Large-scale reproducible experiments

Antonin Voyez[1, 2, 3],

[1]Univ Rennes, CNRS, IRISA, France [2]ENEDIS, France [3]Inria, IRISA, France

Summary

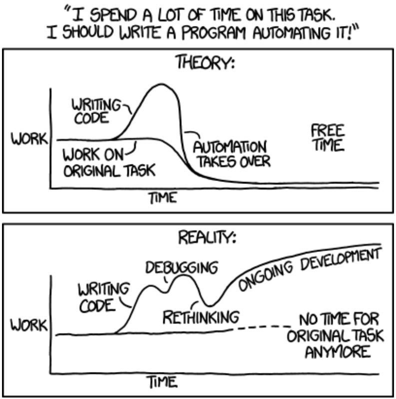


Figure: https://xkcd.com/1319/

The objective

- Experiment task:
  - Train a classifier.
  - Run an optimization solver.
- Explore multiple parameters:
  - Datasets (+ transformations).
  - Classifiers (+ hyperparameters).
- Analyze the results
- **Fast:**
  - Performing multiple experiments campaigns before the end of the Ph.D.
- **Reproducible:**
  - Deterministic: Same parameters = same results.
  - Easy to share the code and the experiments.
  - Should be executable in the future (10 years).

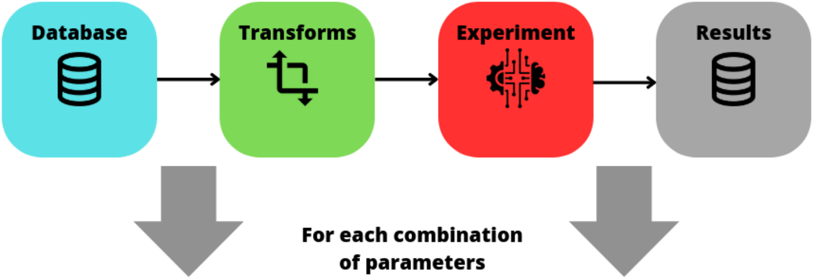
Concretely

- Task: Train a time series classifier
  - 20 classifiers and hyperparameters (testing)
  - 4 classifiers (production)
- Datasets:
  - Small ones (4500 series, 17000 timestamps, approx. 2.7 Gb)
  - Big one (3M series, 34000 timestamps, approx. 3.6 Tb)
- Parameters:
  - 3 layers, 1 to 100 parameters each.
- Total:
  - Approx. 1800 experiments per campaign.
  - 4 main campaigns.
- `https://gitlab.inria.fr/avoyez1/mia_stats`

Available resources

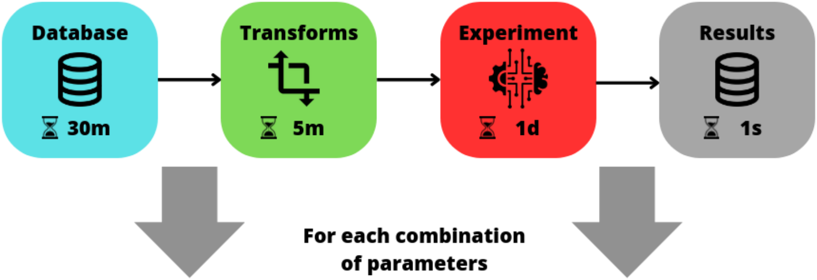

- One computer:
  - PC (low computational power).
  - Server (high computational power).
- Computation grid:
  - Multiple computers sharing a common file system.
  - Various computational power and systems within the grid.
  - IGRIDA / Grid 5000 / OAR.
  - Distributed task workers (Ray).

Naive solution: pipeline



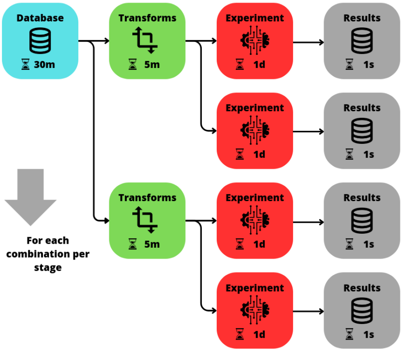
- Pipeline: a set of stages (operations on the data).
- Perform each pipeline stage for each combination of parameters.

Naive solution: problems



- Perform the same task multiple times.
- Bottlenecks (here: loading the data & performing the experiment).
    - Computation / IO (disk).
- More than a day per pipeline = problems

Tree based solution



- Represented as a tree (set of directories).
- Each parameter is computed once.
- Requires to store intermediate values.
  - Extra RAM or disk usage.

Implementation

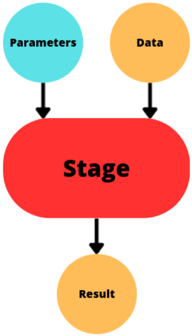
- Experiments descriptor
- Stage
- Orchestration
- Analyzing the results

Building the tree: descriptor

```
seed = 0

# path_data : Path of the train dataset file
train_data_path = "../../data/30_issda.parquet"
# remove_cols : Columns to remove to get a clean ready to use dataset
train_data_cols = ["d"]

# agg_sizes : Aggregate sizes to test.
agg_sizes = [500]#, 10, 50, 100, 1000, 2000]

# agg_types: Aggregate types to test. Possible values: ["sum", "mean"]
agg_types = ["mean"]#, "sum"]

# Set of targets to attack (as columns of targets datasets)
targets = ['1158']#, '2051', '2687']

# Train series subsets
tts_subsets = [
    #(TtsSubseries(0, 2), TtsSubseries(0, 2)),
    (TtsSubseries(0, 500), TtsSubseries(0, 500), TtsSubseries(0, 500))
]

# Protection methods to test
protections = [
    NoneProtect(),
]

# Classifiers
classifiers = [
    #RocketClassifier(num_kernels=10_000, seed=seed),
    #MiniRocketClassifier(num_kernels=10_000, seed=seed),
    LRClassifier(seed=seed),
]
```

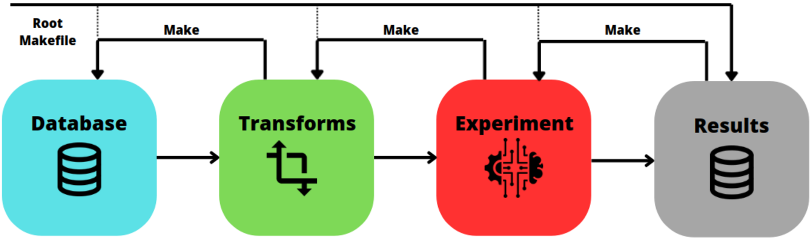
- File describing the experiment campaign.
- Parameters to test.
- Build the experiment tree.
- Reproducibility: from this file, everything is deterministic.

Stage program



- Each stage is a separate program (Python).
- Each stage takes as input a parameter file and outputs a result file.
- Deterministic
- Normalized:
  - Parameters: ./params.yml
  - Data: ../out
  - Result: ./out

Orchestration



- Using Makefiles (Linux Native, Parallelizable)
- Each node has its own Makefile building the node, starting from the leaves.
- A root Makefile pilot the nodes Makefiles

Stage Makefile

```
out.npz: params.yaml ../out.npz
    # be process safe
    @if mkdir data.lock 2>/dev/null; \
    then \
        trap "rm -rf data.lock" 3; \
        protect params.yaml ../out.npz out.npz; \
        result=$$?; rm -rf data.lock; exit $$result; \
    else \
        while test -d data.lock; do \
            sleep 1; \
        done; \
        test -f out; \
    fi;


../out.npz:
    @if test -f $@; then :; else \
        $(MAKE) -C .. ; \
    fi;
```

Atomic

Stage

Wait for lock

Requirements not found: build top level

- The Makefile performs the stage action if the output file does not exist.
- If the node requirements are not found, execute the top-level Makefile.

The root Makefile

```
all: mia

mia: tree/mean/tts/LR/out.csv
tree/mean/tts/LR/out.csv:
	$(MAKE) -C tree/mean/tts/LR

tts: tree/mean/tts/out.npz
tree/mean/tts/out.npz:
	$(MAKE) -C tree/mean/tts

agg: tree/mean/out.npz
tree/mean/out.npz:
	$(MAKE) -C tree/mean


cleanall: cleanlock cleandata cleanres
cleanres:
	find . -name '*.csv' -delete;
cleandata:
	find . -name '*.npz' -delete;
cleanlock:
	find . -name '*data.lock' -delete;
```

- User interface to pilot the experiments.
- Reference all the leaves and stage's nodes as targets.
- Perform cleanup operations.

Results analysis

- Jupyter
  - Interactive Python web-platform
- Matplotlib / Seaborn
  - Graph generating libraries.
  - Can save figures in .pdf format (vectorized image)
- Script to fetch and merge all individual CSV in one:

```
find ./tree -type f -name '*.csv' -exec cat {} \; > res.csv
```

Reproductibility

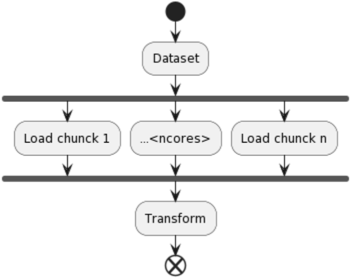
Seed:

- Fixing the random number generator (RNG) state.
- The RNG will become deterministic produce the same sequence of "random" values each run.
- The seed should be fixed for each node.

Packaging:

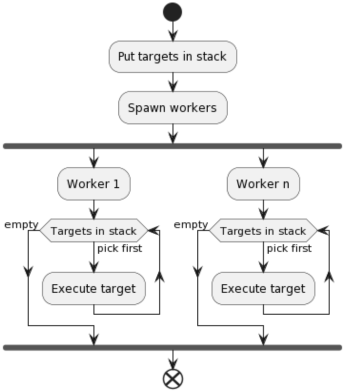
- Poetry
  - Python packaging tool
  - Keep trace of libraries versions
  - Deploy the package with pip install
  - Able to define new linux commands from the source code
- Docker / Singularity
  - Execution environment
  - Isolated from the system (OS versions)

Parallelism: within a program



- Modin
  - Pandas-like (only change the import)
  - Parallelize pandas operations
  - Able to distribute computation on a cluster
- Joblib
  - Parallelize for loops

Parallelism: multiple programs



- Put all targets in a stack
- Each worker:
  - Pull a target from the stack and execute it until the stack is empty.
- Make
  - Make -J <ncore>: Execute <ncore> targets in parallel.
  - Single computer (worker = 1 core)
- OAR
  - Minionize: Python library to manage the stack
  - Requires to extract the targets from the root makefile to the array param file.

Data pre-processing is important

- Data pre-processing
  - Transform the data (Shape, format, typing) to improve disk usage and loading time.
  - Clean missing/invalid values.
  - Should be part of the experiment pipeline.
- Example: Time series
  - "Classical" format: (individual ID, timestamp, value), CSV.
  - My format: Timestamp x Individual matrix, Parquet.
  - Sampling the database: full Enedis DB: 1.8 T (classical, 3M series, 2a)
  - x100 in disk usage and loading time (20 Go to 200 Mo).

Data formats

- SQL:
  - Handle complex data structures.
  - Can store and query a large amount of data.
  - Extracting large dataset can be slow (pandas.read_sql).
- CSV:
  - Heavy disk usage.
  - Slow to read/write.
  - Human readable.
- Pickle
  - Can store anything Python (beware of security issues).
  - Slow to read/write (faster than CSV).
- Parquet
  - Data stored as typed binary vectors.
  - Fasted tabular data format.
  - Lower disk usage
- RAM files
  - /tmp, /dev/shm: The files are stored in RAM
  - Same IO functions as files
  - Not persistent/shareable between computers

The end

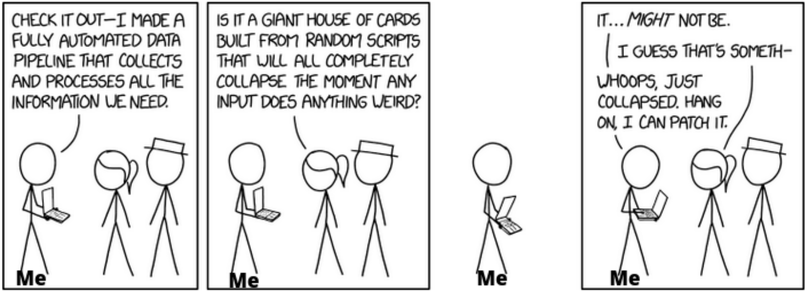


Figure: https://xkcd.com/2054/