

Machine Learning 1.5.1

Experiments

Date published: 2020-07-16

Date modified: 2023-01-31

CLOUDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Experiments with MLflow.....	4
CML Experiment Tracking through MLflow API.....	4
Running an Experiment using MLflow.....	5
Visualizing Experiment Results.....	7
Using an MLflow Model Artifact in a Model REST API.....	8
Deploying an MLflow model as a CML Model REST API.....	10
Automatic Logging.....	13
Setting Permissions for an Experiment.....	13
Known issues and limitations.....	13
Running an Experiment (Legacy).....	13
Limitations.....	17
Tracking Metrics.....	18
Saving Files.....	18
Debugging Issues with Experiments.....	19

Experiments with MLflow

Machine Learning requires experimenting with a wide range of datasets, data preparation steps, and algorithms to build a model that maximizes a target metric. Once you have built a model, you also need to deploy it to a production system, monitor its performance, and continuously retrain it on new data and compare it with alternative models.



Note: This section describes the newer version of the Experiments feature. For information on the legacy Experiments feature, which is now deprecated, see *Experiments (Legacy)*

CML lets you train, reuse, and deploy models with any library, and package them into reproducible artifacts that other data scientists can use.

CML packages the ML models in a reusable, reproducible form so you can share it with other data scientists or transfer it to production.

CML is compatible with the MLflow™ tracking API and makes use of the MLflow client library as the default method to log experiments. Existing projects with existing experiments are still available and usable.

The functionality described in this document is for the new version of the Experiments feature, which replaces an older version of the Experiments feature that could not be used from within Sessions. In Projects that have existing Experiments created using the previous feature, you can continue to view these existing Experiments. New projects use the new Experiments feature.

Related Information

[Running an Experiment \(Legacy\)](#)

CML Experiment Tracking through MLflow API

CML's experiment tracking features allow you to use the MLflow client library for logging parameters, code versions, metrics, and output files when running your machine learning code. The MLflow library is available in CML Sessions without you having to install it. CML also provides a UI for later visualizing the results. MLflow tracking lets you log and query experiments using the following logging functions:



Note: CML currently supports only Python for experiment tracking.

- `mlflow.create_experiment()` creates a new experiment and returns its ID. Runs can be launched under the experiment by passing the experiment ID to `mlflow.start_run`.

Cloudera recommends that you create an experiment to organize your runs. You can also create experiments using the UI.

- `mlflow.set_experiment()` sets an experiment as active. If the experiment does not exist, `mlflow.set_experiment` creates a new experiment. If you do not wish to use the `set_experiment` method, a default experiment is selected.

Cloudera recommends that you set the experiment using `mlflow.set_experiment`.

- `mlflow.start_run()` returns the currently active run (if one exists), or starts a new run and returns a `mlflow.ActiveRun` object usable as a context manager for the current run. You do not need to call `start_run` explicitly; calling one of the logging functions with no active run automatically starts a new one.
- `mlflow.end_run()` ends the currently active run, if any, taking an optional run status.

- `mlflow.active_run()` returns a `mlflow.entities.Run` object corresponding to the currently active run, if any.



Note: You cannot access currently-active run attributes (parameters, metrics, etc.) through the run returned by `mlflow.active_run`. In order to access such attributes, use the `mlflow.tracking.MlflowClient` as follows:

```
client = mlflow.tracking.MlflowClient()
data = client.get_run(mlflow.active_run().info.run_id).data
```

- `mlflow.log_param()` logs a single key-value parameter in the currently active run. The key and value are both strings. Use `mlflow.log_params()` to log multiple parameters at once.
- `mlflow.log_metric()` logs a single key-value metric for the current run. The value must always be a number. MLflow remembers the history of values for each metric. Use `mlflow.log_metrics()` to log multiple metrics at once.

Parameters:

- `key` - Metric name (string)
- `value` - Metric value (float). Note that some special values such as +/- Infinity may be replaced by other values depending on the store. For example, the SQLAlchemy store replaces +/- Infinity with max / min float values.
- `step` - Metric step (int). Defaults to zero if unspecified.

Syntax - `mlflow.log_metrics(metrics: Dict[str, float], step: Optional[int] = None) # None`

- `mlflow.set_tag()` sets a single key-value tag in the currently active run. The key and value are both strings. Use `mlflow.set_tags()` to set multiple tags at once.
- `mlflow.log_artifact()` logs a local file or directory as an artifact, optionally taking an `artifact_path` to place it within the run's artifact URI. Run artifacts can be organized into directories, so you can place the artifact in a directory this way.
- `mlflow.log_artifacts()` logs all the files in a given directory as artifacts, again taking an optional `artifact_path`.
- `mlflow.get_artifact_uri()` returns the URI that artifacts from the current run should be logged to.

For more information on MLflow API commands used for tracking, see [MLflow Tracking](#).

Running an Experiment using MLflow

This topic walks you through a simple example to help you get started with Experiments in Cloudera Machine Learning.

Best practice: It's useful to display two windows while creating runs for your experiments: one window displays the Experiments tab and another displays the MLflow Session.

1. From your Project window, click New Experiment and create a new experiment. Keep this window open to return to after you run your new session.
2. From your Project window, click New Session.
3. Create a new session using ML Runtimes. Experiment runs cannot be created from sessions using Legacy Engine.
4. In your Session window, import MLflow by running the following code: `import mlflow` The ML Flow client library is installed by default, but you must import it for each session.
5. Start a run and then specify the MLflow parameters, metrics, models and artifacts to be logged. You can enter the code in the command prompt or create a project. See *CML Experiment Tracking through MLflow API* for a list of functions you can use.

For example:

```
mlflow.set_experiment(<experiment_name>)
mlflow.start_run()
mlflow.log_param("input", 5)
mlflow.log_metric("score", 100)
with open("data/features.txt", 'w') as f:
```

```
f.write(features)
# Writes all files in "data" to root artifact_uri/states
mlflow.log_artifacts("data", artifact_path="states")
## Artifacts are stored in project directory under
/home/cdsw/.experiments/<experiment_id>/<run_id>/artifacts
mlflow.end_run()<
```

The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The code in the notebook is as follows:

```
27 # Write the features to a file
28 f.write(features)
29
30 # Writes all files in "data" to root artifact_uri/states
31 mlflow.log_artifacts("data", artifact_path="states")
32
33 ## Artifacts are stored in project directory under
34 #/home/cdsw/.experiments/<experiment_id>/<run_id>/artifacts
35 mlflow.end_run()<
```

The execution output on the right shows the MLflow session details and the results of the code execution:

```
MLflow test session2 [Running]
By CDEP-CREATED-ACCOUNT - Session - 1 vCPU / 2 GB Memory - 10 minutes ago

Session Logs
Collapse Show Export PDF

This.x = train.drop(['quality'], axis=1)
test.x = test.drop(['quality'], axis=1)
train.y = train[['quality']]
test.y = test[['quality']]
alpha = float(sys.argv[1]) if len(sys.argv) > 1 else 0.5
l1_ratio = float(sys.argv[2]) if len(sys.argv) > 2 else 0.5
with mlflow.start_run():
    lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
    lr.fit(train.x, train.y)

    predicted_qualities = lr.predict(test.x)

    (rmse, mse, r2) = eval_metrics(test.y, predicted_qualities)
    print('Elasticnet model (alpha=%f, l1_ratio=%f):' % (alpha, l1_ratio))
    print(' RMSE: %a' % rmse)
    print(' MSE: %a' % mse)
    print(' R2: %a' % r2)
    mlflow.log_param('alpha', alpha)
    mlflow.log_param('l1_ratio', l1_ratio)
    mlflow.log_metric('rmse', rmse)
    mlflow.log_metric('r2', r2)
    mlflow.log_metric('mse', mse)
    mlflow.sklearn.log_model(lr, 'model')

Elasticnet model (alpha=0.500000, l1_ratio=0.500000):
RMSE: 0.35857127928567
MSE: 0.1285336274815638
R2: 0.12217381912864190
```

For information on using editors, see [Using Editors for ML Runtimes](#).

- Continue creating runs and tracking parameters, metrics, models, and artifacts as needed.
- To view your run information, display the Experiments window and select your experiment name. CML displays the Runs table.

The screenshot shows the MLflow Experiments window in a web browser. The page title is "admin / MLflow project / Experiments / demo". The experiment name is "demo" and the experiment ID is "51y4e14c4b3qk4kf". The artifact location is "/home/cdsw/.experiments/51y4e14c4b3qk4kf".

The "Runs (2)" section shows a table of runs with the following columns: Status, Start Time, Run Name, User, Source, Version, Parameters, Metrics, and more. The table contains two runs:

Status	Start Time	Run Name	User	Source	Version	Parameters	Metrics	more
Success	2021-10-23 06:...	uap-d4d0-...	admin	ipython	-	alpha: 0.5, l1_ratio: 0.5	rmse: 0.35857127928567, mse: 0.1285336274815638, r2: 0.12217381912864190	...
Success	2021-10-23 06:...	u8v-hy4s-...	admin	ipython	-	alpha: 0.5, l1_ratio: 0.5	rmse: 0.35857127928567, mse: 0.1285336274815638, r2: 0.12217381912864190	...

- Click the Refresh button on the Experiments window to display recently created runs
- You can customize the Run table by clicking Columns, and selecting the columns you want to display.

Related Information

[Using Editors for ML Runtimes](#)

Visualizing Experiment Results

After you create multiple runs, you can compare your results.

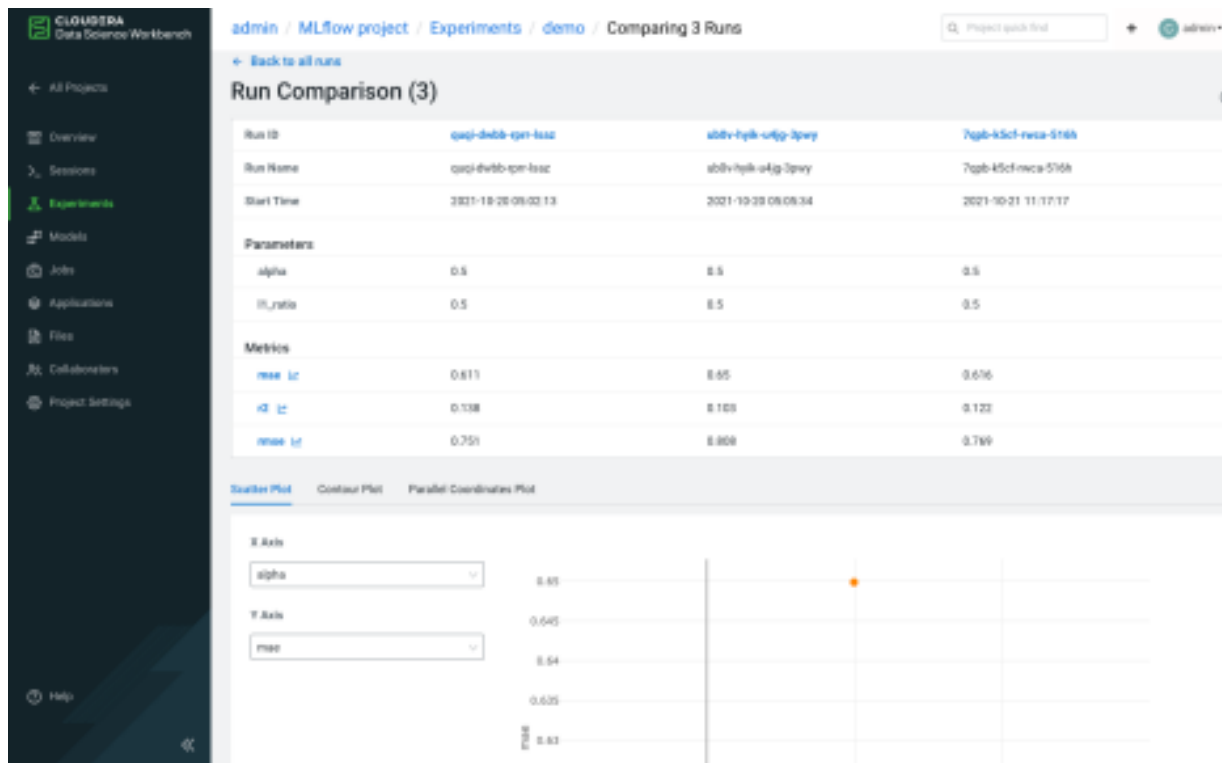
1. Go to Experiments and click on your experiment name. CML displays the Runs table populated by all of the runs for the experiment.

The screenshot shows the MLflow interface for an experiment named 'demo'. The 'Runs (3)' table is displayed with the following data:

	Status	Start Time	Run Name	User	Source	Version	Parameters	Metrics
<input type="checkbox"/>	Success	2021-10-28 06:...	app-14816...	admin	app14816...	-	alpha: 0.5, f1: 0.5	rse: 0.81981488...
<input type="checkbox"/>	Success	2021-10-28 06:...	ab0v-hyke...	admin	app14816...	-	alpha: 0.5, f1: 0.5	rse: 0.84885704...
<input type="checkbox"/>	Success	2021-10-27 11:...	7qpk-4816...	admin	app14816...	-	alpha: 0.5, f1: 0.5	rse: 0.81883302...

2. You can search your run information by using the search field at the top of the Run table.
3. You can customize the Run table by clicking Columns, and selecting the columns you want to display.
4. You can display details for a specific run by clicking the start time for the run in the Run table. You can add notes for the run by clicking the Notes icon. You can display the run metrics in a chart format by clicking the specific metric under Metrics.
5. To compare the data from multiple runs, use the checkbox in the Run table to select the runs you want to compare. You can use the top checkbox to select all runs in the table. Alternatively, you can select runs using the spacebar and arrow keys.

- Click Compare. Alternatively, you can press Cmd/Ctrl + Enter. CML displays a separate window containing a table titled Run Comparison and options for comparing your parameters and metrics.



This Run Comparison table lists all of the parameters and the most recent metric information from the runs you selected. Parameters that have changed are highlighted

- You can graphically display the Run metric data by clicking the metric names in the Metrics section. If you have a single value for your metrics, it will display as a bar chart. If your run has multiple values, the metrics comparison page displays the information with multiple steps, for example, over time. You can choose how the data is displayed:
 - Time (Relative): graphs the time relative to the first metric logged, for each run.
 - Time (Wall): graphs the absolute time each metric was logged.
 - Step: graphs the values based on the cardinal order.
- Below the Run Comparison table, you can choose how the Run information is displayed:
 - Scatter Plot: Use the scatter plot to see patterns, outliers, and anomalies.
 - Contour Plot: Contour plots can only be rendered when comparing a group of runs with three or more unique metrics or parameters. Log more metrics or parameters to your runs to visualize them using the contour plot.
 - Parallel Coordinates Plot: Choose the parameters and metrics you want displayed in the plot.

Using an MLflow Model Artifact in a Model REST API

You can use MLflow to create, deploy, and manage models as REST APIs to serve predictions

- To create an MLflow model add the following information when you run an experiment:

```
mlflow.log_artifacts("output")
mlflow.sklearn.log_model(lr, "model")
```

For example:

```
import os
```



```

import warnings
import sys
import mlflow
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
import mlflow.sklearn

import logging

logging.basicConfig(level=logging.WARN)
logger = logging.getLogger(__name__)

def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    r2 = r2_score(actual, pred)
    return rmse, mae, r2

if __name__ == "__main__":
    mlflow.set_experiment("wine-quality-test")
    csv_url = (
        "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
    )
    try:
        data = pd.read_csv(csv_url, sep=";")
    except Exception as e:
        logger.exception(
            "Unable to download training & test CSV, check your internet connection. Error: %s", e
        )

    # Split the data into training and test sets. (0.75, 0.25)
    split.train, test = train_test_split(data)
    # The predicted column is "quality" which is a scalar from [3, 9]
    train_x = train.drop(["quality"], axis=1)
    test_x = test.drop(["quality"], axis=1)
    train_y = train[["quality"]]
    test_y = test[["quality"]]
    alpha = float(sys.argv[1]) if len(sys.argv) > 1 else 0.5
    l1_ratio = float(sys.argv[2]) if len(sys.argv) > 2 else 0.5
    with mlflow.start_run():
        lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio,
            random_state=42)
        lr.fit(train_x, train_y)
        predicted_qualities = lr.predict(test_x)
        (rmse, mae, r2) = eval_metrics(test_y,
            predicted_qualities)
        print("Elasticnet model (alpha=%f, l1_ratio=%f):" %
            (alpha, l1_ratio))
        print(" RMSE: %s" % rmse)
        print(" MAE: %s" % mae)
        print(" R2: %s" % r2)
        mlflow.log_param("alpha", alpha)
        mlflow.log_param("l1_ratio", l1_ratio)
        mlflow.log_metric("rmse", rmse)
        mlflow.log_metric("r2", r2)
        mlflow.log_metric("mae", mae)

```

```
mlflow.sklearn.log_model(lr, "model")
```

In this example we are training a machine learning model using linear regression to predict wine quality. This script creates the MLflow model artifact and logs it to the model directory: `/home/cdsw/.experiments/<experiment_id>/<run_id>/artifacts/models`

2. To view the model, navigate to the Experiments page and select your experiment name. CML displays the Runs page and lists all of your current runs.
3. Click the run from step 1 that created the MLflow model. CML displays the Runs detail page
4. Click Artifacts to display a list of all the logged artifacts for the run.

The screenshot shows the MLflow interface. At the top, there is an 'Add Tags' section with input fields for 'Enter Key' and 'Enter Value', and an 'Add' button. Below this is the 'Artifacts' section, which is expanded to show a list of artifacts under the 'model' directory: 'requirements.txt', 'conda.yaml', 'MLmodel', and 'model.pkl'. To the right of the artifact list is a 'Make Predictions' section. It contains two code snippets for making predictions. The first is for a Spark DataFrame, showing code to load the model as a Spark UDF and predict on a DataFrame. The second is for a Pandas DataFrame, showing code to load the model as a PyFuncModel and predict on a Pandas DataFrame. Both code snippets have a 'Copy Code' button next to them.

5. Click model. CML displays the MLflow information you use to create predictions for your experiment.

Deploying an MLflow model as a CML Model REST API

In the future, you will be able to register models to a Model Registry and then deploy Model REST APIs with those models. Today, these models can be deployed using the following manual process instead

1. Navigate to your project. Note that models are always created within the context of a project.
2. Click Open Workbench and launch a new Python 3 session.
3. Create a new file within the project if one does not already exist: `cdsw-build.sh`. This file defines the function that will be called when the model is run and will contain the MLflow prediction information.
4. Add the following information to the `cdsw-build.sh` file: `pip3 install sklearn mlflow pandas`
5. For non-Python template projects and old projects check the following.
 - a. Check to make sure you have a `.gitignore` file. If you do not have the file, add it.
 - b. Add the following information to the `.gitignore` file: `!.experiments`

For new projects using a Python template, this is already present.

6. Create a Python file to call your model artifact using a Python function. For example:
 - Filename: mlpredict.py
 - Function: predict
7. Copy the MLflow model file path from the Make Predictions pane in the Artifacts section of the Experiments/Run details page and load it in the Python file. This creates a Python function which accepts a dictionary of the input variables and converts these to a Pandas data frame, and returns the model prediction. For example:

```
import mlflow
import pandas as pd
logged_model =
    '/home/cdsw/.experiments/7qwz-1620-d7v6-1922/glma-oqxb-szc7-c8hf/a
rtifacts/model'
def predict(args):
    # Load model as a PyFuncModel.
    data = args.get('input')
    loaded_model = mlflow.pyfunc.load_model(logged_model)
    # Predict on a Pandas DataFrame.
    return loaded_model.predict(pd.DataFrame(data))
```



Note: In practice, do not assume that users calling the model will provide input in the correct format or enter good values. Always perform input validation.

8. Deploy the predict function to a REST endpoint.
 - a. Go to the project Overview page
 - b. Click Models New Model .
 - c. Give the model a Name and Description
 - d. Enter details about the model that you want to build. In this case:
 - File: mlpredict.py
 - Function: predict
 - Example Input:

```
{
  "input": [
    [7.4, 0.7, 0, 1.9, 0.076, 11, 34, 0.9978,
     3.51, 0.56, 9.4]
  ]
}
```

- Example output:

```
[
  5.575822297312952
]
```

]

File *
mlpredict.py

Function *
predict

Example Input ⓘ
{"input": [[7.4, 0.7, 0, 1.9, 0.076, 11, 34, 0.9978, 3.51, 0.56, 9.4]]}

Example Output ⓘ
[
 5.575822297312952
]

Resources

- e. Select the resources needed to run this model, including any replicas for load balancing.



Note: The list of options here is specific to the default engine you have specified in your Project Settings: ML Runtimes or Legacy Engines. Engines allow kernel selection, while ML Runtimes allow Editor, Kernel, Variant, and Version selection. Resource Profile list is applicable for both ML Runtimes and Legacy Engines.

- f. Click Deploy Model.
9. Click on the model to go to its Overview page.
10. Click Builds to track realtime progress as the model is built and deployed. This process essentially creates a Docker container where the model will live and serve requests.

Add Two Numbers Building Stop Deploy New Build

[Overview](#) [Deployments](#) [Builds](#) [Monitoring](#) [Settings](#)

Build	Status	File	Function	Kernel	Engine Image	Created By	Created At	Comment	Actions
1	Building	add_numbers.py	add	python3	Base Image v*	ambreen	Jun 5, 2018, 5:44 PM	Initial revision.	Delete

Sending build context to Docker daemon 15.05 MB

```
Step 1/16 : FROM docker.repository.cloudera.com/cdsw/engine:~
--> f8955770daa1
Step 2/16 : ENTRYPOINT node /app/model-runtime/model-server.js
--> Running in 58038f1e58d5
```

11. Once the model has been deployed, go back to the model Overview page and use the Test Model widget to make sure the model works as expected. If you entered example input when creating the model, the Input field will be pre-populated with those values.

12. Click Test. The result returned includes the output response from the model, as well as the ID of the replica that served the request.

Model response times depend largely on your model code. That is, how long it takes the model function to perform the computation needed to return a prediction. It is worth noting that model replicas can only process one request at a time. Concurrent requests will be queued until the model can process them.

Automatic Logging

Automatic logging allows you to log metrics, parameters, and models without the need for an explicit log statement.

You can perform autologging two ways:

1. Call `mlflow.autolog()` before your training code. This will enable autologging for each supported library you have installed as soon as you import it.
2. Use library-specific autolog calls for each library you use in your code. See below for examples.

For more information about the libraries supported by autologging, see [Automatic Logging](#).

Setting Permissions for an Experiment

Experiments are associated with the project ID, so permissions are inherited from the project. If you want to allow a colleague to view the experiments of a project, you should give them Viewer (or higher) access to the project.

Known issues and limitations

CML has the following known issues and limitations with experiments and MLflow.

- CML currently supports only Python for experiment tracking.
- Experiment runs cannot be created from MLFlow on sessions using Legacy Engine. Instead, create a session using an ML Runtime.
- The version column in the runs table is empty for every run. In a future release, this will show a git commit sha for projects using git.
- There is currently no mechanism for registering a model to a Model Registry. In a future release, you will be able to register models to a Model Registry and then deploy Model REST APIs with those models.
- Browsing an empty experiment will display a spinner that doesn't go away.
- Running an experiment from the workbench (from the dropdown menu) refers to legacy experiments and should not be used going forward.
- Tag/Metrics/Parameter columns that were previously hidden on the runs table will be remembered, but CML won't remember hiding any of the other columns (date, version, user, etc.)
- Admins can not browse all experiments. They can only see their experiments on the global Experiment page.
- Performance issues may arise when browsing the run details of a run with a lot of metric results, or when comparing a lot of runs.
- Runs can not be deleted or archived.

Running an Experiment (Legacy)

This topic walks you through a simple example to help you get started with experiments in Cloudera Machine Learning.



Note: This page applies to the legacy version of Experiments, which is now deprecated.

The following steps describe how to launch an experiment from the Workbench console. In this example we are going to run a simple script that adds all the numbers passed as arguments to the experiment.

1. Go to the project Overview page.
2. Click Open Workbench.
3. Create/modify any project code as needed. You can also launch a session to simultaneously test code changes on the interactive console as you launch new experiments.

As an example, you can run this Python script that accepts a series of numbers as command-line arguments and prints their sum.

add.py

```
import sys
import cdsw

args = len(sys.argv) - 1
sum = 0
x = 1

while (args >= x):
    print ("Argument %i: %s" % (x, sys.argv[x]))
    sum = sum + int(sys.argv[x])
    x = x + 1

print ("Sum of the numbers is: %i." % sum)
```

To test the script, launch a Python session and run the following command from the workbench command prompt:

```
!python add.py 1 2 3 4
```

4. Click Run Experiment. If you're already in an active session, click **Run** **Run Experiment** . Fill out the following fields:

- Script - Select the file that will be executed for this experiment.
- Arguments - If your script requires any command line arguments, enter them here.



Note: Arguments are not supported with Scala experiments.

- Engine Kernel and Resource Profile - Select the kernel and computing resources needed for this experiment.

For this example we will run the add.py script and pass some numbers as arguments.

Run New Experiment ✕

Script

Arguments ⓘ

Runtime

Editor ⓘ **Kernel** ⓘ

Edition ⓘ **Version**

Runtime Image
- docker-registry.infra.cloudera.com/cdsw/runtime-python-quickstart:2020.08.1-b0

Resource Profile

Comment

Cancel Start Run

5. Click Start Run.

- To track progress for the run, go back to the project Overview. On the left navigation bar click Experiments. You should see the experiment you've just run at the top of the list. Click on the Run ID to view an overview for each individual run. Then click Build.

On this Build tab you can see realtime progress as Cloudera Machine Learning builds the Docker image for this experiment. This allows you to debug any errors that might occur during the build stage.

```

Run-4 New Experiment
Overview Session Build
Sending build context to Docker daemon 14.34 MB

Step 1/5 : FROM docker.repository.cloudera.com/cdsw/engine:5
----> da62f78351e0
Step 2/5 : COPY sources /home/cdsw
----> dc0d62362524
Removing intermediate container 52edb8ecef1f
Step 3/5 : WORKDIR /home/cdsw
----> 604125f1bd47
Removing intermediate container 934e47683660
Step 4/5 : RUN chown -R 8536:8536 /home/cdsw
----> Running in 80097e84eb11

```

- Once the Docker image is ready, the run will begin execution. You can track progress for this stage by going to the Session tab.

For example, the Session pane output from running `add.py` is:

```

Run-4 New Experiment
Overview Session Build
> import sys
> import cdsw
> args = len(sys.argv) - 1
> sum = 0
> x = 1
> while (args >= x):
    print ("Parameter %i: %s" % (x, sys.argv[x]))
    sum = sum + int(sys.argv[x])
    x = x + 1
Parameter 1: 18
Parameter 2: 90
Parameter 3: 34
> print ("Sum of the numbers is: %i." % sum)
Sum of the numbers is: 142.

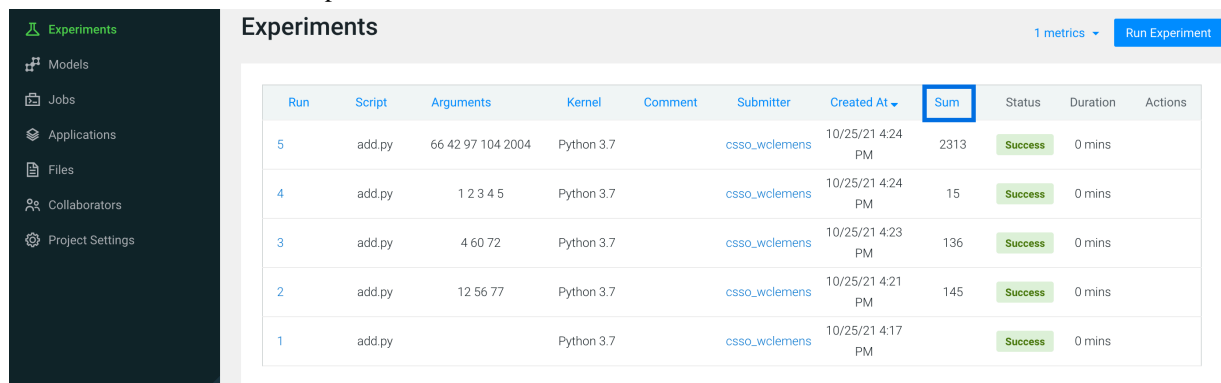
```


8. (Optional) The `cdsw` library that is bundled with Cloudera Machine Learning includes some built-in functions that you can use to compare experiments and save any files from your experiments.

For example, to track the sum for each run, add the following line to the end of the `add.py` script.

```
cdsw.track_metric("Sum", sum)
```

This will be tracked in the Experiments table:



Run	Script	Arguments	Kernel	Comment	Submitter	Created At	Sum	Status	Duration	Actions
5	add.py	66 42 97 104 2004	Python 3.7		csso_wclmens	10/25/21 4:24 PM	2313	Success	0 mins	
4	add.py	1 2 3 4 5	Python 3.7		csso_wclmens	10/25/21 4:24 PM	15	Success	0 mins	
3	add.py	4 60 72	Python 3.7		csso_wclmens	10/25/21 4:23 PM	136	Success	0 mins	
2	add.py	12 56 77	Python 3.7		csso_wclmens	10/25/21 4:21 PM	145	Success	0 mins	
1	add.py		Python 3.7		csso_wclmens	10/25/21 4:17 PM		Success	0 mins	

Related Information

[Tracking Metrics](#)

[Saving Files](#)

Limitations

This topic lists some of the known issues and limitations associated with experiments.



Note: This page applies to the legacy version of Experiments, which is now deprecated.

- Experiments do not store snapshots of project files. You cannot automatically restore code that was run as part of an experiment.
- Experiments will fail if your project filesystem is too large for the Git snapshot process. As a general rule, any project files (code, generated model artifacts, dependencies, etc.) larger than 50 MB must be part of your project's `.gitignore` file so that they are not included in snapshots for experiment builds.
- Experiments cannot be deleted. As a result, be conscious of how you use the `track_metrics` and `track_file` functions.
 - Do not track files larger than 50MB.
 - Do not track more than 100 metrics per experiment. Excessive metric calls from an experiment may cause Cloudera Machine Learning to stop responding.
- The Experiments table will allow you to display only three metrics at a time. You can select which metrics are displayed from the metrics dropdown. If you are tracking a large number of metrics (100 or more), you might notice some performance lag in the UI.
- Arguments are not supported with Scala experiments.
- The `track_metrics` and `track_file` functions are not supported with Scala experiments.
- The UI does not display a confirmation when you start an experiment or any alerts when experiments fail.

Related Information

[Engines for Experiments and Models](#)

Tracking Metrics

This topic teaches you how to use the `track_metric` function to log metrics associated with experiments.



Note: This page applies to the legacy version of Experiments, which is now deprecated.

The `cdsw` library includes a `track_metric` function that can be used to log up to 50 metrics associated with a run, thus allowing accuracy and scores to be tracked over time.

The function accepts input in the form of key value pairs.

```
cdsw.track_metric(key, value)
```

Python

```
cdsw.track_metric("R_squared", 0.79)
```

R

```
cdsw::track.metric("R_squared", 0.62)
```

These metrics will be available on the project's Experiments tab where you can view, sort, and filter experiments on the values. The table on the Experiments page will allow you to display only three metrics at a time. You can select which metrics are displayed from the metrics dropdown.



Note: This function is not supported with Scala experiments.

Saving Files

This topic teaches you how to use the `track_file` function to save files associated with experiments.



Note: This page applies to the legacy version of Experiments, which is now deprecated.

Cloudera Machine Learning allows you to select which artifacts you'd like to access and evaluate after an experiment is complete. These artifacts could be anything from a text file to an image or a model that you have built through the run.

The `cdsw` library includes a `track_file` function that can be used to specify which artifacts should be retained after the experiment is complete.

Python

```
cdsw.track_file('model.pkl')
```

R

```
cdsw::track.file('model.pkl')
```

Specified artifacts can be accessed from the run's Overview page. These files can also be saved to the top-level project filesystem and downloaded from there.



Note: This function is not supported with Scala experiments.

Debugging Issues with Experiments

This topic lists some common issues to watch out for during an experiment's build and execution process.



Note: This page applies to the legacy version of Experiments, which is now deprecated.

Experiment spends too long in Scheduling/Built stage

If your experiments are spending too long in any particular stage, check the resource consumption statistics for the cluster. When the cluster starts to run out of resources, often experiments (and other entities like jobs, models) will spend too long in the queue before they can be executed.

Resource consumption by experiments (and jobs, sessions) can be tracked by site administrators on the Admin Activity page.

Experiment fails in the Build stage

During the build stage Cloudera Machine Learning creates a new Docker image for the experiment. You can track progress for this stage on each experiment's Build page. The build logs on this page should help point you in the right direction.

Common issues that might cause failures at this stage include:

- Lack of execute permissions on the build script itself.
- Inability to reach the Python package index or R mirror when installing packages.
- Typo in the name of the build script (`cdsw-build.sh`). Note that the build process will only run a script called `cdsw-build.sh`; not any other bash scripts from your project.
- Using `pip3` to install packages in `cdsw-build.sh`, but selecting a Python 2 kernel when you actually launch the experiment. Or vice versa.

Experiment fails in the Execute stage

Each experiment includes a Session page where you can track the output of the experiment as it executes. This is similar to the output you would see if you test the experiment in the workbench console. Any runtime errors will display on the Session page just as they would in an interactive session.

Related Information

[Engines for Experiments and Models](#)