used for authentication

      • **password** (*string*) – password to be used for authentication

      • **data_name** (*string*) – name of the raw dataset

      • **target_column** (*string*) – name of the target column

      • **workflow_elements** (*string*) – workflow elements associated with this dataset, e.g., feature_extractor, classifier

      • **data_file** (*string*) – name with absolute path of the dataset file

      • **extra_files** (*list of string*) – list of names with absolute path of extra files (such as a specific.py)

`test_files.post_api.`**`post_split`**(*host_url*, *username*, *password*, *held_out_test*, *raw_data_id*, *random_state=42*)
  To split data between train and test on datarun

    Parameters

      • **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)

      • **username** (*string*) – username to be used for authentication

      • **password** (*string*) – password to be used for authentication

      • **held_out_test** (*float (between 0 and 1)*) – ratio of data for the test set

      • **raw_data_id** (*integer*) – id of the raw dataset on datarun

      • **random_state** (*integer*) – random state to be used in the shuffle split

`test_files.post_api.`**`post_submission_fold`**(*host_url*, *username*, *password*, *sub_id*, *sub_fold_id*, *train_is*, *test_is*, *priority='L'*, *raw_data_id=None*, *list_submission_files=None*, *force=None*)
  To post submission on cv fold and submission (if not already posted). Submission files are compressed (with zlib) and base64-encoded before being posted.

    Parameters

      • **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)

      • **username** (*string*) – username to be used for authentication

      • **password** (*string*) – password to be used for authentication

---

- **sub_id** (*integer*) – id of the submission on databoard
- **sub_fold_id** (*integer*) – id of the submission on cv fold on databoard
- **train_is** (*numpy array*) – train indices for the cv fold
- **test_is** (*numpy array*) – test indices for the cv fold
- **priority** (*string*) – priority level to train test the model: L for low and H for high
- **raw_data_id** (*integer*) – id of the associated data, when submitting a submission
- **list_submission_files** (*list*) – list of files of the submission, when submitting a submission
- **force** (*string*) – to force the submission even if ids already exist force can be 'submission, submission_fold' to resubmit both or 'submission, submission_fold' to resubmit only the submission on cv fold. None by default.

INDICES AND TABLES

- genindex
- modindex
- search

r

runapp.models, 5
runapp.views, 7

t

test_files.post_api, 10