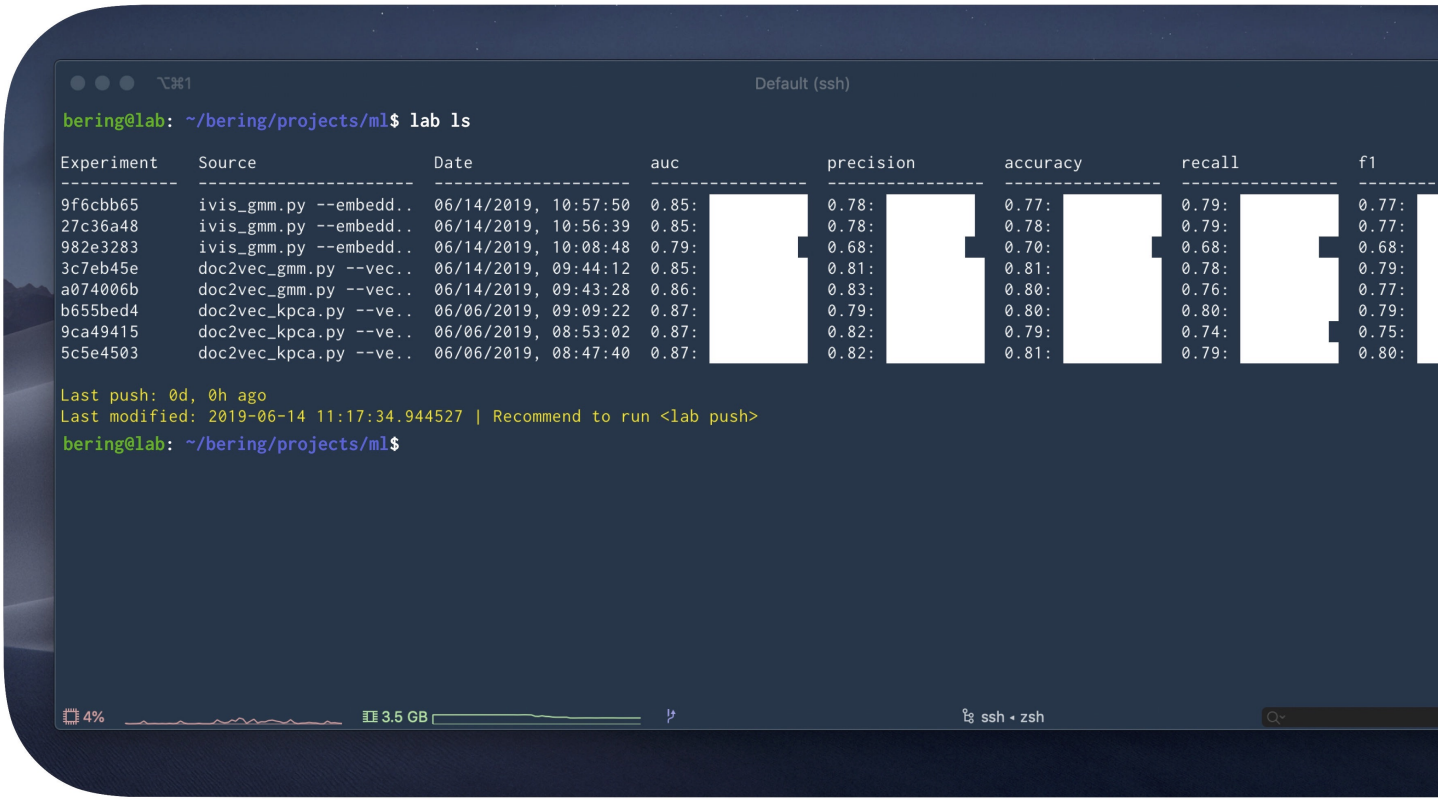
lab

Jan 29, 2021

User Guide:



Lab is an open source platform for managing machine learning pipelines. It addresses three core concepts: **Reproducibility**, **Logging**, and **Model Persistence**. Lab is lightweight and was designed to easily integrate with your existing training scripts.

> **Warning:** Lab is in active development and the current version of Lab is a beta release. This means that APIs and storage formats are subject to breaking change.

CHAPTER 1

Quickstart

1.1 Installing Lab

For the time being, lab is available through our github repository:

```
git clone https://github.com/beringresearch/lab
cd lab
pip install --editable .
```

Note: You cannot install Lab on the MacOS system installation of Python. We recommend installing Python 3 through the Homebrew package manager using `brew install python`.

1.2 Setting up your first Project

Lab projects are initiated using a `requirements.txt` file. This ensures a consistent and reproducible environment.

Let's create a simple environment that imports sklearn:

```
echo "scikit-learn" >> requirements.txt
lab init --name test
```

Lab will run through project initialisation and create a new **test** project with its own virtual environment.

1.3 Creating your first Lab Experiment

Training scripts can be placed directly into the *test/* directory. Here's an example training script, *train.py*, set up to train a Random Forest classifier with appropriate Lab logging API:

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score

from lab.experiment import Experiment # Import Experiment

e = Experiment() # Initialise Lab Experiment

@e.start_run # Indicate the start of the Experiment
def train():
    iris = datasets.load_iris()
    X = iris.data
    y = iris.target

    X_train, X_test, \
      y_train, y_test = train_test_split(X, y,
                                         test_size=0.24,
                                         random_state=42)

    n_estimators = 100

    e.log_features(['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'])
    clf = RandomForestClassifier(n_estimators = n_estimators)
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average = 'macro')

    e.log_metric('accuracy_score', accuracy)        # Log accuracy
    e.log_metric('precision_score', precision)      # Log aprecision

    e.log_parameter('n_estimators', n_estimators)   # Log parameters of your choice

    e.log_model('randomforest', clf)                # Log the actual model
```

1.4 Running a Lab Experiment

The Experiment can now be launched through:

```
lab run train.py
```

Lab will log performance metrics and model files into appropriate Experiment folders.

1.5 Compare Lab Experiments

Multiple Experiments can be compared from the root of the Project folder:

```
lab ls

Experiment    Source              Date        accuracy_score    precision_score
------------  ------------------  ----------  ----------------  -----------------
```

(continues on next page)

(continued from previous page)

```
49ffb76e      train_mnist_mlp.py  2019-01-15  0.97:   0.97:
261a34e4      train_mnist_cnn.py  2019-01-15  0.98:   0.98:
```

CHAPTER 2

Concepts

Lab is centred around three core concepts: *Reproducibility*, *Logging*, and *Model Persistence*. Lab is designed to integrate with your existing training scripts, with imposing as few constraints as possible.

2.1 Reproducibility

Lab Projects are designed to be shared and re-used. This feature makes havy use of Python's `virtualenv` module, enabling users to precisely define modules and environments that are required to run the associated experiments.

Every Project is initiated using a requirements.txt file.

2.2 Logging

Lab was designed to benchmark multiple predictive models and hyperparameters. To accomplish this, it implements a simple API that stores:

- Feature names
- Hyperparameters
- Performance metrics
- Model files

2.3 Model Persistence

Models are logged using the `joblib` module. This applies to both `sklearn` and `keras` experiments. This simple structure allows for a quick performance assessment and deployment of a model of choice into production.

2.4 Example Use Cases

At Bering, we use Lab for a number of use cases:

Data Scientists track individual experiments locally on their machine, consistently organising all files and artefacts for reproducibility. By setting up a naming schema, Teams can work together on the same datasets to benchmark performance of novel ML algorithms.

Production Engineers assess model performances and decide on the best possible model to be served in production environments. Lab's strict model versioning serves as a link between research and development environment and evolving production components.

ML Researchers can publish code to GitHub as a Lab Project, making it easy for others to reproduce findings.

CHAPTER 3

Command Line Interface

Lab is invoked through a simple Command Line Interface (CLI).

```
lab --help

Usage: lab [OPTIONS] COMMAND [ARGS]...

Bering's Machine Learning Lab

Copyright 2020 Bering Limited. https://beringresearch.com

Options:
--help  Show this message and exit.

Commands:
  config    Global Lab configuration
  info      Display system-wide information
  init      Initialise a new Lab Project
  ls        Compare multiple Lab Experiments
  notebook  Launch a jupyter notebook
  pull      Pulls Lab Experiment from minio to current...
  push      Push Lab Experiment to minio
  rm        Remove a Lab Experiment
  run       Run a training script
  show      Show a Lab Experiment
  update    Update Lab Environment from Project's...
```

3.1 General Parameters

3.1.1 `config minio`

Setup remote minio host

```
Usage: lab config minio [OPTIONS]

Setup remote minio host

Options:
  --tag TEXT        helpful minio host tag  [required]
  --endpoint TEXT   minio endpoint address  [required]
  --accesskey TEXT  minio access key  [required]
  --secretkey TEXT  minio secret key  [required]
  --help            Show this message and exit.
```

tag option is a helpful name to identify a minio endpoint. It can be used to quickly access push and pull APIs.

3.1.2 info

Display system-wide information, including Lab version, number of CPUs, etc.

```
Usage: lab info [OPTIONS]
```

3.2 Project

3.2.1 init

Initialise a new Lab Project.

```
Usage: lab init [OPTIONS]

Options:
  --name TEXT  environment name
  --help       Show this message and exit.
```

Command is run in the presence of a `requirements.txt` file that describes the Project environment. Lab will create a dedicate virtual environemnt in a `.venv` directory.

3.2.2 ls

List Lab Experiments and their performance metrics.

```
Usage: lab ls [OPTIONS] [SORT_BY]

Options:
  --help  Show this message and exit.
```

Optional `SORT_BY` option is a string column name in the results table. For example, if a Lab Experiment logged a metric AUC, calling `lab ls AUC` sort all Experiments by decreasing AUC values. The default is to show the most recently completed Lab run.

3.2.3 show

Create a PNG file of experiment-data-script-hyperparameter-performance diagram.

```
Usage: lab show

Options:
  --help  Show this message and exit.
```

3.2.4 notebook

Lancuhes a jupyter notebook, pointing to the `notebooks` directory. If this is the first time launching the notebook, Lab will automatically create a jupyter kernel using the `requirements.txt` file. Kernel name is stored on your system as TIMESTAMP_PROJECT_NAME.

3.2.5 update

Updates the Lab project. Can be run if the local Lab version was updated or if `requirements.txt` has been modified with additional dependencies.

3.3 Experiment

3.3.1 run

Execute a Lab Experiment.

```
Usage: lab run [OPTIONS] [SCRIPT]...

Options:
  --help  Show this message and exit.
```

3.3.2 rm

Remove a Lab Experiment

```
Usage: lab rm [OPTIONS] EXPERIMENT_ID
```

EXPERIMENT_ID can be obtained by running `lab ls` inside the Project directory.

3.4 Model Management

3.4.1 push

Push Lab Project to a configured minio repository.

```
lab push --tag [MINIO_TAG] --bucket [TEXT] --force.
```

3.4.2 pull

Pull a Lab Project from a configured minio repository.

```
lab pull --tag [MINIO_TAG] --bucket [TEXT] --project [TEXT] --force.
```

CHAPTER 4

Tracking Machine Learning Experiments

The Lab logging component was designed to interface directly with your training code without disrupting the machine learning workflow. Currently, users can keep track of the following experiment artfacts:

- `e.log_features`: Feature names
- `e.log_parameter`: Hyperparameters
- `e.log_metric`: Performance metrics
- `e.log_artifact`: Experimental artifacts
- `e.log_model`: Model persistence

4.1 Feature names

Data features are simply lists of feature names or column indices. Consider the snippet:

```
from sklearn import datasets

iris = datasets.load_iris()
feature_names = iris['feature_names']

print(feature_names)

['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

We can log these features by adding a few lines of code:

```
from sklearn import datasets
from lab.experiment import Experiment #import lab Experiment

```

(continues on next page)

(continued from previous page)

```
e = Experiment()

# Initialize Lab Experiment
@e.start_run
def train():
    iris = datasets.load_iris()
    feature_names = iris['feature_names']

    # Log features
    e.log_features(feature_names)
```

4.2 Hyperparameters: `e.log_parameter`

Let's carry on with the Iris dataset and consider a Random Forest Classifier with an exhaustive grid search along the number of trees and maximum depth of a tree:

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from lab.experiment import Experiment #import lab Experiment

e = Experiment()

# Initialize Lab Experiment
@e.start_run
def train():
    iris = datasets.load_iris()

    feature_names = iris['feature_names']

    # Log features
    e.log_features(feature_names)

    parameters = {'n_estimators': [10, 50, 100],
                  'max_depth': [2, 4]}

    rfc = RandomForestClassifier()

    # Run a grid search
    clf = GridSearchCV(rfc, parameters)
    clf.fit(iris.data, iris.target)

    best_parameters = clf.best_estimator_.get_params()

    # Log parameters
    e.log_parameter('n_estimators', best_parameters['n_estimators'])
    e.log_parameter('max_depth', best_parameters['max_depth'])
```

4.3 Performance Metrics: `e.log_metric`

Lab was designed to easily compare multiple machine lerning experiments through consistent performance metrics. Let's expand our example and assess model accuracy and precision.

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score
from lab.experiment import Experiment

e = Experiment()

# Initialize Lab Experiment
@e.start_run
def train():
    iris = datasets.load_iris()

    feature_names = iris['feature_names']

    # Log features
    e.log_features(feature_names)

    parameters = {'n_estimators': [10, 50, 100],
                  'max_depth': [2, 4]}

    # Run a grid search
    rfc = RandomForestClassifier()
    clf = GridSearchCV(rfc, parameters)
    clf.fit(iris.data, iris.target)

    best_parameters = clf.best_estimator_.get_params()

    # Log parameters
    e.log_parameter('n_estimators', best_parameters['n_estimators'])
    e.log_parameter('max_depth', best_parameters['max_depth'])

    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                            test_size=0.25, random_state=42)

    rfc = RandomForestClassifer(n_estimators = best_parameters['n_estimators'],
                                max_depth = best_parameters['max_depth'])
    rfc.fit(X_train, y_train)

    # Generate predictions
    y_pred = rfc.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average = 'macro')

    # Log performance metrics
    e.log_metric('accuracy_score', accuracy)
    e.log_metric('precision_score', precision)
```

4.4 Experiment Artifacts: `e.log_artifact`

In certain cases, it may be desirable for a Lab Experiment to write certain artifacts to a temporary folder - e.g. ROC curves or Tensorboard log directory. Lab naturally bundles these artifacts within each respective experiment for subsequent exploration.

Let's explore an example where Lab logs Tensorboard outputs:

```
# Additional imports would go here
from keras.callbacks import TensorBoard
import tempfile

from lab.experiment import Experiment

e = Experiment()

@e.start_run
def train():

  # ... Further training code goes here

  # Create a temporary directory for tensorboard logs
  output_dir = dirpath = tempfile.mkdtemp()
  print("Writing TensorBoard events locally to %s\n" % output_dir)

  tensorboard = TensorBoard(log_dir=output_dir)

  model.fit(x_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            verbose=1,
            validation_data=(x_test, y_test),
            callbacks=[tensorboard])

  # Log tensorboard artifact
  e.log_artifact('tensorboard', output_dir)
```

In this example, Tensorboard logs are written to a temporary folder, which can be tracked in real-time. Once the run is complete, Lab moves all the directory content into a subdirectory of the current Lab Experiment.

4.5 Model Persistence: `e.log_model`

Finally, it's useful to store model objects themselves for future use. Consider our fitted GridSearchCV object `clf` from an earlier example. It can now be logged using a simple expression:

```
e.log_model('GridSearchCV', clf)
```

CHAPTER 5

Managing Deep Learning Experiments

Deep Learning experiment lifecycle generates a rich set of data artifacts, e.g., expansive datasets, complex model architectures, varied hyperparameters, learned weights, and training logs. To produce an effective model, a researcher often has to iterate over multiple scripts, making it challenging to reproduce complex experiments.

Lab functionality offers a clean and standardised interface for managing the many moving parts of a Deep Learning experiment.

5.1 MNIST Example

Consider the following lab training script. Let's set up our hyperparameters and training, validation, testing sets:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from keras.callbacks import TensorBoard

import tempfile

from sklearn.metrics import accuracy_score, precision_score

from lab.experiment import Experiment

BATCH_SIZE = 128
EPOCHS = 20
CHECKPOINT_PATH = 'tf/weights'
num_classes = 10


# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

(continues on next page)

(continued from previous page)

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Set up a simple model and train:

```
e = Experiment()


@e.start_run
def train():

    # Create a temporary directory for tensorboard logs
    output_dir = tempfile.mkdtemp()
    print("Writing TensorBoard events locally to %s\n" % output_dir)
    tensorboard = TensorBoard(log_dir=output_dir)

    # During Experiment execution, tensorboard can be viewed through:
    # tensorboard --logdir=[output_dir]

    model.fit(x_train, y_train,
              batch_size=BATCH_SIZE,
              epochs=EPOCHS,
              verbose=1,
              validation_data=(x_test, y_test),
              callbacks=[tensorboard])

    model.save_weights(CHECKPOINT_PATH)

    y_prob = model.predict(x_test)
    y_classes = y_prob.argmax(axis=-1)
    actual = y_test.argmax(axis=-1)

    accuracy = accuracy_score(y_true=actual, y_pred=y_classes)
    precision = precision_score(y_true=actual, y_pred=y_classes,
                                average='macro')

    # Log tensorboard
    e.log_artifacts('tensorboard', output_dir)
    e.log_artifacts('weights', CHECKPOINT_PATH)

    # Log all metrics
    e.log_metric('accuracy_score', accuracy)
    e.log_metric('precision_score', precision)

    # Log parameters
    e.log_parameter('batch_size', BATCH_SIZE)
```

(continues on next page)

(continued from previous page)

```
    e.log_parameter('epochs', EPOCHS)
```

When training on distributed systems with Horovod, *model.fit* element can be abstracted into a file, say *horovod-train.py* and called directly from the *train()* method:

```
import subprocess

args = ['-np', str(8), # 8 GPUs
        '-H', 'localhost:8', 'python',
        'horovod-train.py',
        '--checkpoint', CHECKPOINT_PATH,
        '--batch-size', BATCH,
        '--epochs', EPOCHS]
```

Note that you need to enable your Horovod script to accept some basic model hyperparameters that you wish to log downstream.

CHAPTER 6

Working with Jupyter Notebooks

Lab makes it easy to work with Jupyter notebooks by creating a kernel directly from a lab project

```
lab notebook
```

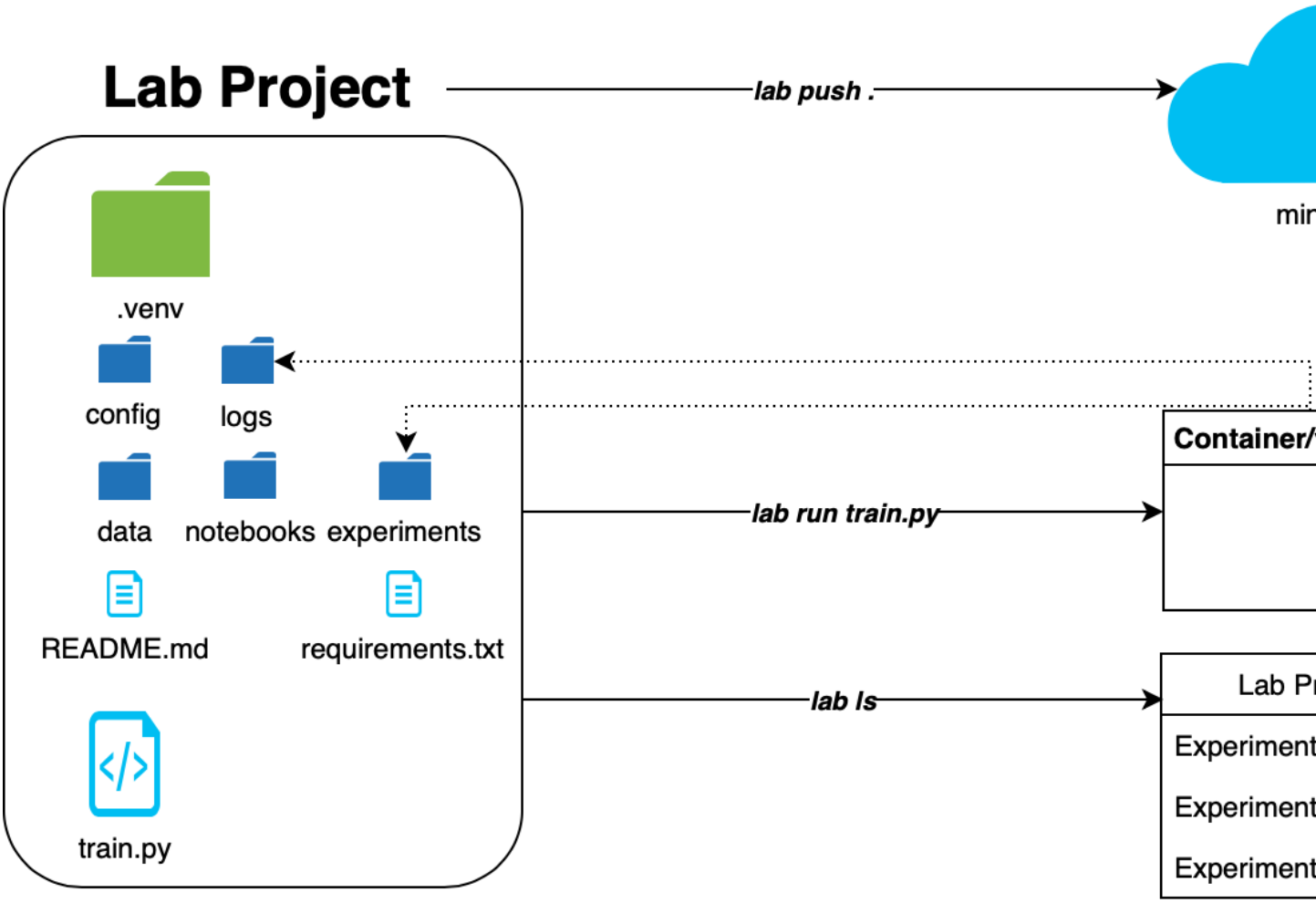
Once the kernel is created, you can select it from any Jupyter session.

Lab also provides a *notebooks* directory to organise and maintain all notebooks associated with a Lab Project.

CHAPTER 7

Model Repository

Lab uses minio to store Projects. Minio is a high performance distributed object storage server, designed for large-scale private cloud infrastructure. This makes it a great fit as a storage environment for multiple Lab Projects and Experiments. Lab makes it trivial to back up completed Projects and share them across teams.



7.1 Configuring minio server

There are a number of ways to install minio on a wide range of operating systems. See more details installation instructions in minio documentation pages.

7.2 Setting up Lab minio interface

Once minio is up and running, you will need to make a note of the `endpoint`, `access key`, and `secret key`. Lab supports multiple minio configurations through a convenient tagging system. Each configuration can be set up through CLI:

```
lab config minio --tag [MINIO_TAG] -- endpoint [TEXT] --accesskey [TEXT] --secretkey␣
↪[TEXT]
```

Note that the endpoint is simply an IP address and port of a minio host, e.g. `192.168.1.50:9000`.

7.3 Storing Lab Projects

Lab Projects can be pushed to a specific minio host by running a simple command from the Project root folder:

```
lab push --tag [MINIO_TAG] --bucket [TEXT] .
```

Here, `--tag` specifies a nickname of an exisiting minio connection and `--bucket` refers to a unique destination name on minio host, analogous to an S3 bucket.

Each project contains a *.labignore* file that specifies intentionally untracked files to ignore during a push. A default *.labignore* will omit the virtual environment directory *.venv*. Further omitions can be specified on each line:

```
.venv
data
experiments/abcdefgh/model.joblib
```

7.4 Pruning remote repository

Sometimes it may be desirable to prune a remote repository. Pruning simply replaces the entire content of a remote repository with local files. The user is warned just before proceding, as this operation can have undersirable consequences.

```
lab push --tag [MINIO_TAG] --bucket [TEXT] --force .
```

7.5 Pulling from a remote repository

To retrieve a Lab Project from a minio host, run a simple command from folder into which you'd like to pull the Project:

```
lab pull --tag [MINIO_TAG] --bucket [TEXT] --project [TEXT].
```

In cases where connection with minio has already been establish, a project can be pushed/pulled directly from the project directory via `lab push` or `lab pull` without further options.

CHAPTER 8

Frequently Asked Questions

8.1 How can I include a `github` repository in a lab Project

Like `pip`, lab works with `requirements.txt` file. To let lab know that your virtual environment should contain a package maintained on github, add the following line to your `requirements.txt`:

```
-e git+https://github.com/beringresearch/ivis#egg=ivis
```

Modify repository and package information accordingly.

CHAPTER 9

Examples Gallery

Several examples of how Lab can be used in common machine learning projects.

9.1 Getting started with Lab and scikit-learn

This example illustrates how Lab can be used to create and run a simple classifier on the iris dataset.

Begin by creating a new Lab Project:

```
>>> echo "scikit-learn" > requirements.txt
>>> lab init --name simple-iris
```

```
import argparse
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score

from lab.experiment import Experiment

parser = argparse.ArgumentParser('Test arguments')

parser.add_argument('--n_estimators', type=int, dest='n_estimators')
args = parser.parse_args()

n_estimators=args.n_estimators

if n_estimators is None:
    n_estimators=100
    max_depth=2

if __name__ == "__main__":
    e = Experiment(dataset='iris_75')
```

(continues on next page)

(continued from previous page)

```
    @e.start_run
    def train():
        iris = datasets.load_iris()
        X = iris.data
        y = iris.target

        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.25,
                                                            random_state=42)

        e.log_features(['Sepal Length', 'Sepal Width', 'Petal Length',
                        'Petal Width'])
        clf = RandomForestClassifier(n_estimators=n_estimators)

        clf.fit(X_train, y_train)

        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average = 'macro')

        e.log_metric('accuracy_score', accuracy)
        e.log_metric('precision_score', precision)

        e.log_parameter('n_estimators', n_estimators)
        e.log_parameter('max_depth', max_depth)

        e.log_model('randomforest', clf)
```

After execute training script through the *lab run* command.

```
>>> lab run train.py
>>> lab ls
```

Total running time of the script: (0 minutes 0.000 seconds)

9.2 Running Keras models with Tensorboard

Lab integrates into a typical keras workflow.

WARNING: model persistence in Keras can be complicated, especially when working with complext models. It is recommended to checkpoint each training epoch independently from Lab's `log_model` API.

Bering by creating a new Lab Project:

```
>>> echo "keras" > requirements.txt
>>> lab init --name simple-keras
```

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from keras.callbacks import TensorBoard
```

(continues on next page)

(continued from previous page)

```
import tempfile

from sklearn.metrics import accuracy_score, precision_score

from lab.experiment import Experiment

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

e = Experiment()


@e.start_run
def train():

    # Create a temporary directory for tensorboard logs
    output_dir = tempfile.mkdtemp()
    print("Writing TensorBoard events locally to %s\n" % output_dir)
    tensorboard = TensorBoard(log_dir=output_dir)

    # During Experiment execution, tensorboard can be viewed through:
    # tensorboard --logdir=[output_dir]

    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(x_test, y_test),
              callbacks=[tensorboard])
```

(continues on next page)

(continued from previous page)

```
    y_prob = model.predict(x_test)
    y_classes = y_prob.argmax(axis=-1)
    actual = y_test.argmax(axis=-1)

    accuracy = accuracy_score(y_true=actual, y_pred=y_classes)
    precision = precision_score(y_true=actual, y_pred=y_classes,
                                average='macro')

    # Log tensorboard
    e.log_artifacts('tensorboard', output_dir)

    # Log all metrics
    e.log_metric('accuracy_score', accuracy)
    e.log_metric('precision_score', precision)

    # Log parameters
    e.log_parameter('batch_size', batch_size)

    # Save model
    e.log_model('mnist-mlp', model)
```

Total running time of the script: (0 minutes 0.000 seconds)