I3: Metaflow

Jointly submitted by Tasheena Narraidoo and Yuanqi Zhao

1. Why Metaflow

Data science has been applied to a wide range of businesses to help people make decisions. Companies in different industry areas hire data scientists to deal with diverse real-life problems such as predicting sales and recommending movies. Solving real-life problems is unlike finishing up a lab when data scientists are trained at school -- you can get full points for your lab as long as your model's accuracy tested on a limited and cleaned-up dataset has reached a given threshold provided by the graders. When building and deploying models in production, data scientists need to also think in the shoes of a software engineer. A typical data science project in production touches all layers of the stack including model development, feature engineering, model operations, versioning, architecture, job scheduler, computer resources and data warehouse. And the success of a data science project is not judged only by the model's accuracy, but also depends on qualities in a system-level such as scalability, latency, computational costs, etc.

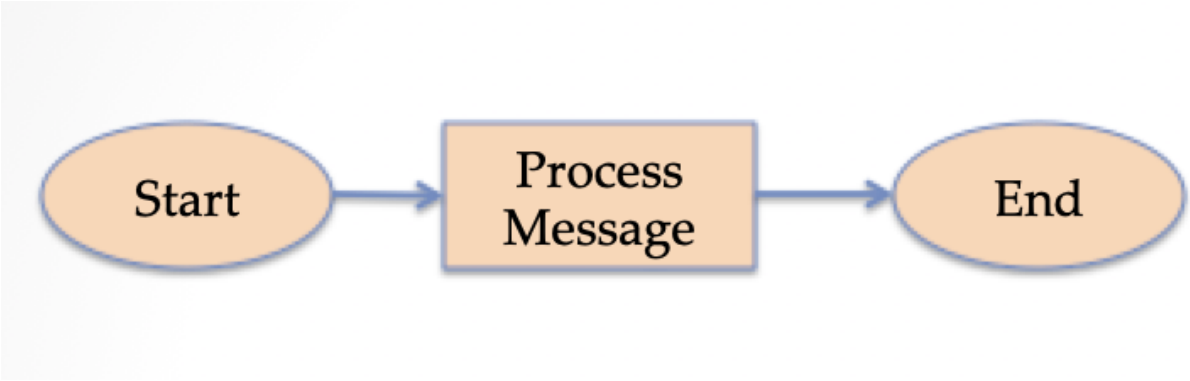
However, it is often the case that data scientists focus more on modeling and feature engineering, and lack of training or experience in some of the other layers just as the figure below shows. It is true that tools exist for each single layer of the stack. For example, in Netflix, they use Amazon S3 as the data warehouse, Spark as the query engine, Titus for compute resources, Meson as the job scheduler, and Nteract for prototyping. Data scientists can certainly learn how to use them and integrate those tools into their work, but then more than half of their codes would be related to the infrastructure, and a significant part of their time would not be spent in their expertise area which is model development and feature engineering. Metaflow makes it all easier. It provides a unified API to the infrastructure stack that is required to execute data science projects, from prototype to production. In addition, Metaflow does not focus on any specific use cases, but a wide variety of ML use cases so that it is highly usable in lots of settings.

Data Science vs Infrastructure Concerns [3]

2. Application on Movie Recommendation System

We now illustrate MetaFlow with a movie recommendation system. Our goal is to train a machine learning model and provide our recommendation movie list once we receive a request. We would use MetaFlow to help with the infrastructure including managing compute resources, performing containerized runs, versioning, replaying and resuming workflow runs so that we can focus more on the model development. Before integrating Metaflow into the movie recommendation system, we first explain the MetaFlow using a simple example.

Metaflow flows are Directed Acyclic Graphs (DAGs) which consists of a set of processing steps and their dependencies. As shown below in the figure, a node in the graph represents a process in the workflow and the arrows represent their dependencies. In fact, each flow can be represented by a standard python class. This architecture allows users to use any external libraries from the conda ecosystem in Metaflows without the need for plugins.



Simple Three Node MetaFlow Example

Below is a code snippet showing how we construct a simple flow with the three nodes shown above. In the codes, all the nodes are connected with “self.next()”. Each node is programmed as a python class tagged with a “@step” decorator.

```python
from metaflow import FlowSpec, step

class LinearFlow(FlowSpec):

    """
    A flow to verify you can run a basic metaflow flow.
    """

    # Global initializations here

    @step
    def start(self):
        self.message = 'Thanks for reading.'
        self.next(self.process_message)

    @step
    def process_message(self):
        print('the message is: %s' % self.message)
        self.next(self.end)

    @step
    def end(self):
        print('the message is still: %s' % self.message)

if __name__ == '__main__':
    LinearFlow()
```

metaflow-linear-example.py hosted with ❤ by GitHub

Python Class representing a simple linear flow diag [3]

We can simply put the code snippet in a python file and run it using the command `python test.py run`. The following is our result.

```
→  8:22PM yuanqizhao Desktop (main) ✗ 🧚   python test.py run
Metaflow 2.2.8 executing LinearFlow for user:yuanqizhao
Validating your flow...
    The graph looks good!
Running pylint...
    Pylint is happy!
2021-03-24 20:22:34.176 Workflow starting (run-id 1616642554170336):
2021-03-24 20:22:34.183 [1616642554170336/start/1 (pid 58265)] Task is starting.
2021-03-24 20:22:34.543 [1616642554170336/start/1 (pid 58265)] Task finished successfully.
2021-03-24 20:22:34.548 [1616642554170336/process_message/2 (pid 58269)] Task is starting.
2021-03-24 20:22:34.879 [1616642554170336/process_message/2 (pid 58269)] the message is: Thanks for reading.
2021-03-24 20:22:34.914 [1616642554170336/process_message/2 (pid 58269)] Task finished successfully.
2021-03-24 20:22:34.918 [1616642554170336/end/3 (pid 58273)] Task is starting.
2021-03-24 20:22:35.237 [1616642554170336/end/3 (pid 58273)] the message is still: Thanks for reading.
2021-03-24 20:22:35.270 [1616642554170336/end/3 (pid 58273)] Task finished successfully.
2021-03-24 20:22:35.271 Done!
```

Sample Running Result

MetaFlow for Movie Recommendation System

Next, we use Metaflow to help us build the movie recommendation system. We have combined a sample movie data frame provided by MetaFlow with the ratings sample from our GitHub repo. Below is a sample code and a snapshot of the combined dataset.

```
library(metaflow)
clean_ratings <- read.csv("clean_ratings.txt") # our github repo
movies <- read.csv("movies.csv") # from metaflow
write.csv(df,"smallExample.csv")

# code from Metaflow library

library(metaflow)

# Parse the CSV file
start <- function(self){
  self$df <- read.csv("./movies.csv", stringsAsFactors=FALSE)
}

# Filter the movies by genre.
pick_movie <- function(self){
  # select rows which has the specified genre
  movie_by_genre <- self$df[self$df$genre == self$genre, ]

  # randomize the title names
  shuffled_rows <- sample(nrow(movie_by_genre))
  self$movies <- movie_by_genre[shuffled_rows, ]
}
```

The above figure is a snapshot of the metaflow library we have been exploring. The Metaflow library is filtering the movies list by genre and shuffles the data before randomly choosing five movie titles.

```
> df2
  movie_title title_year    genre     gross   movieid mean_ratings
1        2012       2009   Action 166112167 2012+2009            3
2         300       2006  Fantasy 210592590  300+2006            3
3          21       2008 Thriller  81159365   21+2008            3
4           9       2009  Mystery  31743332    9+2009            5
5        1408       2007   Horror  71975611 1408+2007            3
6          54       1998    Music  16574731   54+1998            4
```

Combined Dataset Example, with the mean ratings obtained from Kafka and the genre, BO figures obtained from MetaFlow

As can be seen MetaFlow can greatly enrich a recommendation system and provides the infrastructure to scale a system. Metaflow is fairly simple to install and use. Below is a code snippet where MetaFlow filters the movie by genre and adds a bonus movie from a different genre at the end.

```
# Print out the playlist and bonus movie.
end <- function(self){
    message("Playlist for movies in genre: ", self$genre)
    print(head(self$playlist))
    for (i in 1:nrow(self$playlist)){
        message(sprintf("Pick %d: %s", i, self$playlist$movie_title[i]))

        if (i >= self$top_k) break;
    }
}

metaflow("PlayListFlow") %>%
    parameter("genre",
              help = "Filter movies for a particular genre.",
              default = "Sci-Fi") %>%
    parameter("top_k",
              help = "The number of movies to recommend in the playlist.",
              default = 5,
              type = "int") %>%
    step(step = "start",
         r_function = start,
         next_step = c("pick_movie", "bonus_movie")) %>%
    step(step = "pick_movie",
         r_function = pick_movie,
         next_step = "join") %>%
    step(step = "bonus_movie",
         r_function = bonus_movie,
         next_step = "join") %>%
    step(step = "join",
         r_function = join,
         join = TRUE,
         next_step = "end") %>%
    step(step = "end",
         r_function = end) %>%
    run()
```

Code Snippet from the MetaFlow library

Run `python 01-playlist/playlist.py run` to generate a five-movie playlist of a random genre (you can also specify the genre as an option) with an added bonus track of a completely different genre because why not! (And you never know - you might have a new favorite movie voire genre!).

Five-movie Recommendation Playlist

Installation and Setup:

Metaflow can be simply installed using `pip3 install metaflow`

3. Strengths and limitations

Strengths:

One of the strong features of Metaflow is that it does extensive logging and versioning automatically. In a data science project, it is not rare that some parts fail and need debug. Metaflow makes debugging easier by saving all intermediate results without taking so much memory. In addition, it can also be helpful for updating our model because we have all the results we need to dig into the mistakes and understand what went wrong. More importantly, you can resume your work from any checkpoint that you have set. It can be extremely helpful in a data science project with a huge dataset. It is highly possible that in production, we have to spend two hours to just load in the dataset. If anything fails to proceed afterwards, we need to spend another two hours for the same loading task. However, with Metaflow, we can simply resume the job from wherever it fails.

In addition, Metaflow has built-in support for parallelism. We can easily fan out our graph by this built-in feature to save computational costs. For example, it is an unavoidable step to search for hyperparameters in any machine learning project. With Metaflow, data scientists can parallelize this step easily.

Limitations:

Since Metaflow is still in active development, the documentations and features are still limited. Also, Metaflow is not supported on Windows. It also does not currently have the support for Azure. Metaflow was released as an open source project by Netflix in December 2019. The idea first came up two years prior to Netflix's ML Infrastructure team. So, there is still a lot to learn and explore when it comes to using MetaFlow and potential vulnerabilities. It should be noted however, that Metaflow adoption has more than doubled at Netflix over the past two years.

4. References

[1] https://docs.metaflow.org/introduction/what-is-metaflow

[2] Metaflow by Netflix — the good, the bad and the ugly

[3] https://towardsdatascience.com/learn-metaflow-in-10-mins-netflixs-python-r-framework-for-data-scientists-2ef124c716e4