flor Documentation

Release 0.0

UC Berkeley, RISE Lab

Sep 29, 2022

INSTALLATION

Flor is a diagnostics and sharing system for the Machine Learning Lifecycle. Flor transparently captures complete execution traces (CETs) and surfaces them as relational tables with minimal human guidance. You can run SQL queries over CETs by annotating your code with log statements — after your code executes. Run your ML code with confidence: you won't forget to track something important. When you need the value of some expression in your code, just click on it and give it a name, we'll do ETL. Ask your questions in SQL.

Flor is a work in progress. You are welcome to try it now and give us feedback via Github Issues. We're working on making Flor run faster, without sacrificing completeness, transparency, or ease of use.

CHAPTER

ONE

INSTALLATION

Flor requires Python 3.6 or higher. Flor requires Anaconda.

1. Run the command: `pip install pyflor`
2. Run the command, following the instructions as prompted: `pyflor_install`

CHAPTER

TWO

STEP 1: EXECUTE YOUR CODE, GENERATING A COMPLETE EXECUTION TRACE

Below is the well-known scikit-learn Iris example. We've modified this example to do a small hyper-parameter sweep over gamma, a hyper-parameter for the Support Vector Classifier.

For what value of gamma do we get the best score?

To understand the motivation behind Flor, resist the urge to add print statements. Flor will capture everything for you, we'll extract it in a future step.

```
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import train_test_split
import random

iris = datasets.load_iris()
X_tr, X_te, y_tr, y_te = train_test_split(iris.data,
                                          iris.target,
                                          test_size=0.15,
                                          random_state=random.randint(1,100))
for g in [0.1, 0.01, 0.001]:
    clf = svm.SVC(gamma=g, C=100.0)
    clf.fit(X_tr, y_tr)
    score = clf.score(X_te, y_te)

print('--- Job Finished ---')
```

Copy and paste the Iris example above and name it `iris.py`, preferably in a new directory that you don't mind Flor modifying.

From the directory you just created, containing `iris.py`, run:

```
flor python iris.py iris_demo
```

The command tells Flor to capture a Complete Execution Trace for `python iris.py`, and to name this experiment `iris_demo`.

When the execution finishes, you can peek at the Complete Execution Trace (CET) at `~/.flor/iris_demo/log.json`, we show you this now as evidence that Flor captured the contents of the execution.

In the next steps, we'll show you how Flor can transform that CET into a CSV table.

CHAPTER

THREE

STEP 2: ANNOTATE YOUR CODE, CHOOSING AND NAMING THE EXPRESSIONS YOU WANT TO EXTRACT FROM THE CET

Want to know the value for the seed that we used in `train_test_split`? Then, we annotate the expression that generated that value: `random.randint(1,100)`.

Here is the syntax for a Flor Annotation:

```
GET("name_of_expression", e)
```

An annotation lets you choose which expressions in the code you care about, and it gives you an opportunity to name them, so that there are no ambiguities.

Let's annotate `iris.py`. We don't want the annotations to modify the original file, so let's work on a copy of the file. Flor also needs some metadata, such as the origins of the file, so we will call a special Flor command to copy the file and put the metadata at the top of the file.

```
flor cp iris.py iris_h.py
```

Now, `iris_h.py` is a copy of the file that you can annotate. You should annotate this file next.

Below is an example annotated file:

```
#/Users/rogarcia/sandbox/iris.py
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import train_test_split
import random

iris = datasets.load_iris()
X_tr, X_te, y_tr, y_te = train_test_split(iris.data,
                                          iris.target,
                                          test_size=GET("test_size", 0.15),
                                          random_state=GET("random_state", random.
↪randint(1,100)))
for g in [0.1, 0.01, 0.001]:
    clf = svm.SVC(gamma=GET("gamma", g), C=GET("C", 100.0))
    clf.fit(X_tr, y_tr)
    score = GET("score", clf.score(X_te, y_te))

print('--- Job Finished ---')
```

There are 5 annotations in the example above. Your annotated file should also contain a comment at the top with the path where `iris.py` was run.

The proper way to annotate your code is on a GUI: you simply highlight the expressions you want and give them a name. Flor includes a PyCharm plugin that allows you to annotate your code more easily. You can find the plugin here. Documentation for the PyCharm plugin will follow shortly.

CHAPTER

FOUR

STEP 3: SCAN THE CETS TO GENERATE A TABLE

At this point, we have a CET and an annotated Python script. The CET is a durable copy of a previous execution. Flor uses the annotated Python script `iris_h.py` to identify the values you want to fetch from the CET. Then, Flor scans the CET to extract the values and transform them into a relational table.

```
flor etl iris_demo iris_h.py
```

`iris_demo` is the name of the experiment and `iris_h.py` is the path to the annotated file.

When the program finishes, you can view the results in `iris_demo.csv`.

CHAPTER

FIVE

LICENSE

Flor is Licensed under the Apache V2 License.

Flor requires Python 3.6 or higher. Flor requires Anaconda.

5.1 Install Flor from Pip

1. Run the command: `pip install pyflor`
2. Run the command, following the instructions as prompted: `pyflor_install`

5.2 Uninstall Flor

1. Run the command, `pyflor_uninstall`

5.3 Contact

Flor is developed and maintained by the RISE Lab.

For bug reports and feature requests, please open GitHub Issues.