ml2p Documentation

Release 0.2.0

Prodigy Finance

Sep 23, 2020

CONTENTS:

CHAPTER

ONE

OVERVIEW

Todo: Write the overview.

CHAPTER

TWO

INSTALLATION

Install ML2P with:

```
$ pip install ml2p
```

And you're ready to go!

CHAPTER

THREE

TUTORIAL

Welcome to ML2P! In this tutorial we'll take you through:

- setting up your first project
- uploading a training dataset to S3
- training a model
- deploying a model
- making predictions

Throughout all of this we'll be working with the classic Boston house prices dataset that is available within scikit-learn.

3.1 Setting up your project

Before running ML2P you'll need to create:

- a docker image,
- a S3 bucket,
- and an AWS role

yourself. ML2P does not manage these for you.

Once you have the docker image, bucket and role set up, you are ready to create your ML2P configuration file. Save the following as *ml2p.yml*:

```
project: "ml2p-tutorial"
s3folder: "s3://your-s3-bucket/"
models:
  boston: "models.BostonModel"
defaults:
  image: "XXXXX.dkr.ecr.REGION.amazonaws.com/your-docker-image:X.Y.Z"
  role: "arn:aws:iam::XXXXX:role/your-role"
train:
  instance_type: "ml.m5.large"
deploy:
  instance_type: "ml.t2.medium"
  record_invokes: true  # record predictions in the S3 bucket
notebook:
  instance_type: "ml.t2.medium"
  volume_size: 8  # Size of the notebook server disk in GB
```

The yellow high-lights show the lines where you'll need to fill in the details for the docker image, S3 bucket and AWS role.

3.2 Initialize the ML2P project

You're now ready to run your first ML2P command!

First set the AWS profile you'd like to use:

```
$ export AWS_PROFILE="my-profile"
```

You'll need to set the *AWS_PROFILE* or otherwise provide your AWS credentials whenever you run an *ml2p* command.

If you haven't initialized your project before, run:

```
$ ml2p init
```

which will create the S3 model and dataset folders for you.

Once you've run *ml2p init*, ML2P have will created the following folder structure in your S3 bucket:

```
s3://your-s3-bucket/
  models/
    ... ml2p will place the outputs of your training jobs here ...
  datasets/
    ... ml2p will place your datasets here, the name of the
        subfolder is the name of the dataset ...
```

3.3 Creating a training dataset

First create a CSV file containing the Boston house prices that you'll be using to train your model. You can do this by saving the file below as *create_boston_prices_csv.py*:

```
# -*- coding: utf-8 -*-

""" A small script for creating a Boston house price data training set.
"""

import pandas
import sklearn.datasets


def write_boston_csv(csv_name):
    """ Write a Boston house price training dataset. """
    boston = sklearn.datasets.load_boston()
    df = pandas.DataFrame(boston["data"], columns=boston["feature_names"])
    df["target"] = boston["target"]
    df.to_csv(csv_name, index=False)


if __name__ == "__main__":
    write_boston_csv("house-prices.csv")
```

and running:

```
$ python create_boston_prices_csv.py
```

This will write the file *house-prices.csv* to the current folder.

Now create a training dataset and upload the CSV file to it:

```
$ ml2p dataset create boston-20200901
$ ml2p dataset up boston-20200901 house-prices.csv
```

And check that the contents of the training dataset is as expected by listing the files in it:

```
$ ml2p dataset ls boston-20200901
```

3.4 Training a model

You'll need to start by implementing a subclass of *ml2p.core.ModelTrainer*. Your subclass needs to define a *.train(...)* method that will load the training set, train the model, and save it.

A simple implementation for the Boston house price model can be found in *model.py*:

```
# -*- coding: utf-8 -*-

""" A model for predicting Boston house prices (part of the ML2P tutorial).
"""

import jsonpickle

import pandas as pd
from ml2p.core import Model, ModelPredictor, ModelTrainer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split


class BostonTrainer(ModelTrainer):
    def train(self):
        """ Train the model. """
        training_channel = self.env.dataset_folder()
        training_csv = str(training_channel / "house-prices.csv")
        df = pd.read_csv(training_csv)
        y = df["target"]
        X = df.drop(columns="target")
        features = sorted(X.columns)
        X = X[features]
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
        model = LinearRegression().fit(X_train, y_train)
        with (self.env.model_folder() / "boston-model.json").open("w") as f:
            f.write(jsonpickle.encode({"model": model, "features": features}))


class BostonPredictor(ModelPredictor):
    def setup(self):
        """ Load the model.
        """
        with (self.env.model_folder() / "boston-model.json").open("r") as f:
            data = jsonpickle.decode(f.read())
```

(continues on next page)

(continued from previous page)

```
            self.model = data["model"]
            self.features = data["features"]

    def result(self, data):
        """ Perform a prediction on the given data and return the result.

            :param dict data:
                The data to perform the prediction on.

            :returns dict:
                The result of the prediction.
        """
        X = pd.DataFrame([data])
        X = X[self.features]
        price = self.model.predict(X)[0]
        return {"predicted_price": price}


class BostonModel(Model):

    TRAINER = BostonTrainer
    PREDICTOR = BostonPredictor
```

The training data should be read from *self.env.dataset_folder()*. This is the folder that SageMaker will load your training dataset into.

Once the model is trained, you should write your output files to *self.env.model_folder()*. SageMaker will read the contents of this folder once training has finished and store them as a *.tar.gz* file in S3.

Before you train your model in SageMaker, you can try it locally as shown in *local.py*:

```
# -*- coding: utf-8 -*-

""" Train the Boston house prices model on your local machine.
"""

import pandas as pd
from ml2p.core import LocalEnv
import model


def train(env):
    """ Train and save the model locally. """
    trainer = model.BostonModel().trainer(env)
    trainer.train()


def predict(env):
    """ Load a model and make predictions locally. """
    predictor = model.BostonModel().predictor(env)
    predictor.setup()
    data = pd.read_csv("house-prices.csv")
    house = dict(data.iloc[0])
    del house["target"]
    print("Making a prediction for:")
    print(house)
    result = predictor.invoke(house)
```

(continues on next page)

(continued from previous page)

```
    print("Prediction:")
    print(result)


if __name__ == "__main__":
    env = LocalEnv(".", "ml2p.yml")
    train(env)
    predict(env)
```

ML2P provides *ml2p.core.LocalEnv* which you can use to emulate a real SageMaker environment. SageMaker will read the training data from *input/data/training/* so you will need to place a copy of *house-prices.csv* there for the script to run successfully.

Later in the tutorial you will learn how to download a dataset directly from S3 for use in a local environment.

Once your model works locally, you are ready to train it in SageMaker by creating a training job with:

```
$ ml2p training-job create boston-train boston-20200901 --model-type boston
```

The first argument is the name of the training job, the second is the name of the dataset. You will need to have uploaded some training data. The *–model-type* argument is optional – if you have only a single model defined in *ml2p.yml*, ML2P will automatically select that one for you.

Wait for your training job to finish. To check up on it you can run:

```
$ ml2p training-job wait boston-train  # wait for job to finish
$ ml2p training-job describe boston-train  # inspect job
```

Once your training job is done, there is one more step. The training job records the trained model parameters, but we also need to specify the Docker image that should be used along with those parameters. We do this by creating a SageMaker model from the output of the training job:

```
$ ml2p model create boston-model boston-train --model-type boston
```

The first argument is the name of the model to create, the second is the training job the model should be created from.

The Docker image to use is read from the *image* parameter in *ml2p.yml* so you don't have to specify it here.

The model is just an object in SageMaker – it doesn't run any instances – so it will be created immediately.

Now it's time to deploy your model by creating an endpoint for it!

3.5 Deploying a model

To deploy a model you'll need to implement a subclass of *ml2p.core.ModelPredictor*.

You might have seen the implementation for the Boston house price model in *model.py* while looking at the code for training, but here it is again:

```
# -*- coding: utf-8 -*-

""" A model for predicting Boston house prices (part of the ML2P tutorial).
"""

import jsonpickle

```

(continues on next page)

(continued from previous page)

```
import pandas as pd
from ml2p.core import Model, ModelPredictor, ModelTrainer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split


class BostonTrainer(ModelTrainer):
    def train(self):
        """ Train the model. """
        training_channel = self.env.dataset_folder()
        training_csv = str(training_channel / "house-prices.csv")
        df = pd.read_csv(training_csv)
        y = df["target"]
        X = df.drop(columns="target")
        features = sorted(X.columns)
        X = X[features]
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
        model = LinearRegression().fit(X_train, y_train)
        with (self.env.model_folder() / "boston-model.json").open("w") as f:
            f.write(jsonpickle.encode({"model": model, "features": features}))


class BostonPredictor(ModelPredictor):
    def setup(self):
        """ Load the model.
        """
        with (self.env.model_folder() / "boston-model.json").open("r") as f:
            data = jsonpickle.decode(f.read())
            self.model = data["model"]
            self.features = data["features"]

    def result(self, data):
        """ Perform a prediction on the given data and return the result.

            :param dict data:
                The data to perform the prediction on.

            :returns dict:
                The result of the prediction.
        """
        X = pd.DataFrame([data])
        X = X[self.features]
        price = self.model.predict(X)[0]
        return {"predicted_price": price}


class BostonModel(Model):

    TRAINER = BostonTrainer
    PREDICTOR = BostonPredictor
```

The *.setup()* method is called only once when starting up a prediction instance. It should read the model from *self.env.model_folder()* – SageMaker will have placed them in the same location where they were stored while running *.train()*. Other kinds of setup can be done in this function too if you need to.

The *.result(data)* method is called when a prediction needs to be made. It will be passed the data that was sent to the prediction API endpoint (usually a dictionary with the features as the keys) and should return the prediction.

As you can see in *local.py*, *.result()* is usually not called directly. Instead, when a prediction needs to be made, ML2P will call *.invoke()*, which will then call *.result()* and add some metadata to the result before returning it.

If you ran *local.py* earlier, you've already successfully run a local prediction.

Once you're ready to deploy your model, you can create an endpoint by running:

```
$ ml2p endpoint create boston-endpoint --model-name boston-model
```

The first argument is the name of the endpoint to create, the second is the name of the model to create the endpoint from.

Note that endpoints can be quite expensive to run, so check the pricing for the instance type you have specified before pressing enter!

Setting up the endpoint takes awhile. To check up on it you can run:

```
$ ml2p endpoint wait boston-endpoint  # wait for endpoint to be ready
$ ml2p endpoint describe boston-endpoint  # inspect endpoint
```

Once the endpoint is ready, your model is deployed!

You can make a test prediction using:

```
$ ml2p endpoint invoke boston-endpoint '{"CRIM": 0.00632, "ZN": 18.0, "INDUS": 2.31,
↪"CHAS": 0.0, "NOX": 0.5379999999999999, "RM": 6.575, "AGE": 65.2, "DIS": 4.09, "RAD
↪": 1.0, "TAX": 296.0, "PTRATIO": 15.3, "B": 396.9, "LSTAT": 4.98}'
```

Congratulations! You have trained and deployed your first model using ML2P!

3.6 Security considerations

ML2P runs inside SageMaker, so authentication and authorization of prediction requests is managed using AWS IAM profiles and roles, but there are still some important things to consider:

3.6.1 Roles

The *role* in *ml2p.yml* defines the permissions your training jobs and endpoints will assume while they run. Best practice is to have a role specific to each ML2P project and for that role to have only the permissions it requires.

3.6.2 VPCs

By default, SageMaker instances run outside of any AWS VPC. This means that the instances access other AWS services (e.g. downloading training data or a stored model from S3) via the public internet address of the service (the connection is encrypted and authenticated as usual) and that they have no special access to any other services you might be running inside a VPC.

You can attach your SageMaker instances to a VPC by specifying a *vpc_config*:

```
vpc_config:
  security_groups:
    - "sg-XXXX"
  subnets:
    - "net-YYYY"
```

This will do two things.

Firstly, it will allow your SageMaker instances to access instances within the VPC (according to the subnet and security group rules).

Secondly, it will prevent your SageMaker instances from accessing the public internet (unless allowed to by the security group or subnet rules). This second point means you may have to configure a VPC Endpoint to allow your SageMaker instances to access other AWS services such as S3.

You can read more on how to Give Endpoints Access to Resources in Your VPC and how to Give Training Jobs Access to Resources in Your VPC in the AWS SageMaker documentation.

3.6.3 Prediction web requests

While your SageMaker instance is likely protected from unauthorised access, like any web API care should be taken when handling untrusted data. This includes any fields passed to the model that can be manipulated by an untrusted party (including, for example, emails or other text from customers).

3.7 Working with models locally

At times it may be convenient to work with ML2P models on a local machine, rather than within SageMaker. ML2P supports both training models locally and loading models trained in SageMaker for local analysis.

In either case, first create a local environment:

```
# set up a connection to AWS, specifying an appropriate AWS profile name:
import boto3
session = boto3.session.Session(profile_name="aws-profile")

# create a local environment
from ml2p.core import LocalEnv
env = LocalEnv(".", "./ml2p.yml", session)

# import your ml2p model class:
from model import BostonModel
```

The first argument to *LocalEnv* is the local folder to store the environment in, and the second is the path to the *ml2p.yml* config file for the project.

The third argument, *session*, is an optional boto3 session and is only needed if you wish to download datasets or models from S3 to your local environment.

To download a dataset from S3 into the local environment used:

```
env.download_dataset("dataset-name")
```

If you prefer not to download a dataset, you can also copy a local file into:

```
input/data/training/
```

For example, for this tutorial it may be useful to copy the *house-prices.csv* training file into this folder using:

```
$ mkdir -p input/data/training/
$ cp house-prices.csv input/data/training/
```

Once you have a dataset you can train a model locally using:

```
env.clean_model_folder()
trainer = BostonModel().trainer(env)
trainer.train()
```

The first line, *env.clean_model_folder()* just deletes any old files created by previous local training runs.

You can list the model files created during training using:

```
$ ls model/
```

If you have already trained a model in SageMaker with *ml2p create training-job* and would like to examine it locally you can download it into the model folder by running:

```
env.download_model("training-job-name")
```

Once you have a model available locally, either by training it locally or by downloading it, you can make predictions with:

```
predictor = BostonModel().predictor(env)
predictor.setup()
predictor.invoke(data)
```

Happy local analyzing and debugging!

CHAPTER

FOUR

REFERENCE GUIDE

4.1 ML2P CLI Reference

4.1.1 ml2p

Minimal Lovable Machine Learning Pipeline.

A friendlier interface to AWS SageMaker.

```
ml2p [OPTIONS] COMMAND [ARGS]...
```

Options

--cfg <cfg>
Project configuration file. Default: ./ml2p.yml.

--version
Show the version and exit.

dataset

Create and manage datasets.

```
ml2p dataset [OPTIONS] COMMAND [ARGS]...
```

create

Create a dataset.

```
ml2p dataset create [OPTIONS] DATASET
```

Arguments

DATASET
Required argument

delete

Delete a dataset.

```
ml2p dataset delete [OPTIONS] DATASET
```

Arguments

DATASET
Required argument

dn

Download a file SRC from the dataset and save it in DST.

If DST is omitted, the source file is downloaded as its own name.

```
ml2p dataset dn [OPTIONS] DATASET SRC [DST]
```

Arguments

DATASET
Required argument

SRC
Required argument

DST
Optional argument

list

List datasets for this project.

```
ml2p dataset list [OPTIONS]
```

ls

List the contents of a dataset.

```
ml2p dataset ls [OPTIONS] DATASET
```

Arguments

DATASET
Required argument

rm

Delete a file from a dataset.

```
ml2p dataset rm [OPTIONS] DATASET FILENAME
```

Arguments

DATASET
Required argument

FILENAME
Required argument

up

Upload a file SRC to a dataset as DST.

If DST is omitted, the source file is uploaded under its own name.

```
ml2p dataset up [OPTIONS] DATASET SRC [DST]
```

Arguments

DATASET
Required argument

SRC
Required argument

DST
Optional argument

endpoint

Create and inspect endpoints.

```
ml2p endpoint [OPTIONS] COMMAND [ARGS]...
```

create

Create an endpoint for a model.

```
ml2p endpoint create [OPTIONS] ENDPOINT_NAME
```

Options

-m, --model-name <model_name>
The name of the model to base the endpoint on. Defaults to the endpoint name without the live/analysis/test suffix.

Arguments

ENDPOINT_NAME
Required argument

delete

Delete an endpoint.

```
ml2p endpoint delete [OPTIONS] ENDPOINT_NAME
```

Arguments

ENDPOINT_NAME
Required argument

describe

Describe an endpoint.

```
ml2p endpoint describe [OPTIONS] ENDPOINT_NAME
```

Arguments

ENDPOINT_NAME
Required argument

invoke

Invoke an endpoint (i.e. make a prediction).

```
ml2p endpoint invoke [OPTIONS] ENDPOINT_NAME JSON_DATA
```

Arguments

ENDPOINT_NAME
Required argument

JSON_DATA
Required argument

list

List endpoints for this project.

```
ml2p endpoint list [OPTIONS]
```

wait

Wait for an endpoint to be ready or dead.

```
ml2p endpoint wait [OPTIONS] ENDPOINT_NAME
```

Arguments

ENDPOINT_NAME
Required argument

init

Initialize the project S3 bucket.

```
ml2p init [OPTIONS]
```

model

Create and inspect models.

```
ml2p model [OPTIONS] COMMAND [ARGS]...
```

create

Create a model.

```
ml2p model create [OPTIONS] MODEL_NAME
```

Options

-t, --training-job <training_job>
The name of the training job to base the model on. Defaults to the model name without the patch version number.

-m, --model-type <model_type>
The name of the type of model.

Arguments

MODEL_NAME
Required argument

delete

Delete a model.

```
ml2p model delete [OPTIONS] MODEL_NAME
```

Arguments

MODEL_NAME
Required argument

describe

Describe a model.

```
ml2p model describe [OPTIONS] MODEL_NAME
```

Arguments

MODEL_NAME
Required argument

list

List models for this project.

```
ml2p model list [OPTIONS]
```

notebook

Create and manage notebooks.

```
ml2p notebook [OPTIONS] COMMAND [ARGS]...
```

create

Create a notebook instance.

```
ml2p notebook create [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

delete

Delete a notebook instance.

```
ml2p notebook delete [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

describe

Describe a notebook instance.

```
ml2p notebook describe [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

list

```
ml2p notebook list [OPTIONS]
```

presigned-url

Create a URL to connect to the Jupyter server from a notebook instance.

```
ml2p notebook presigned-url [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

start

Start a notebook instance.

```
ml2p notebook start [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

stop

Stop a notebook instance.

```
ml2p notebook stop [OPTIONS] NOTEBOOK_NAME
```

Arguments

NOTEBOOK_NAME
Required argument

repo

Describe and list code repositories.

```
ml2p repo [OPTIONS] COMMAND [ARGS]...
```

describe

Describe a code repository SageMaker resource.

```
ml2p repo describe [OPTIONS] REPO_NAME
```

Arguments

REPO_NAME
Required argument

list

List code repositories.

```
ml2p repo list [OPTIONS]
```

training-job

Create and inspect training jobs.

```
ml2p training-job [OPTIONS] COMMAND [ARGS]...
```

create

Create a training job.

```
ml2p training-job create [OPTIONS] TRAINING_JOB DATASET
```

Options

-m, --model-type <model_type>
The name of the type of model.

Arguments

TRAINING_JOB
Required argument

DATASET
Required argument

describe

Describe a training job.

```
ml2p training-job describe [OPTIONS] TRAINING_JOB
```

Arguments

TRAINING_JOB
Required argument

list

List training jobs for this project.

```
ml2p training-job list [OPTIONS]
```

wait

Wait for a training job to complete or stop.

```
ml2p training-job wait [OPTIONS] TRAINING_JOB
```

Arguments

TRAINING_JOB
Required argument

4.2 ML2P Docker CLI Reference

4.2.1 ml2p-docker

ML2P Sagemaker Docker container helper CLI.

```
ml2p-docker [OPTIONS] COMMAND [ARGS]...
```

Options

--ml-folder <ml_folder>
The base folder for the datasets and models.

--model <model>
The fully qualified name of the ML2P model interface to use.

--version
Show the version and exit.

serve

Serve the model and make predictions.

```
ml2p-docker serve [OPTIONS]
```

Options

--debug, --no-debug

train

Train the model.

```
ml2p-docker train [OPTIONS]
```

4.3 Library Reference

4.3.1 ML2P Core

ML2P core utilities.

Models

class ml2p.core.**Model**
A holder for a trainer and predictor.

Sub-classes should:

- Set the attribute TRAINER to a ModelTrainer sub-class.
- Set the attribute PREDICTOR to a ModelPredictor sub-class.

class ml2p.core.**ModelTrainer**(*env*)
An interface that allows ml2p-docker to train models within SageMaker.

train()
Train the model.

This method should:

- Read training data (using self.env to determine where to read data from).
- Train the model.
- Write the model out (using self.env to determine where to write the model to).
- Write out any validation or model analysis alongside the model.

class ml2p.core.**ModelPredictor**(*env*)
An interface that allows ml2p-docker to make predictions from a model within SageMaker.

batch_invoke(*data*)
Invokes the model on a batch of input data and returns the full result for each instance.

Parameters **data** (*dict*) – The batch of input data the model is being invoked with.

Return type list

Returns The result as a list of dictionaries.

By default this method results a list of dictionaries containing:

- metadata: The result of calling .metadata().
- result: The result of calling .batch_result(data).

batch_result(*data*)
Make a batch prediction given a batch of input data.

Parameters **data** (*dict*) – The batch of input data to make a prediction from.

Return type list

Returns The list of predictions made for instance of the input data.

This method can be overrided for sub-classes in order to improve performance of batch predictions.

invoke(*data*)
Invokes the model and returns the full result.

Parameters **data** (*dict*) – The input data the model is being invoked with.

Return type dict

Returns The result as a dictionary.

By default this method results a dictionary containing:

- metadata: The result of calling .metadata().
- result: The result of calling .result(data).

metadata()
Return metadata for a prediction that is about to be made.

Return type dict

Returns The metadata as a dictionary.

By default this method returns a dictionary containing:

- model_version: The ML2P_MODEL_VERSION (str).
- timestamp: The UTC POSIX timestamp in seconds (float).

record_invoke(*datum*, *prediction*)
Store an invocation of the endpoint in the ML2P project S3 bucket.

Parameters

- **datum** (*dict*) – The dictionary of input values passed when invoking the endpoint.
- **result** (*dict*) – The prediction returned for datum by this predictor.

record_invoke_id(*datum*, *prediction*)
Return an id for an invocation record.

Parameters

- **datum** (*dict*) – The dictionary of input values passed when invoking the endpoint.
- **result** (*dict*) – The prediction returned for datum by this predictor.

Returns dict Returns an *ordered* dictionary of key-value pairs that make up the unique identifier for the invocation request.

By default this method returns a dictionary containing the following:

- "ts": an ISO8601 formatted UTC timestamp.
- "uuid": a UUID4 unique identifier.

Sub-classes may override this method to return their own identifiers, but including these default identifiers is recommended.

The name of the record in S3 is determined by combining the key value pairs with a dash ("-") and then separating each pair with a double dash ("–").

result(*data*)
Make a prediction given the input data.

Parameters **data** (*dict*) – The input data to make a prediction from.

Return type dict

Returns The prediction result as a dictionary.

setup()
Called once before any calls to .predict(...) are made.

This method should:

- Load the model (using self.env to determine where to read the model from).
- Allocate any other resources needed in order to make predictions.

teardown()
Called once after all calls to .predict(...) have ended.

This method should:

- Cleanup any resources acquired in .setup().

SageMakerEnv

class ml2p.core.**SageMakerEnv**(*ml_folder*, *environ=None*)
An interface to the SageMaker docker environment.

Attributes that are expected to be available in both training and serving environments:

- *env_type* - Whether this is a training, serving or local environment (type: ml2p.core.SageMakerEnvType).

- *project* - The ML2P project name (type: str).
- *model_cls* - The fulled dotted Python name of the ml2p.core.Model class to be used for training and prediction (type: str). This may be None if the docker image itself specifies the name with *ml2p-docker –model ...*.
- *s3* - The URL of the project S3 bucket (type: ml2p.core.S3URL).

Attributes that are only expected to be available while training (and that will be None when serving the model):

- *training_job_name* - The full job name of the training job (type: str).

Attributes that are only expected to be available while serving the model (and that will be None when serving the model):

- *model_version* - The full job name of the deployed model, or None during training (type: str).
- *record_invokes* - Whether to store a record of each invocation of the endpoint in S3 (type: bool).

In the training environment settings are loaded from hyperparameters stored by ML2P when the training job is created.

In the serving environment settings are loaded from environment variables stored by ML2P when the model is created.

class ml2p.core.**SageMakerEnvType**

The type of SageMakerEnvironment.

TRAIN = 'train'

SERVE = 'serve'

LOCAL = 'local'

LocalEnv

class ml2p.core.**LocalEnv**(*ml_folder*, *cfg*, *session=None*)

An interface to a local dummy of the SageMaker environment.

Parameters

- **ml_folder** (*str*) – The directory the environments files are stored in. An error is raised if this directory does not exist. Files and folders are created within this directory as needed.
- **cfg** (*str*) – The path to an ml2p.yml configuration file.
- **session** (*boto3.session.Session*) – A boto3 session object. Maybe be None if downloading files from S3 is not required.

Attributes that are expected to be available in the local environment:

- *env_type* - Whether this is a training, serving or local environment (type: ml2p.core.SageMakerEnvType).
- *project* - The ML2P project name (type: str).
- *s3* - The URL of the project S3 bucket (type: ml2p.core.S3URL).
- *model_version* - The fixed value "local" (type: str).

In the local environment settings are loaded directly from the ML2P configuration file.

clean_model_folder()

Remove and recreate the model folder.

This is useful to run before training a model if one wants to ensure that the model folder is empty beforehand.

download_dataset(*dataset*)
Download the given dataset from S3 into the local environment.

Parameters **dataset** (*str*) – The name of the dataset in S3 to download.

download_model(*training_job*)
Download the given trained model from S3 and unpack it into the local environment.

Parameters **training_job** (*str*) – The name of the training job whose model should be downloaded.

S3URL

class ml2p.core.**S3URL**(*s3folder*)
A friendly interface to an S3 URL.

bucket()
Return the bucket of the S3 URL.

Return type str

Returns The bucket of the S3 URL.

path(*suffix*)
Return the base path of the S3 URL followed by a '/' and the given suffix.

Parameters **suffix** (*str*) – The suffix to append.

Return type str

Returns The path with the suffix appended.

url(*suffix=''*)
Return S3 URL followed by a '/' and the given suffix.

Parameters **suffix** (*str*) – The suffix to append. Default: "".

Return type str

Returns The URL with the suffix appended.

CHAPTER

FIVE

HISTORY

5.1 0.2.0 (2020-09-16)

- Added reference documentation.
- Added a tutorial.
- Added tests for the ML2P command line utilities.
- Added support for attaching training and deployment instances to VPCs.
- Open sourced ML2P under the ISCL.

5.2 0.1.5 (2020-06-12)

- Added *ml2p dataset delete* which deletes an entire dataset.
- Added *ml2p dataset ls* which lists the contents of a dataset.
- Added *ml2p dataset up* which uploads a local file to a dataset.
- Added *ml2p dataset dn* which downloads a file from a dataset.
- Added *ml2p dataset rm* which deletes a file from a dataset.

5.3 0.1.4 (2020-02-21)

- Correctly handle folder keys when downloading datasets from S3. Previously folder keys created files, now they created folders.

5.4 0.1.3 (2020-02-20)

- Added support for local environments. These allow ML2P models to be trained and used to make predictions locally, as though they were being loaded in SageMaker.
- Added support for downloading datasets and models from S3 into local environments.

5.5 0.1.2 (2020-01-23)

- Fix support for recording predictions in S3 (in first release of this feature, the code attempted to pass a boolean value as an environment variable, which failed as expected).

5.6 0.1.1 (2020-01-22)

- Add support for recording predictions in S3.

5.7 0.1.0 (2019-10-22)

- Improve batch prediction support to allow models to separately implement batch prediction (e.g. a model might want to implement batch prediction separately to improve performance).
- Tweak training job version format to only include major and minor versions numbers. Patch version numbers are now reserved for models and intended for use in the case where the code used to make predictions changes but the underlying model is the same.
- Model creation now defaults to using the training job with the same version as the model but with the patch number removed.
- Endpoint creation now defaults to using the model with the same version as the endpoint.
- When creating training jobs or models, specifying the model type is now required if the ml2p configuration file contains more than one model. If there is exactly one model type listed, that is the default. If there are no model types, the docker file must specify the model on the command line.
- Metadata returned by predictions now includes the ML2P version number.
- Version bumped to 0.1.0 now that versioning support is complete(-ish).

5.8 0.0.9 (2019-10-15)

- Add support for client and server error exception handling.
- Deprecate passing a channel name to dataset_folder and add a new data_channel_folder method to allow data in other channels to be accessed.
- Add dataset create and list commands to ml2p CLI.
- Add –version to ml2p and ml2p-docker CLIs.
- Allow model and endpoint version numbers to be multiple digits.

5.9 0.0.8 (2019-09-11)

- Added validation of naming convention

5.10 0.0.7 (2019-08-29)

- Added Sphinx requirements to build file.

5.11 0.0.6 (2019-08-29)

- Cleaned up support for passing ML2P environment data into training jobs and model deployments. Environment settings such as the S3 URL and the project name are now passed into training jobs via hyperparameters and into model deployments via model environment variables.
- Added support for training and serving multiple models using the same docker image by optionally passing the model to use into training jobs and endpoint deployments.
- Added support for rich hyperparameters. This sidesteps SageMaker API's limited hyperparameter support (it only supports string values) by encoding any JSON-compatible Python dictionary to a flattened formed and then decoding it when it is read by the training job.
- Added skeleton for Sphinx documentation.
- Removed old pre-0.0.1 example files.

5.12 0.0.5 (2019-07-23)

- Disabled direct internet access from notebooks by default.
- Added tests for cli_utils.

5.13 0.0.4 (2019-06-26)

- Fixed bug in setting of ML2P_S3_URL on model creation.

5.14 0.0.3 (2019-06-26)

- Added new ml2p notebook command group for creating, inspecting, and deleting SageMaker Notebook instances.
- Added new ml2p repo command group for inspecting code repository SageMaker resources.

5.15 0.0.2 (2019-05-24)

- Complete re-write.
- Added new ml2p-docker command added that assists with training and deploying models in SageMaker.

5.16 0.0.1 (2018-10-19)

- Initial hackathon release.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

B

C

D

E

N

P

R

S

T

U