TecAlliance

Experimentation Framework

Presentation and Discussion

Dr. Klaus Varrentrapp

Rational

- Systematic experimentation of proof of concepts
  - Needs to be
    - Reproducible
    - Reconstructible
  - Begins earlier than you think
    - Latest when you tune the first hyperparameters
    - After your first initial trials about the approach such as which method to use, which (network) architecture to use, …

Beware of silent bugs which only occur in production!

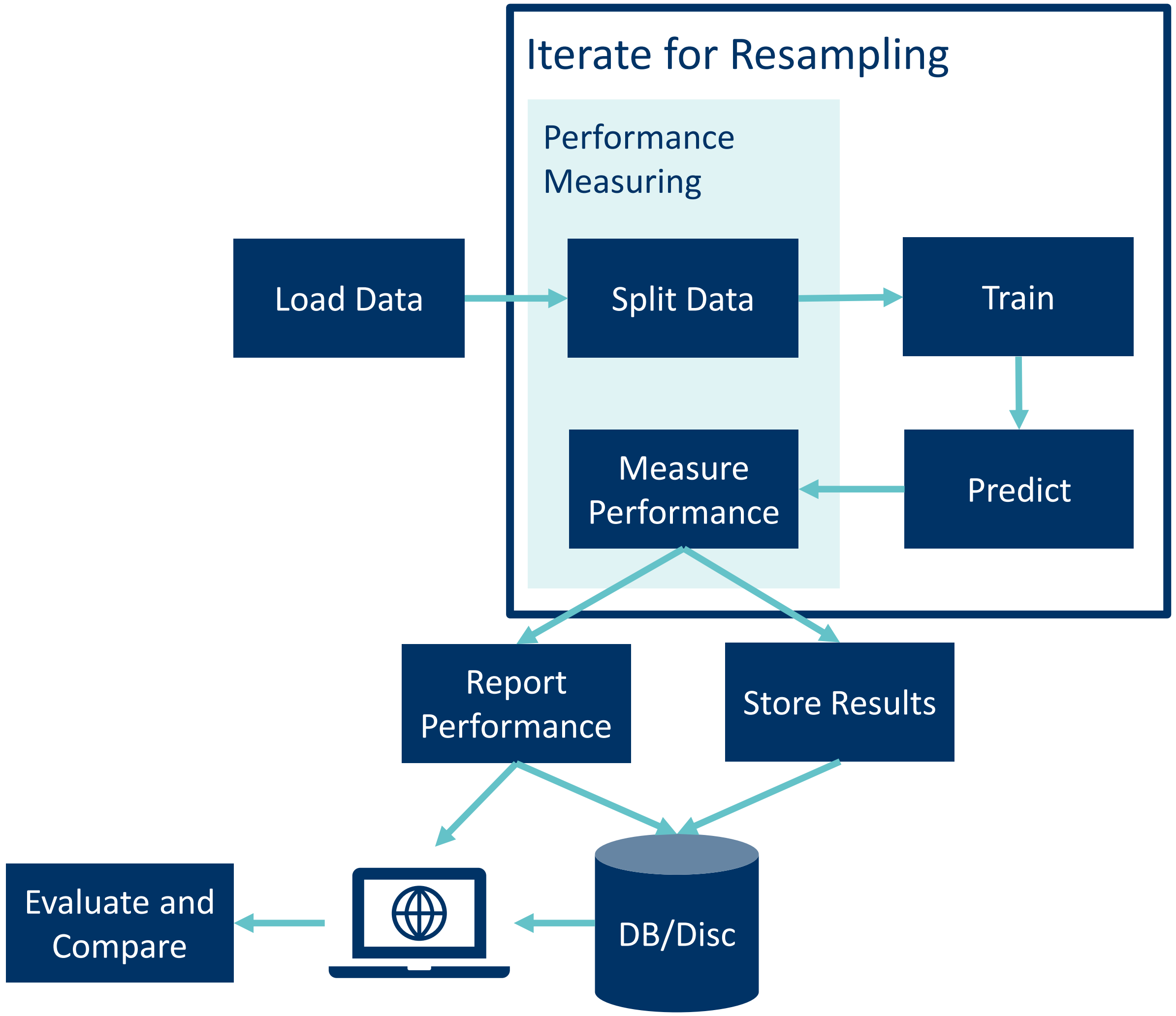
- Reproducibilty and reconstructability require:
  - Having an overview over the experiments conducted and the major performance measurements and being able to compare them
    - Already at the point of construction/configuration
  - To be able to rerun an experiment and understand its details, e.g. by having declarative configurations (files)
  - Store (intermediate) data, logs, models, and human readable reports systematically

TecAlliance

Existing Tools for Experimentation Support

- Cloud: AWS SageMaker, Azure ML Service
- Open Source: Sacred, DVC, MLFLow, GuildAI, Pachyderm
- Commercial: Comet, Neptune.ai, MissingLink, Weights & Biases, SigOpt, DotScience
- Offer:
  - Rerun of experiments
  - Organization and analysis of results
  - Deployment and monitoring
- What‘s missing everywhere?
  - Organization of complex experiment configurations
    - Only indirectly and partially via result overviews
  - Built-in declarative, easily human-readable experiment configuration to enable robust experiment set-up and details drill-down
    - Complex command line calls or full-blown notebooks as required by all tools are tedious and error-prone

Simplified General Machine Learning Workflow



Comments

- Splitting the data (in train, validation, test, …, sets) and measuring the performance in combination have to be adjusted to your use case
- Resampling: Bootstrapping, Cross-Validation, etc.

=> ML experiment configuration is complex with many hyperparameters to set

Decision: Implement Lightweight Framework

- Fixed workflow enables lightweight IoC (Inversion of Control) and DI (Dependency Injection)
  - Add interfaces for the most important parts of the workflow (including interfaces/defaults implementations for data)
    - Implementations of these become the arbitrary combinable building blocks
  - Orchestrate (connect) according to workflow providing basic, unchangeable framework
- => Component-based Experimentation Framework:
  - Enables systematic and declarative configuration (e.g. via human-readable configuration files)
    - Create new experiment configurations by altering old ones (human-readable => less error prone)
  - Enables integration of shared services such as result directory creation, reporting (with the ReportCollector tool), logging, restart, timing, etc.
  - Saves (boilerplate) code
  - Greatly supports QA, e.g.:
    - Smoke test configs
    - Code reviews by means of debugging smoke tests (and interactively explore used data structures, etc.) via an IDE

Python Implementation

- Read config file in JSON with structure analog to workflow and with global default mechanism
  - Specify for each interface which implementation to use and a dictionary of variable and implementation specific (hyper) parameters
- Use `import_module` and `getattr` to load the implementations specific classes and provide parameters via `**kwargs` mechanism (more elaborate version used as `FromParams` in allennlp.org)
- Define fixed structure for storing configuration files and experiment results
  - config
  - result
    - result_experiment_N
      - data => All (intermediate) data, can be reused/referenced by subsequent experiments
      - log => Logs, configuration (file), environment used, etc.
      - model => Computed and saved models, can be reused/referenced by subsequent experiments
      - report => Human readable reports, e.g. in HTML format as produced by the ReportCollector tool

TecAlliance

Reporting

- Use Sacred (https://github.com/IDSIA/sacred) to store logs, results, reports, and other artefacts in a central (MongoDB) database
  - Done via easy code instumentation, mostly done within the framework code itself
- For each experiment an ID, name, main results (performance measurements), and other automatically added information is stored
- Visualize using the web-based tool Omniboard
  - Tabular overview over all experiments and sortable according to any column/information source such as execution time, performance measurements, grouping, etc.
  - Drill-down view of individual experiments with more information
- Both Sacred and the MongoDB are Docker empowered
  => Easy to use locally also

Still missing

- Support for resampling
- Hyperparameter search => Systematic generation of experiment configurations

Why we do not use …

- Jupyter Notebooks
  - We do, but only for initial trials, not for systematic experimentation
  - See here https://docs.google.com/presentation/d/1ivK8AKgz8Hx-ZYzPC9gJyQK6tzuhR3UuhCEajFGJDlA/edit#slide=id.p
  - Basically: Notebooks encourage bad (software engineering) practice and are hardly reproducible
    - https://docs.google.com/presentation/d/17NoJY2SnC2UMbVegaRCWA7Oca7UCZ3vHnMqBV4SUayc/edit#slide=id.p
  - nbdev (http://nbdev.fast.ai/) does not convince us
    - Poor documentation
    - Too much overhead, hardly any savings to applying best software engineering practices
    - Unit test support insufficient

Thank you for your attention!

TecAlliance