Machine Learning Infrastructure at Netflix

July 12th 2019
Savin Goyal
This is a high-level view of what Netflix does.
It is probably necessary to get **smarter** about everything:

- Content acquisition
- Marketing
- Discovery
- Delivery
- and more.

ML gets applied everywhere!
Reality is not that straightforward:

- How to run at scale?
- How to access data at scale?
- How to schedule the model to update daily?
- How to monitor models in production?
- How to debug failed production runs?
- How to iterate on new versions?
- How to collaborate with other users?
- ...
- ...
How much data scientist cares

- ML Libraries
- Feature Engineering
- Model Deployment
- Collaboration Tools
- Versioning
- Job Scheduler
- Compute Resources
- Data Warehouse

How much infrastructure is needed
Translate your domain knowledge to models with low cognitive overhead using tools you know.

Easy path from exploration to business value.

Collaboration.
Structure your code as a DAG.

It is a natural way to express ML pipelines.

Many technical benefits follow when you do this.
metaflow("BranchFlowR") %>%
step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
) %>%
step(
    step = "a",
    r_function = a,
    next_step = "join"
) %>%
step(
    step = "b",
    r_function = b,
    next_step = "join"
) %>%
step(
    step = "join",
    r_function = join,
    next_step = "end",
    join = TRUE
) %>%
step(
    step = "end",
    r_function = end
) %>%
run()
metaflow("BranchFlowR") %>%
step(
  step = "start",
  r_function = start,
  next_step = c("a", "b")
) %>%
step(
  step = "a",
  r_function = a,
  next_step = "join"
) %>%
step(
  step = "b",
  r_function = b,
  next_step = "join"
) %>%
step(
  step = "join",
  r_function = join,
  next_step = "end",
  join = TRUE
) %>%
step(
  step = "end",
  r_function = end
) %>%
run()
metaflow("BranchFlowR") %>%
  step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
  ) %>%
  step(
    step = "a",
    r_function = a,
    next_step = "join"
  ) %>%
  step(
    step = "b",
    r_function = b,
    next_step = "join"
  ) %>%
  step(
    step = "join",
    r_function = join,
    next_step = "end",
    join = TRUE
  ) %>%
  step(
    step = "end",
    r_function = end
  ) %>%
  run()
```r
metaflow("BranchFlowR") %>%
  step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
  ) %>%
  step(
    step = "a",
    r_function = a,
    next_step = "join"
  ) %>%
  step(
    step = "b",
    r_function = b,
    next_step = "join"
  ) %>%
  step(
    step = "join",
    r_function = join,
    next_step = "end",
    join = TRUE
  ) %>%
  step(
    step = "end",
    r_function = end
  ) %>%
run()
```

```r
test <- function(self) {
  self$my_var <- "hello world"
}
```
```r
metaflow("BranchFlowR") %>>%
  step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
  ) %>>%
  step(
    step = "a",
    r_function = a,
    next_step = "join"
  ) %>>%
  step(
    step = "b",
    r_function = b,
    next_step = "join"
  ) %>>%
  step(
    step = "join",
    r_function = join,
    next_step = "end",
    join = TRUE
  ) %>>%
  step(
    step = "end",
    r_function = end
  ) %>>%
run()
```

```
start <- function(self) {
  self$my_var <- "hello world"
}
```

```
a <- function(self) {
  message(
    "my_var is : ", self$my_var
  )
}
```
Execute as you would any R code

```
metaflow("BranchFlowR") %>%
  step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
  ) %>%
  step(
    step = "a",
    r_function = a,
    next_step = "join"
  ) %>%
  step(
    step = "b",
    r_function = b,
    next_step = "join"
  ) %>%
  step(
    step = "join",
    r_function = join,
    next_step = "end",
    join = TRUE
  ) %>%
  step(
    step = "end",
    r_function = end
  ) %>%
run()
```

Rscript debug.R
metaflow("BranchFlowR") %>%
  step(
    step = "start",
    r_function = start,
    next_step = c("a", "b")
  ) %>%
step(
  step = "a",
  r_function = a,
  next_step = "join"
) %>%
step(
  step = "b",
  r_function = b,
  next_step = "join"
) %>%
step(
  step = "join",
  r_function = join,
  next_step = "end",
  join = TRUE
) %>%
step(
  step = "end",
  r_function = end
) %>%
run()

Checkpointing by default

start A join B

Rscript debug.R resume
Minimize Cognitive Overhead
Keep using tools and libraries you are familiar with.

Dedicate your cognitive bandwidth on data science.

Metaflow stays out of your way.

No artificial limitations. Explore freely!

```r
fit_models <- function(self) {
  library(caret)
  param <- self$input
  train_control <- trainControl(
    method = "cv",
    number = 5
  )
  grid <- data.frame(
    interaction.depth = param$interaction.depth,
    shrinkage = param$shrinkage,
    n.trees = param$n.trees,
    n.minobsinnode = param$n.minobsinnode
  )
  x <- self$features
  y <- self$labels
  gbmfit <- train(
    x = x,
    y = y,
    method = "gbm",
    tuneGrid = grid,
    trControl = train_control,
    verbose = FALSE
  )
  self$model <- gbmfit$finalModel
  self$fit <- gbmfit$results
}
```
Keep using tools and libraries you are familiar with.

Dedicate your cognitive bandwidth on data science.

Metaflow stays out of your way.

No artificial limitations. Explore freely!
Keep using tools and libraries you are familiar with.

Dedicate your cognitive bandwidth on data science.

Metaflow stays out of your way.

No artificial limitations. Explore freely!

```r
fit_models <- function(self) {
  library(caret)
  param <- self$input
  train_control <- trainControl(
    method = "cv",
    number = 5
  )
  grid <- data.frame(
    interaction.depth = param$interaction.depth,
    shrinkage = param$shrinkage,
    n.trees = param$n.trees,
    n.minobsinnode = param$n.minobsinnode
  )
  x <- self$features
  y <- self$labels
  gbmfit <- train(
    x = x,
    y = y,
    method = "gbm",
    tuneGrid = grid,
    trainControl = train_control,
    verbose = FALSE
  )
  self$model <- gbmfit$finalModel
  self$fit <- gbmfit$results
}
```
Focus on the following:

- Feature engineering.
- Training logic.
- Format of the output.
Metaflow takes the pain away from distractions like:

- Scalability.
- Scheduling.
- Operations.

```r
library(metaflow)

start <- function(self) {
  self$x <- c(10000, 40000, 80000)
}

a <- function(self) {
  x <- self$input
  big_matrix <- matrix(rexp(x*x), x)
  message(sum(big_matrix))
}

metaflow("BigSumFlowR") %>%
  step(
    step = "start",
    r_function = start,
    next_step = "a",
    foreach = "x"
  ) %>%
  step(
    step = "a",
    r_function = a,
    next_step = "join"
  ) %>%
  step(
    step = "join",
    next_step = "end",
    join = TRUE
  ) %>%
  step(
    step = "end"
  ) %>%
  run(meson = "create")
```
First Class Collaboration
I want to collaborate with other people but I don’t want to think about it all the time!

- Everything is versioned.
- Everything can be tagged with human-readable annotations.
- All data artifacts are stored.
- Easy access to all code, data, & results.
Monitor models and examine results

```
library(metaflow.client)

housing_flow <- flow$new("HousingFlow")
summary(housing_flow)

latest_successful_run <- housing_flow$run(housing_flow$latest_successful_run)
summary(latest_successful_run)

--- Flow Summary: HousingFlow
Created At: 2018-04-25 15:39:46
Latest Run: 185
Latest Successful Run: 185
Runs: 92

--- Run Summary: 185
Successful: TRUE
Created at: 2019-01-17 00:21:08
Finished at: 2019-01-17 00:30:13
Time: 9.08 mins

score_data_step <- latest_successful_run$step('score_data')
summary(score_data_step)

--- Step Summary: score_data
# Tasks: 1
Created at: 2019-01-17 00:29:08
Finished at: 2019-01-17 00:29:40
Time: 32 secs
```
Reproducible Research
Metaflow solves the practical problem of being able to run the script again by:

- Storing **immutable snapshots** of code and data.
- Managing **external dependencies**.
- Maintaining a **detailed audit log** of all past runs, both during development and in production.

Gets us pretty close to the holy grail of reproducible research.
Many more features -

Supports **Python** and **R**
High-throughput **data access**
Deploy models as **microservices**
Flexible **parametrization** of workflows
Isolated environments for **dependencies**
**Slack** bot
And more!
Thank you!

savin@netflix.com

PS: We’re hiring!