How to run and track machine learning models in R with MLflow

This article has briefly described what ***MLflow*** is and how it works. *MLflow* currently provides APIs in Python language that users can invoke in their machine learning source codes to log parameters, metrics, and artifacts to be tracked by the *MLflow* tracking server.

Users familiar with R and perform machine learning operations in R may like to track their models and every runs with *MLflow*. There are several approaches users can take.

- Waiting for *Mlflow* to release the APIs in R, or
- Wrapping *MLflow* RESTful APIs and logging through `curl` commands, or
- Calling existing Python APIs with some R packages that can invoke Python interpreter

The last approach is simple and easy enough while allows users to interact with *MLflow* without waiting for R APIs to be available. This tutorial will illustrate how to achieve this with R package ***reticulate***.

reticulate is an open source R package that allows to call Python from R by embedding a Python session within the R session. It provides seamless and high-performance interoperability between R and Python. The package is available in CRAN repository.

MLflow also comes with a `Projects` component that packs data, source code with commands, parameters and execution environment setup together as a self-contained specification. Once a `MLproject` is defined, users can run it everywhere. Currently `MLproject` can run Python code or shell command. It can also set up the Python environment for the project specified in the `conda.yaml` file defined by users.

For R users, it is common to load some packages in the R source codes. These packages need to installed for the R code to run. In the future, it could be a good enhancement for *MLflow* to add something similar to `conda.yaml` to set up R package dependencies. This tutorial will show how to create a `MLproject` containing R source code and run it with **`mlflow run`** command.

Learning objectives

In this tutorial, developers will install and set up the *MLflow* environment, train and track machine learning models in R, package source codes and data in a `MLproject` and run with `mlflow run` command.

Prerequisites

Before beginning this tutorial, you should have Python installed on the platform where R is running. I prefer installing miniconda. Since the machine learning training will be done in R, R should be already installed on the platform as well.

Estimated time

Completing this tutorial should take approximately 30 minutes.

Steps

Step 1: Install *MLflow*

Create a virtualenv for *MLflow* and install mlflow package as follow (with `conda`):

```
conda create -q -n mlflow python=3.6
source activate mlflow
pip install -U pip
pip install mlflow
```

Step 2: Install `reticulate` R package

Install reticulate package through R.

```R
install.packages("reticulate")
```

`reticulate` allows R to call Python functions seamlessly. The Python package is loaded by the `import` statement. Calling to a function is through `$` operator.

```R
> library(reticulate)
> path <- import("os.path")
> path$isdir("/tmp")
[1] TRUE
```

As you can see above, it is very simple to call Python functions in `os.path` module from R with this package. So you can do the same thing with `mlflow` package by importing it and then call `mlflow$log_param` and `mlflow$log_metric` to log parameters and metrics for the R script.

Step 3: Train a GLM model with SparkR

Following R script builds a linear regression model with SparkR. You need `SparkR` package installed for this example.

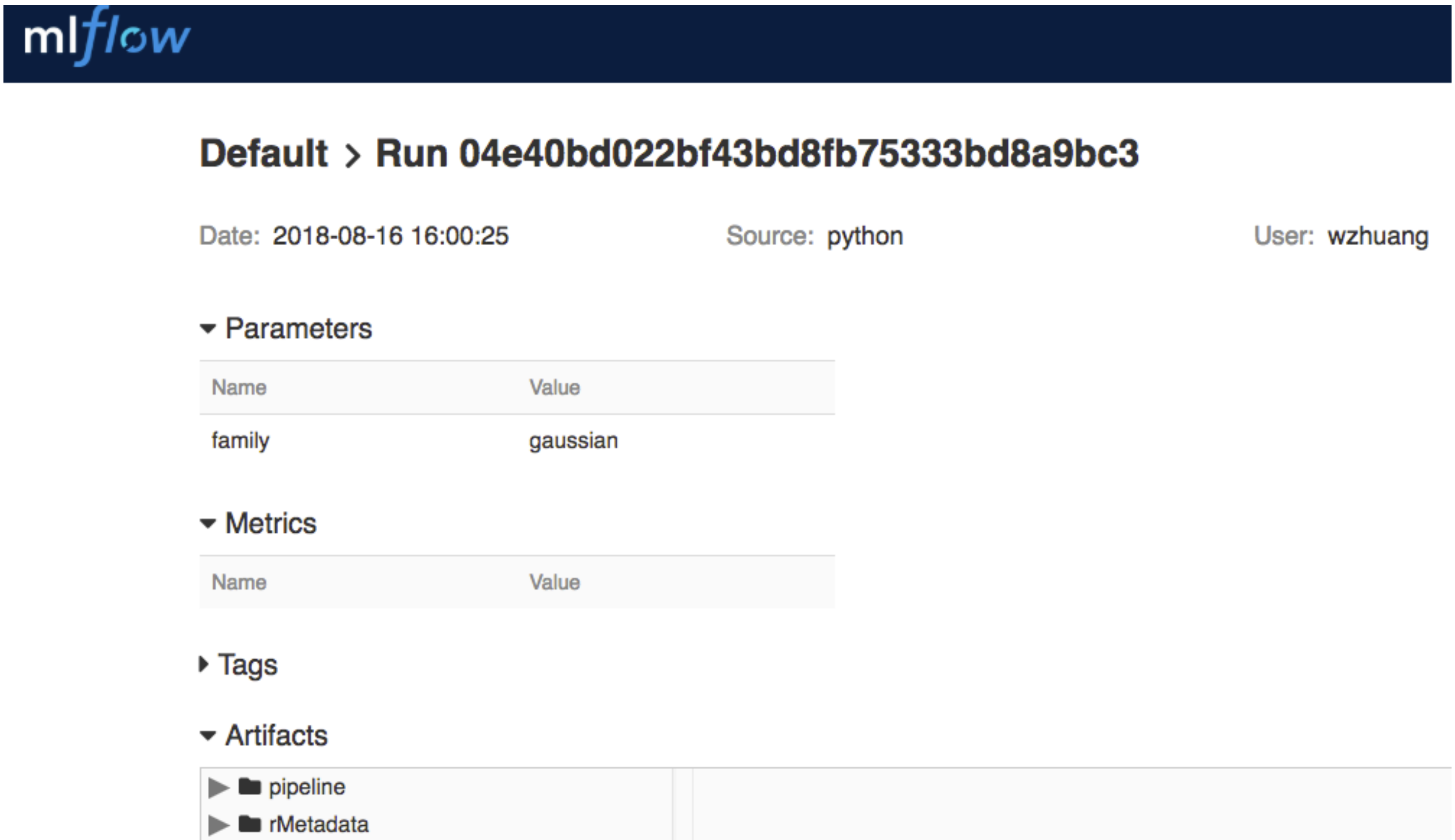
```R
# load the reticulate package and import mlflow Python module
library(reticulate)
mlflow <- import("mlflow")

# load SparkR package and start spark session
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
sparkR.session(master="local[*]")

# convert iris data.frame to SparkDataFrame
df <- as.DataFrame(iris)

# parameter for GLM
family <- c("gaussian")

# log the parameter
mlflow$log_param("family", family)

# fit the GLM model
model <- spark.glm(df, Species ~ ., family = family)

# exam the model
summary(model)

# path to save the model
model_path <- "/tmp/mlflow-GLM"

# save the model
write.ml(model, model_path)

# log the artifact
mlflow$log_artifacts(model_path)

# stop spark session
sparkR.session.stop()
```

You can either copy the script to `R` or `Rstudio` and run interactively, or save it to a file and run with

`Rscript` command. Make sure that the `PATH` environment variable includes the path to the *mlflow* Python virtualenv.

Step 4: Launch the *MLflow* UI

Launch *MLflow* UI by running `mlflow ui` command from a shell. Then open browser and go to page link with url `http://127.0.0.1:5000`. Your previous GLM model training is now showing and so it can be tracked. Here is a snapshot.



Step 5: Train a decision tree model

Download the [`wine-quality.csv`]() data to be learned to your platform.

Install the `rpart` package on your R environment:

```R
install.packages("rpart")
```

Follow this example [`rpart-example.R`]() to fit a tree model:

```r
# Source prep.R file to install the dependencies
source("prep.R")

# Import mlflow python package for tracking
library(reticulate)
mlflow <- import("mlflow")

# Load rpart to build a tree model
library(rpart)

# Read in data
wine <- read.csv("wine-quality.csv")

# Build the model
fit <- rpart(quality ~ ., wine)

# Save the model that can be loaded later
saveRDS(fit, "fit.rpart")

# Save the model to mlflow tracking server
mlflow$log_artifact("fit.rpart")

# Plot
jpeg("rplot.jpg")
par(xpd=TRUE)
plot(fit)
text(fit, use.n=TRUE)
dev.off()

# Save the plot to mlflow tracking server
mlflow$log_artifact("rplot.jpg")
```

The R code above includes three parts: the model training, the artifacts logging through *MLflow*, and the R package dependencies installation.

Step 6: Prepare package dependencies for MLproject

In above example, these two R packages, `reticulate` and `rpart`, are required for the code to run. To pack these codes into a self-contained project, some sort of script should be run to automatically install these packages if the platform does not have them installed.

Any specific R package needed for the project is going to be installed through [`prep.R`] with these codes:

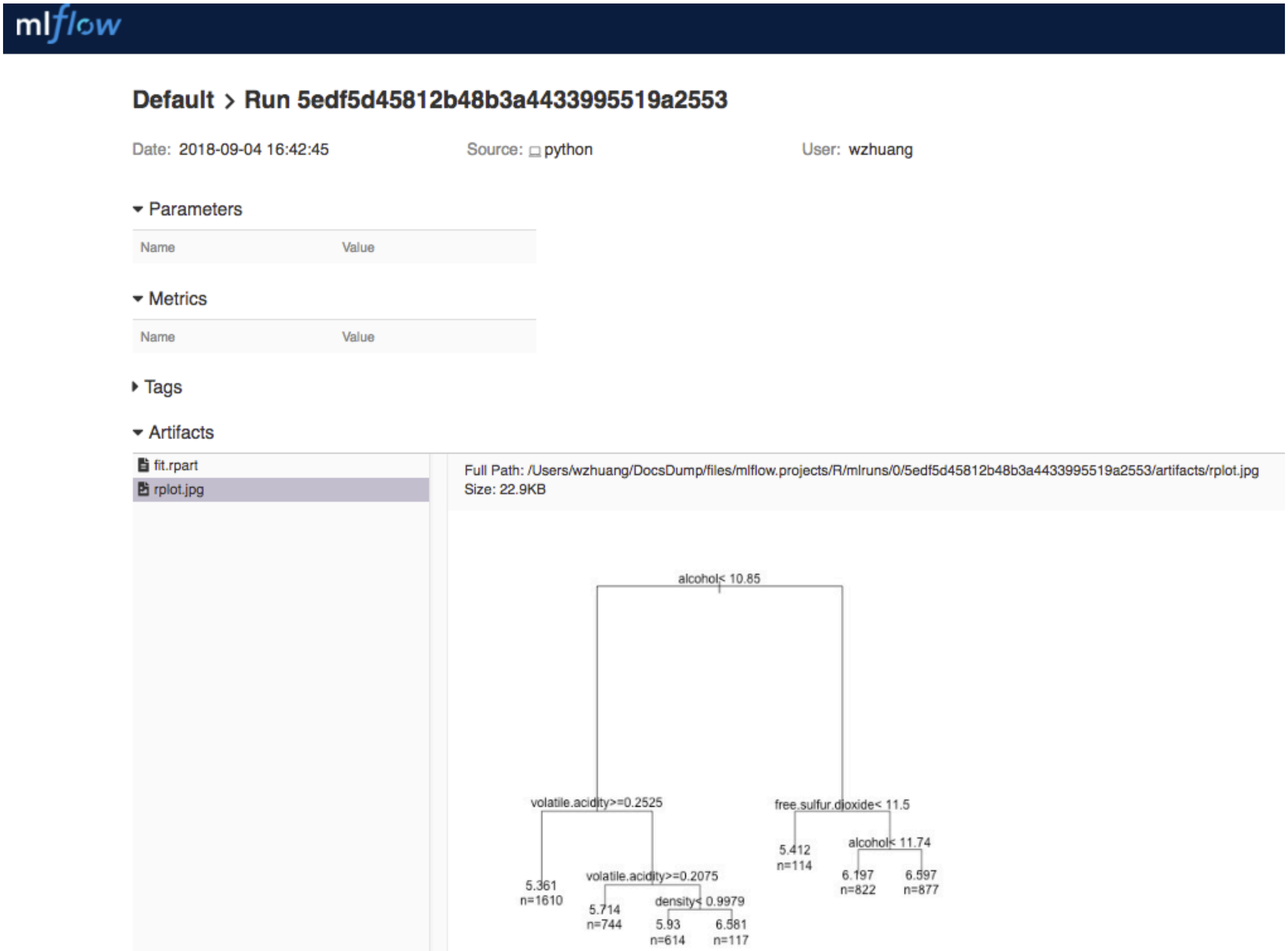
```r
# Accept parameters, args[6] is the R package repo url
args <- commandArgs()

# All installed packages
pkgs <- installed.packages()

# List of required packages for this project
reqs <- c("reticulate", "rpart")

# Try to install the dependencies if not installed
sapply(reqs, function(x){
  if (!x %in% rownames(pkgs)) {
    install.packages(x, repos=c(args[6]))
  }
})
```

Step 7: Test your codes

Before packaging these into a *MLproject*, try to test by directly invoking `Rscript` command as follow:

```
Rscript rpart-example.R https://cran.r-project.org/
```

From the *MLflow* UI, you should see this run been tracked like this screen snapshot:



Step 8: Create a MLproject

Now let's write the spec and pack this project into a *MLproject* that *MLflow* knows to run. All needed to be done is creating the [`MLproject`]() file in the same directory.

```yaml
name: r_example

entry_points:
    main:
        parameters:
            r-repo: {type: string, default: "https://cran.r-project.org/"}
        command: "Rscript rpart-example.R {r-repo}"
```

In this file, it defines a `r_example` project with a `main` entry point. The entry point specifies the command and parameters to be executed by the `mlflow run`. For this project, `Rscript` is the shell command to invoke the R source code. `r-repo` parameter provides the URL string where the dependent packages can be installed from. A default value is set. This parameter is passed to the command running the R source code.

You have all the files required to train this tree model, you can create a `MLproject` by creating a directory and copying the data and R source codes to that directory.

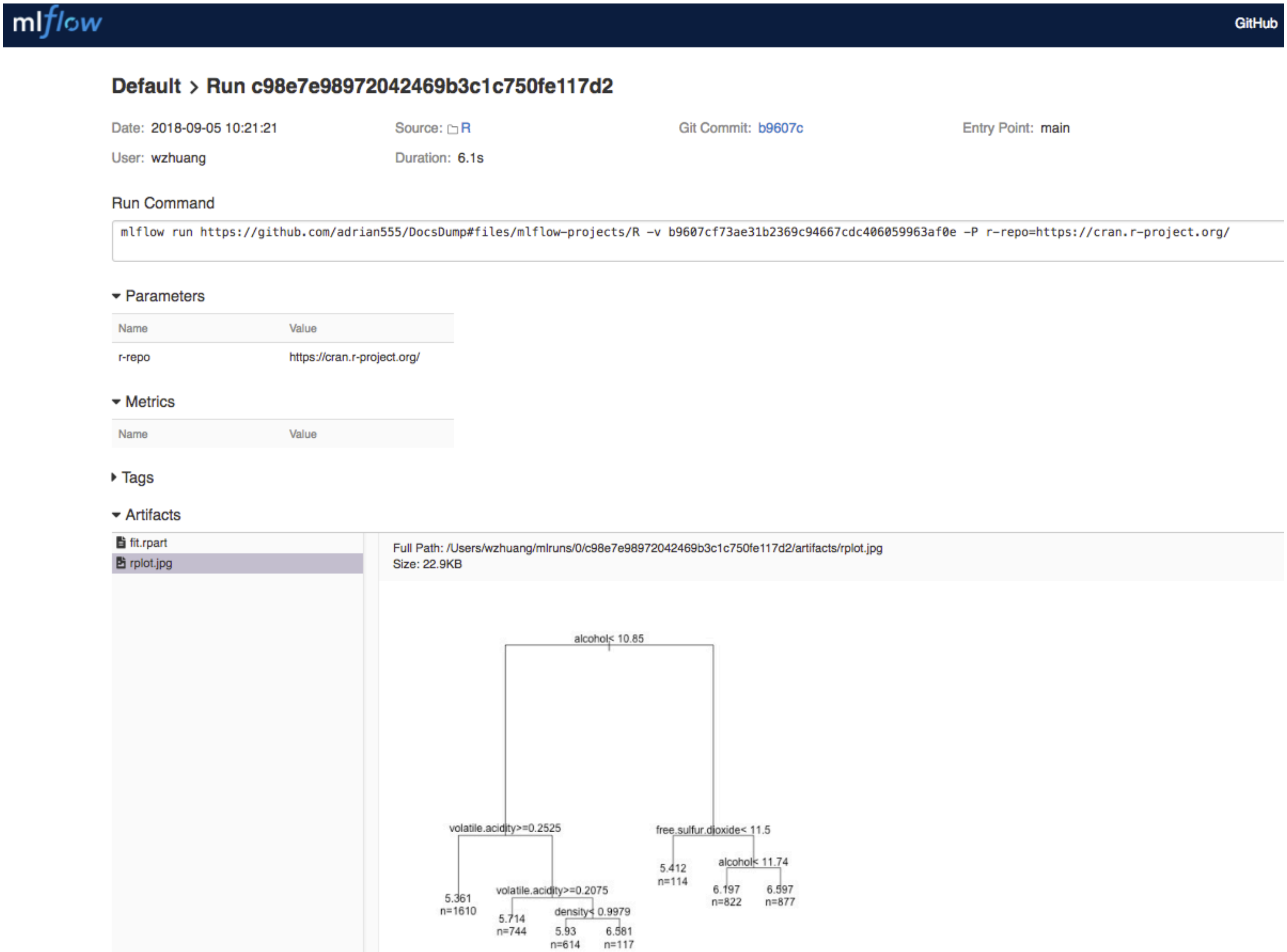
```
.
└── R
    ├── MLproject
    ├── prep.R
    ├── rpart-example.R
    └── wine-quality.csv
```

Step 9: Check in and test the MLproject

The above `MLproject` can be checked in and pushed to github repository. To test the project, with following command, it can be run on any platform that has R installed.

```
mlflow run https://github.com/adrian555/DocsDump#files/mlflow-projects/R
```

The project can also be viewed from the *MLflow* tracking UI like this screen snapshot:



The differences between this view and previous run without `Mlproject` spec are the `Run Command` which captures the exact command to run the project, and the `Parameters` which automatically logs any parameters passed to entry points.

Summary

In this tutorial, you have successfully created a `MLproject` in R, track and run it with *MLflow*. The approach taken here lets R users take benefit of *MLflow* `Tracking` component and track their R models in a quick way. It also demonstrates what `Projects` component of *MLflow* is designed for - to define the project and make it easily to be rerun. R users can quickly set up their projects and enjoy the easiness of tracking and running projects with *MLflow* once going through this tutorial.